DataFunTalk 2019年度技术应用文章合集

各厂最新技术应用分享

全集

算法工程师必备！

本书只限DataFunTalk的小伙伴们学习使用

如有其它平台发送，欢迎举报。

DataFunTalk：专注于大数据、人工智能技术应用的分享与交流

关于 DataFunTalk：

DataFunTalk 专注于大数据、人工智能技术应用的分享与交流，发起于 2017.12，至今已在全国 7 个数据智能企业和人才集中的城市举办了近 100 场线下技术分享和两场千人规模的行业峰会，邀请近 400 位工业界专家和 40 位知名学者参与分享，近 30000 人次从业者参与线下交流。合作企业包含 BAT 等大厂，以及知名互联网公司和数据智能创业公司。其运营的公众号 DataFunTalk 共生产原创文章近 300 篇，超百万阅读，4 万+精准粉丝。

声明：

本电子书，系 DataFunTalk 原创文章合集整理，仅限 DataFunTalk 的小伙伴们收藏学习，如有其它渠道未经允许擅自发送本资料，欢迎小伙伴举报！

联系方式：hongfei@datafun.ltd

以下，Enjoy~~

目录

推荐篇

推荐篇从推荐推理奔向未来 AI…… 9

达观数据个性化推荐系统实践…… 39

Embedding 技术在民宿推荐中的应用…… 59

基于行列式点过程的推荐多样性提升算法……79

网易新闻推荐：深度学习排序系统及模型……94

低频少样本长验证周期场景下的算法设计…… 108

阿里零售通智能导购推荐技术实践…… 122

推荐技术随谈……137

京东电商推荐系统实践……151

阿里妈妈深度树检索技术（TDM）及应用框架的探索实践…… 170

快看漫画个性化推荐探索与实践…… 197

浅谈 UC 国际信息流推荐…… 219

UC 信息流视频标签识别技术…… 231

UC 信息流推荐模型在多目标和模型优化方面的进展…… 252

深度学习技术在美图个性化推荐的应用实践…… 272

推荐系统中模型训练及使用流程的标准化…… 299

如何从 0 到 1 构建个性化推荐？……319

RALM: 实时 Look-alike 算法在微信看一看中的应用……344

广告篇

阿里妈妈：电商预估模型的发展与挑战…… 372

深度学习在 360 搜索广告 NLP 任务中的应用……385

Hulu：视频广告系统中的算法实践…… 400

微博广告策略工程架构体系演进…… 421

AI 在爱奇艺视频广告中的探索……446

OCPC 广告算法在凤凰新媒体的实践探索…… 462

阿里妈妈：品牌广告中的 NLP 算法实践……483

爱奇艺效果广告的个性化探索与实践…… 501

360 展示广告召回系统的演进…… 513

解密商业化广告投放平台技术架构…… 532

优酷 DSP 广告投放系统架构实践……565

阿里妈妈点击率预估中的长期兴趣建模…… 582

自然语言处理篇

NLP 在 Hulu 的研究与实践……608

人机交互式机器翻译研究与应用…… 644

百度中文纠错技术…… 667

阿里小蜜：智能服务技术实践及场景探索…… 682

阿里小蜜中的情绪回复能力…… 699

通用领域对话问答…… 712

美团对话理解技术及实践…… 727

智能机器人在滴滴出行场景的技术探索…… 748

Hulu 在视频内容理解上的应用和最佳实践 762

对话智能：国际视角，国内形势及案例学习 784

云小蜜：在中国移动的落地实践 803

阿里小蜜：知识结构化推动智能客服升级 822

阿里云小蜜对话机器人背后的核心算法 854

UC 国际信息流推荐中的多语言内容理解 876

用人工智能技术打造虚拟生命 895

人机对话关键技术及挑战 941

基于机器阅读理解（MRC）的信息抽取方法 965

基于知识图谱的问答在美团智能交互场景中的应用和演进 995

人机语音对话技术在 58 同城的应用实践 1024

人机对话技术研究进展与思考 1045

腾讯信息流内容理解技术实践 1081

知识图谱的自动构建 1097

知识结构化在阿里小蜜中的应用 1122

NLP 技术在微博 feed 流中的应用 1147

机器学习篇

阿里新零售中的智能补货（Ⅰ）— 库存模型 1175

新零售中的智能补货（Ⅱ）— 需求预测 1188

绿湾机器学习平台（AutoML）打造与落地实践 1199

新一代联邦学习技术及应用实战 1217

AutoML 在表数据中的研究与应用……1250

强化学习：原理与应用……1274

强化学习在滴滴车服用户运营中的实践……1303

小米移动搜索中的 AI 技术……1316

搜索引擎中的 web 数据挖掘……1331

利用高效率的补贴策略驱动用户增长……1343

风控篇

携程金融大数据风控算法实践……1362

金融风控反欺诈之图算法……1376

风险价值量化术—互金爆发期的风控技术实践……1397

音频内容理解的关键技术……1424

汽车金融风控流程设计和机器学习实践……1434

机器学习在信贷风控建模中的优势和挑战……1463

风控建模流程：以京东群体感知项目为例……1481

信贷业务风控策略简介……1503

度小满智能获客系统实践……1517

机器学习在反欺诈中应用……1527

自动驾驶篇

自动驾驶从 0 到 1 之路……1539

自动驾驶中轨迹规划的探索和挑战……1559

L4 自动驾驶中感知系统遇到的挑战及解决方案……1577

高精地图和定位在自动驾驶的应用……1593

Pony.ai 的基础架构挑战与实践……1619

在自动驾驶研发中充分发挥数据的潜能……1633

传统方法和深度学习结合的感知策略探索……1646

自动驾驶复杂环境下高精度定位技术……1662

自动驾驶多传感器感知的探索……1678

“老司机”成长之路：自动驾驶车辆调试实践……1702

安全舒适两不误：探讨优化算法在规划控制中的应用……1718

打造最可靠的自动驾驶基础架构……1735

从硬件角度剖析自动驾驶，为什么说它是复杂的系统工程？……1757

大数据篇

Sophon ：Hulu 智能 OLAP 缓存层技术实践……1773

举重若轻的人人车移动端数据平台……1789

时序数据在滴滴实时数据开发平台中的处理和应用……1808

阿里巴巴双十一千万级实时监控系统技术揭秘……1819

深入了解 TiDB SQL 优化器……1835

Druid SQL 和 Security 在美团点评的实践……1853

网易大数据体系之时序数据技术……1873

快手 Druid 精确去重的设计和实现……1885

基于 Kafka 的实时计算引擎如何选择？Spark or Flink ？……1910

万亿数据下 Hadoop 的核心竞争力……1923

Apache Doris：一个开源 MPP 数据库的架构与实践……1935

TiDB 的 HTAP 之路：过去，现在和将来……1957

快手 HBase 在千亿级用户特征数据分析中的应用与实践……1976

贝壳：流式数据的平台化实践与挑战……1992

Apache Beam 架构原理及应用实践……2012

阿里小蜜新一代智能对话开发平台技术解析……2050

构建端到端的联邦学习 Pipeline 生产服务……2067

Hadoop or TDengine，如何做物联网大数据平台的选型？……2100

实时计算引擎在贝壳的应用与实践……2120

通用高效的数据修复方法：Row level repair……2139

Cassandra 在 360 的实践与改进……2156

Apache Dubbo Roadmap 2019……2175

推荐篇

从推荐推理奔向未来 AI

嘉宾：徐潇然 编辑：Hoh

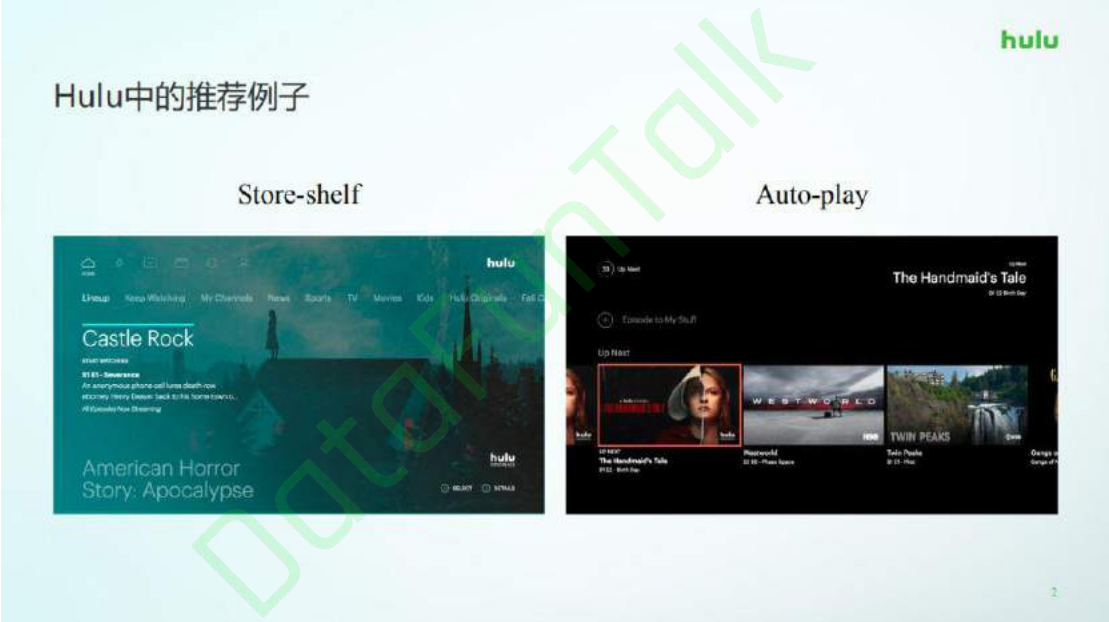
看到题目，很多人会问：推荐为何需要推理？推荐推理和 AI 有何关系？为什么说是未来的 AI？所以，这里我要做的，不仅是回答什么是推荐推理、为什么要推荐推理，还要讲清楚它和未来 AI 的关系，而且后面这部分才是重中之重的。

什么是未来 AI？当然是强 AI，或者称通用 AI。我们不会希望十年后的 AI 产品，仍局限于人脸识别、物体检测、语音识别等。为了实现强 AI，我们决不能仅满足于当前 AI 所取得的效果，而要大胆地 think out of box，想象并描绘你心中的强 AI 应有的特征。对于我来说，强 AI 应该具备这样的能力：推理能力、类比能力和归纳能力，从具体的现象中抽象出常识性的知识和法则，并运用之去指导具体的实践，还有想象力和创造力，最后是意识。对，这些能力都是我们人类具有的能力，作为宇宙中目前仅知的智慧体，我们不希望自己是孤独的，也不要傲慢地认为自己是唯一的那个特例。我们如果相信人类所代表的智慧具有某种普遍真理，为什么不认为未来 AI 也必然走上这一道路呢。

我们需要首先建立信心。对于强 AI 这个看似遥远的目标，我们其实可以从当下飞速进展的 AI 研究前沿获得启示。我们要去翻阅近年 AI 顶会 NeurIPS、ICML 和 ICLR 的论文，要去追踪顶级研究机构 DeepMind、MILA、FAIR 和 Vector Institute 等的工作，要去聆听活跃在研究一线的大佬们发言，然后结合自己手头的工作，更要超越手头的工作，脚踩大地、仰望星空，才能将隐藏在千头万绪中的启示，在经过我们深思后，归纳出或将触及未来终极人工智能的“道”之一。

招式万变，心法恒一。在强 AI 的层面上思考，不是“技”或“术”能解决的，不是要讨论用一个什么新的神经网络层，或是换一个什么新的目标函数，而应该从“道”这个更本质的层面去探索破解 AI 的秘密。

这里的“推荐推理”，是我思考的起点而不是终点，我所分享的也不是在 Hulu 如何具体做推荐推理，也就是说，任何人可以选择不同的起点去思考去探寻未来 AI 之路，但是我们仍然可以殊途同归。



先看两个例子，这是 Hulu 推荐系统在产品端实现的具体形态。左边的图为“store-shelf”，不同栏目下有针对用户的个性化推荐结果，包括电视剧、电影、脱口秀等；右边的图称为“auto-play”，也是个性化推荐的结果。事实上，抛开 Hulu 的例子，纵观国内外的互联网和移动互联网产品，推荐系统的形态基本大同小异，就是针对不同的用户或用户群，给出分数从高到低的推荐排序，取 Top-K 的项展示出来，其

中评分的过程有受纯算法控制的部分，也有受人为策略干预的部分。然而，无论这些系统呈现出如何的花样，本质始终是一个“货架”。

推荐系统的意义，是从仓库货架到个性化货架，因为展现给用户的货架大小有限，所以必须把用户最可能购买或最可能点击的放在里面。于是，每一个推荐系统的算法工程师，只关心一件事情——如何提高点击率（CTR）。如果我是一名推荐算法工程师，终有一天，我会感到厌烦，停止手下的键盘敲击，抬起头，重新审视推荐这项工作，我会疑问：推荐的本质难道就是从货架到个性化货架吗？如果这不应该就是推荐的本质，那么这可能就是一个束缚我们的程式。

我们回到前互联网时代的商业社会，当提到“推荐”二字时，我们一定会想到“销售”。我们不妨做一下类比，在电子商务未出现的年代，设计一个线下纯人工的“推荐系统”！假想你是一个商店的店员，你的老板想卖出更多的货，他要你想办法。这时，你会根据自己的销售经验，把用户可能最需要的商品列出一个清单，把它们一一摆在前排或者刚进商店门的位置，并尽可能地做一些吸引眼球的促销广告，留住你的顾客，引导他们去购买，即使不买也多来转转提升人气，就像互联网上增加流量。

既然我们谈到了“销售”，我们不妨从销售的角度出发，来重新思考推荐。一个店员，如果仅仅摆弄货架，不会招揽顾客，那么他的老板一定不会高兴。市面上有很多讲“销售艺术”的文章，我摘录了几条：

1. 内容搭建关系，关系基于信任，信任驱动收益

2. 你的顾客不会为了支持你的公司而购买你的产品；他们购买你的产品只因为你的产品让他们的生活变得更好。因此，你想卖出去点什么，你需要解释你为什么能帮到他们。

3. 如果你有一个好的故事要讲，那么尽量用一种真实的口吻和你的顾客对话，而不是去说教。你需要吸引他们，他们才会去听、去学、去信任你。

4. 销售员可以花上大量精力来促销一款产品，但是除非顾客主动地来相信你讲的故事，什么事情都不会发生。

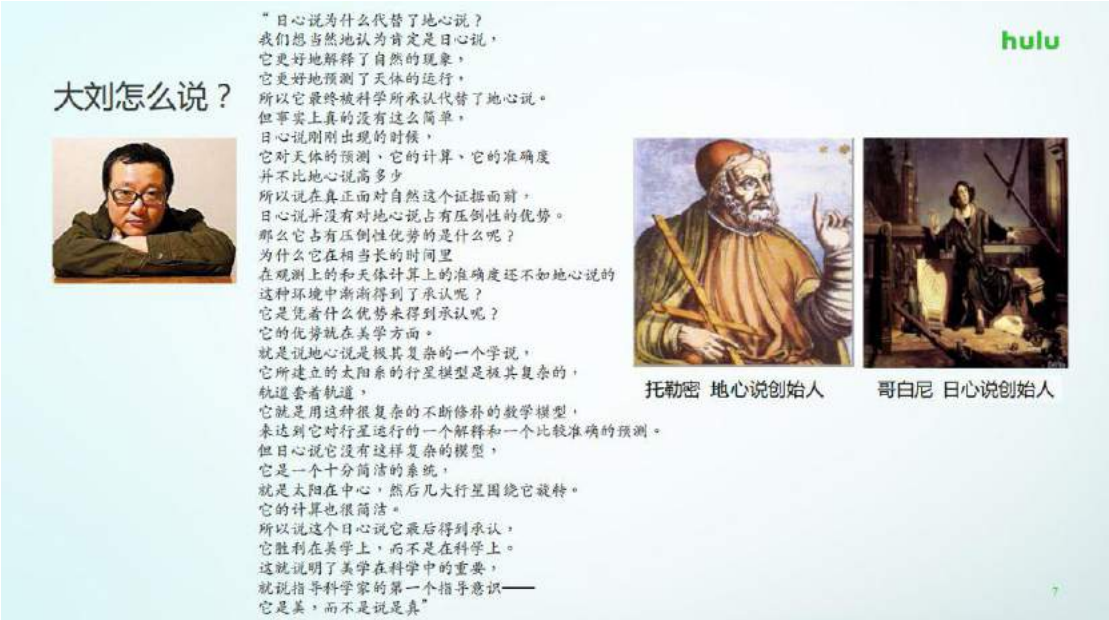
5. 悖论往往是，你提供的信息越多，越多的人会来购买。

总之，这里没有一条教你如何摆货架，都在教你：如何讲好产品的故事？如何解释给消费者这款产品能够帮助他们？如何让你的顾客相信你的故事从而更愿意买你的产品？回到推荐系统上，想想这里到底缺失了什么？

我不妨再举一个例子，就是人类社会最古老的销售——销售信仰。设想一个摆满各类宗教的货架，我们要为它设计一个推荐系统，货架上根据不同地区不同用户群的口味排列好宗教产品。你觉得怎样？是不是太荒唐了，哪里有人这么干。事实上，为了让用户购买“信仰上帝吧”，需要解释“上帝是谁”、“为什么要信仰上帝”以及“如何信仰上

帝”，于是有了圣经，有了亚当和夏娃的传说，基督教的神学体系构建起来，然后还要传教士们不厌其烦地去传教，尽可能地说服他见到的每一个人去信仰上帝。其他宗教，大致也是这样。因此，一个好的推荐系统不应当局限于摆货架，还应当扮演起一个好的传教士，一个好的推销员。总结起来，一句话——提供解释，说服别人。

什么是“解释”？解释，一端是被解释的客观事实，一端是解释给的主观个体，这构成了解释在功能上的两重性：客观性和主观性。客观性，从解释某个事实的角度出发，需要解释在某种程度上能被客观事实检验，不仅能回答“为什么过去是这样”，还能预测“未来是什么样”；主观性，从让人理解并接受的角度出发，解释本身要简洁、易懂、自洽，甚至某种意义上，解释不一定要吻合真相，因为很多情况下真相是不可知的。如何理解解释的两重性？这里有一个例子——股票评论员，股票评论员说的不见得符合事实本身，很多情况下他确实不知道幕后真相，但是这不妨碍他提出合理的解释并让听众接受。



有一次听刘慈欣讲科幻，在谈到“日心说”和“地心说”时，他说了这么一段话让我印象很深刻：

“日心说为什么代替了地心说？

我们想当然地认为肯定是日心说，

它更好地解释了自然的现象，

它更好地预测了天体的运行，

所以它最终被科学所承认代替了地心说。

但事实上真的没有这么简单，

日心说刚刚出现的时候，

它对天体的预测、它的计算、它的准确度

并不比地心说高多少

所以说在真正面对自然这个证据面前，

日心说并没有对地心说占有压倒性的优势。

那么它占有压倒性优势的是什么呢？

为什么它在相当长的时间里

在观测上的和天体计算上的准确度还不如地心说的

这种环境中渐渐得到了承认呢？

它是凭着什么优势来得到承认呢？

它的优势就在美学方面。

就是说地心说是极其复杂的一个学说，

它所建立的太阳系的行星模型是极其复杂的，

轨道套着轨道，

它就是用这种很复杂的不断修补的数学模型，

来达到它对行星运行的一个解释和一个比较准确的预测。

但日心说它没有这样复杂的模型，

它是一个十分简洁的系统，

就是太阳在中心，然后几大行星围绕它旋转。

它的计算也很简洁。

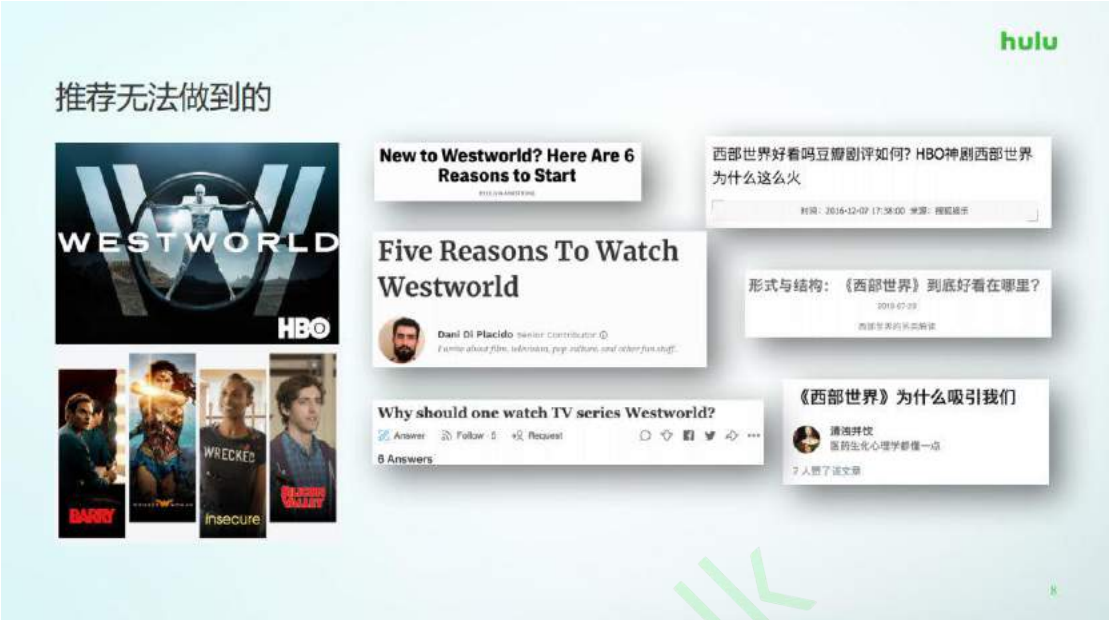
所以说这个日心说它最后得到承认，

它胜利在美学上，而不是在科学上。

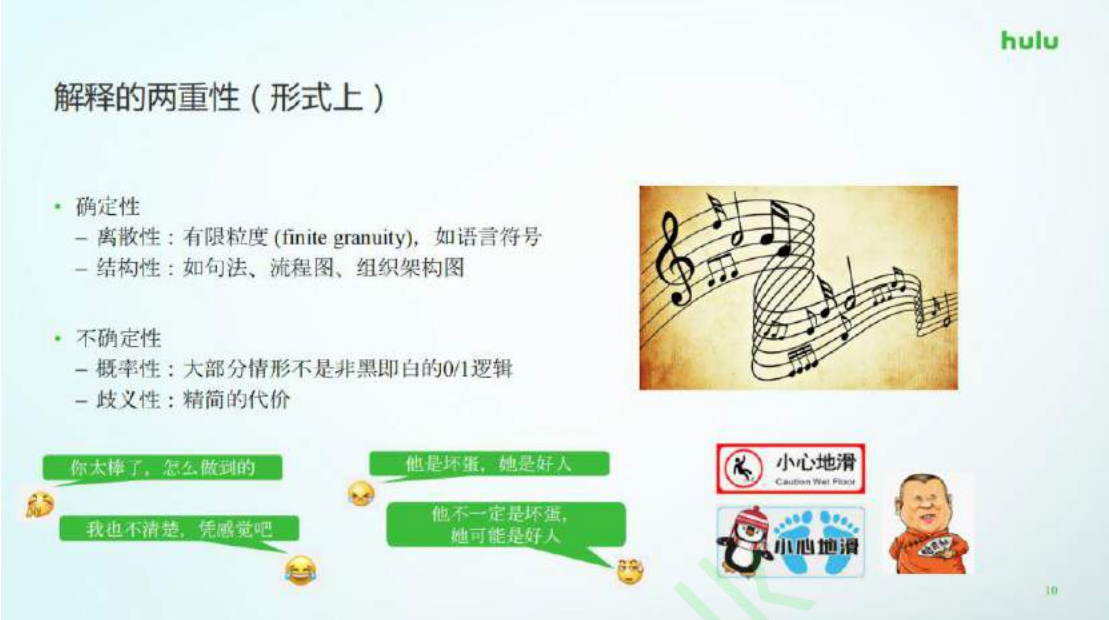
这就说明了美学在科学中的重要，

就说指导科学家的第一个指导意识——

它是美，而不是说是真”



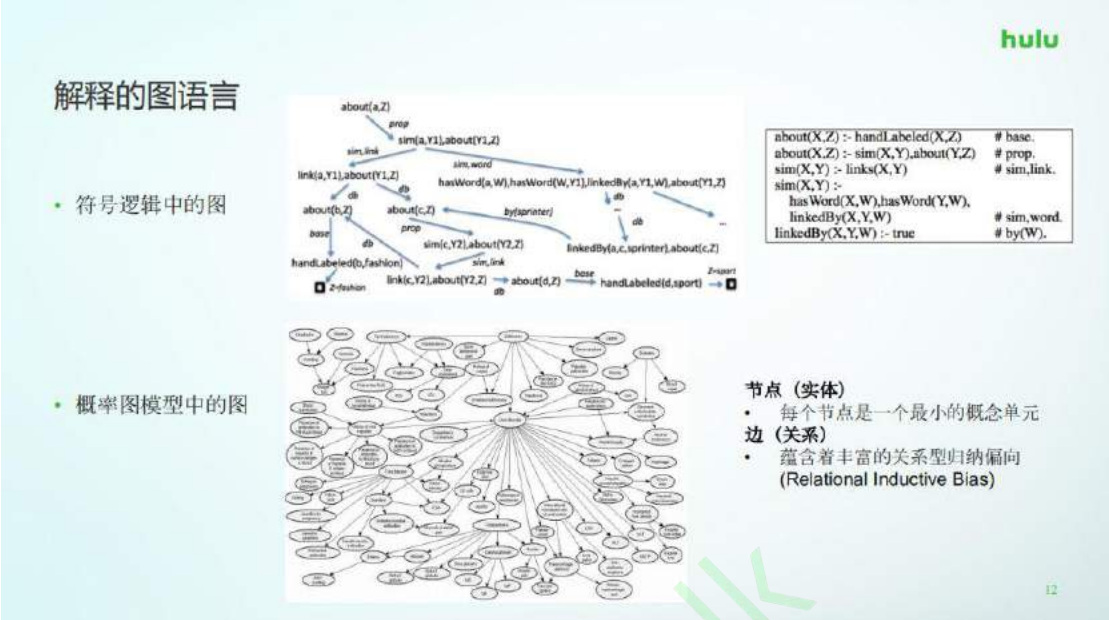
回看我们当下的推荐系统，还确实有很长的路要走。现在的推荐系统，可以把某用户可能喜欢看的《西部世界》排在置顶位置，却不能告诉他必看的 N 条理由，当用户需要更多解释时，推荐系统就变成了一个哑巴，就像一个低着头不开口的推销员，只在不停地往用户手里塞传单。但是，作为《西部世界》的忠粉，如果我来推荐这部美剧给我的朋友看， 我心里肯定已不下 N 条理由，而我的朋友也一定会问我为啥推荐。此外，我们往往会受到一些影评的影响，然后决定去看某部电影或电视剧。

分析推荐系统缺陷的根源，发现现有推荐模型本质上是一个黑盒，这也是很多机器学习算法共通的问题。推荐算法的这支黑盒家族，可以追溯到协同过滤算法，或者更反映其数学本质的矩阵分解、因式分解机；随后，深度学习的兴起，刮起了一波 embedding 之风，有 word embedding 如 Skip-gram 等，graph embedding 如 DeepWalk、TransE 等。于是，我们倾向于尽可能地把一切都 embedding 化，推荐系统中用户有用户 embedding，电影有电影 embedding，标签有标签 embedding；再细些，每个属性都有自己的 embedding，80 后 90 后 00 后各有一个 embedding，悬疑动作科幻各有一个 embedding，周一到周末再加节假日也各有一个 embedding；再然后，我们把这些 embedding ,像下作料一样 ,多多益善 ,放在一个大锅里 ,称它为——Deep & Wide。我不是要否定这种做法，从预测的目的看我反而支持这种做法，很多时候确实管用；然而，除了预测，我们需要的还有解释。

解释长什么样？形式上解释也具有两重性，体现在确定性和不确定性这两个方面。怎么理解？首先说确定性，对于一个解释系统，确定性体现在它的离散性和结构性这两点。举个例子，当你称赞他人时，上前问道：“你做得太棒了！你是怎么做到的？”，然而你可能会收到这样的答复“这个……我也不清楚，凭感觉吧”。他或许在撒谎，不想分享给你听，但是也有可能他确实不知道，或者说不出来。一句话就是：内心有料但无法名状。他能感觉到，但是解释不出来。解释清楚一件事是费脑子的，就像把琢磨不定的乐曲旋律，化作一个个具体的音符，把信息浓缩成一个个离散的符号，固定下来，不让它们在变幻不定中消散掉，就像南北方言可以听上去完全不同，但是对应的汉字都是一样的。这点很重要，在一个黑盒系统的函数拟合学习中，我们通常采用随机梯度下降的方法，在一个高维的梯度场中，只感知到当前点及邻近区域的梯度信息，周围稍远一点的都是变幻不定的漆黑存在，前面几步的路未知，过去几步的路消失在漆黑中，我们只是顺着梯度下降、下降，我们关心的只是让损失函数最小，至于我们是怎么来到当前状态的，我们不需解释，也无法解释。任何的解释，都需要是清晰确定的，需要经得起时

间的考验，而变幻连续的形态无法像离散形态那样稳固，一个像液态的水，一个像固态的冰。而且，离散且有限的形态，更容易让人读懂，更容易传播分享，并且在这一过程保持自身的含义稳定不变，如我们使用的各种语言符号。除了离散性，解释系统还具有结构性，即按照一定的结构模式，将有限离散的符号排列组合成一个复杂的系统，从而尽可能表达出无限的含义。

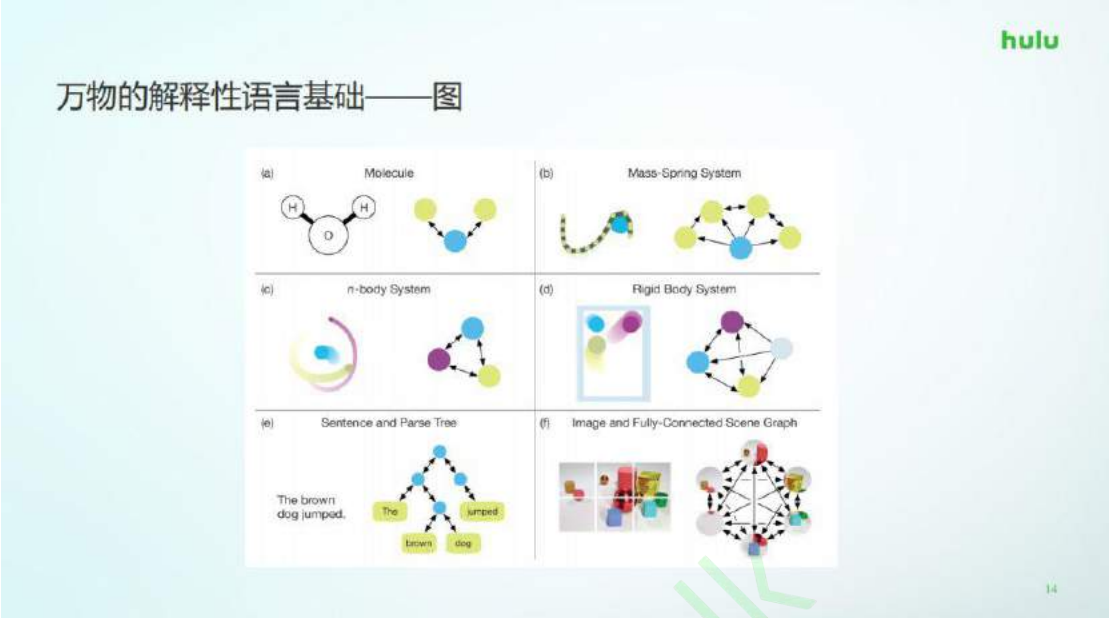
解释的另一重性质是不确定性，这体现在解释系统的概率性和歧义性上。现实世界并非由二值逻辑构成，即使二值逻辑的语言表达起来最清晰最直白，如“他是坏蛋，她是好人”。为了更准确地反映现实，在解释中引入概率或体现概率的字眼是必要的，如“他不一定是坏蛋，她可能是好人”。再说歧义性，我们可以把它理解成精简表达的代价，听上去是一个缺点，但是生活中我们会乐此不疲。一点是，我们都懒得说长句，在当前上下文没变化的情况下，我们会遵循能省则省的原则；另一点是，歧义性也给我们带来了很多乐趣，想一想“小心地滑”（“地”作名词）和“小心地滑”（“地”作助词），不正体现了语言的魅力所在吗，也许这是智慧生物独有的特权吧。

解释的样子最终由解释的语言来确定。对于我们人类来说，主要就是我们平时使用的语言，如：汉语、英语、法语等。但是，对于机器来说，我们需要从计算模型的角度思考“解释”，根据前面所说的性质——离散性、结构性、概率性，不难发现解释的语言正对应到两类经典的 AI 学派：符号学派和统计学派。符号逻辑，这一听上去很远古的领域，在 AI 研究的早期阶段最受推崇，其中研究最广的包括谓词逻辑和一阶逻辑。上图中有一个一阶逻辑的例子。概率图模型，在深度学习火起来前，可能是机器学习界最受推崇的方法论之一，因为引入了概率，所以比符号逻辑的方法论能更好地拟合实际数据并提供解释，通常采用最大似然或最大后验的方法进行优化，具体的方法包括：马尔科夫随机场、话题模型、贝叶斯网等。当然，你可能会想到，何不把二者结合起来，造出一个新方法，名字可以叫“概率逻辑”。事实上，这种方法也已被人想到了，学名叫 Probabilistic Logic Programming。



如果进一步观察符号逻辑和概率图模型这两种形态的语言，可以发现它们在某种程度上都构建了自己的图结构，组织起各自体系中的语义概念（符号的含义）或语义状态（当前上下文）。首先，每个节点代表了一个最小的语义概念或语义状态；其次，一个很重要的概念——关系型归纳偏置（Relational Inductive Biases），通过图中各类型的边刻画出来。

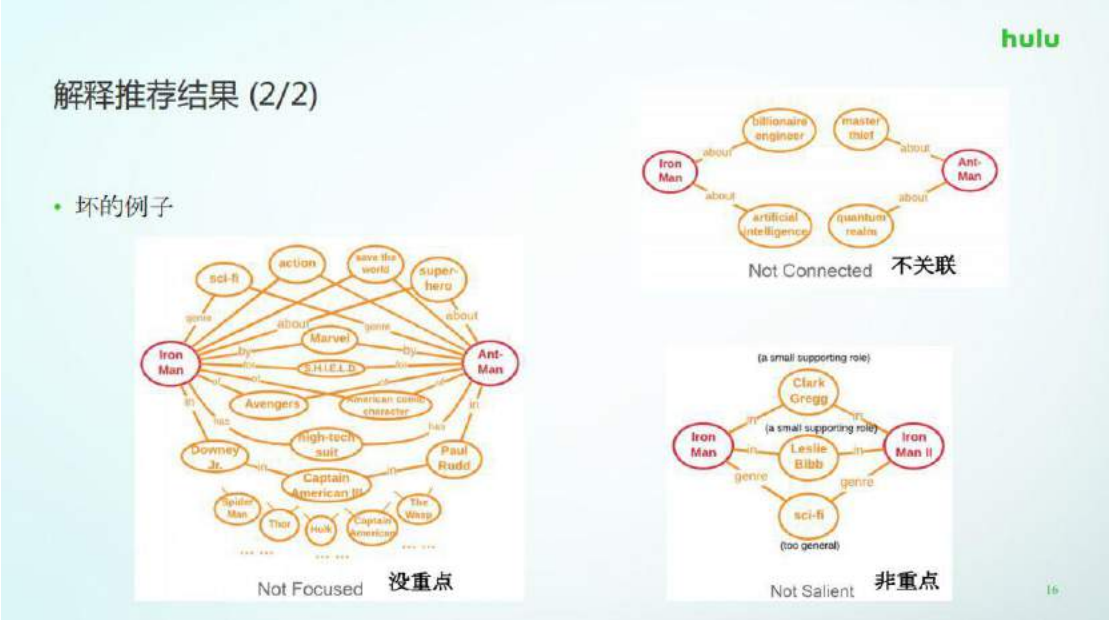
近年来，关系型归纳偏置的概念被人们关注，让我们看看 2018 年 6 月 DeepMind 集众作者合写的一篇论文——Relational Inductive Biases, Deep Learning, and Graph Networks（关系型归纳偏置、深度学习与图网络），文中提到：关系型归纳偏置会是连接深度学习与知识推理、规划的一个关键桥梁，而打通这一联系的实施者就是持有丰富图结构信息的图网络。一方面，图网络是深度神经网络的延伸，继承了神经网络的超强拟合能力来从数据中发现规律；另一方面，图网络因持有的图结构信息，具备基于关系的组合泛化能力，这为塑造解释性的语言提供了前提，也就是说，构成解释语言的第一步要有图。



某种程度上说，图是一种可解释万物的语言基础，从分子结构到弹簧系统，从多体系统到刚体系统，从句法树到场景图片，关系无处不在，多种关系交织在一起形成复杂的图。

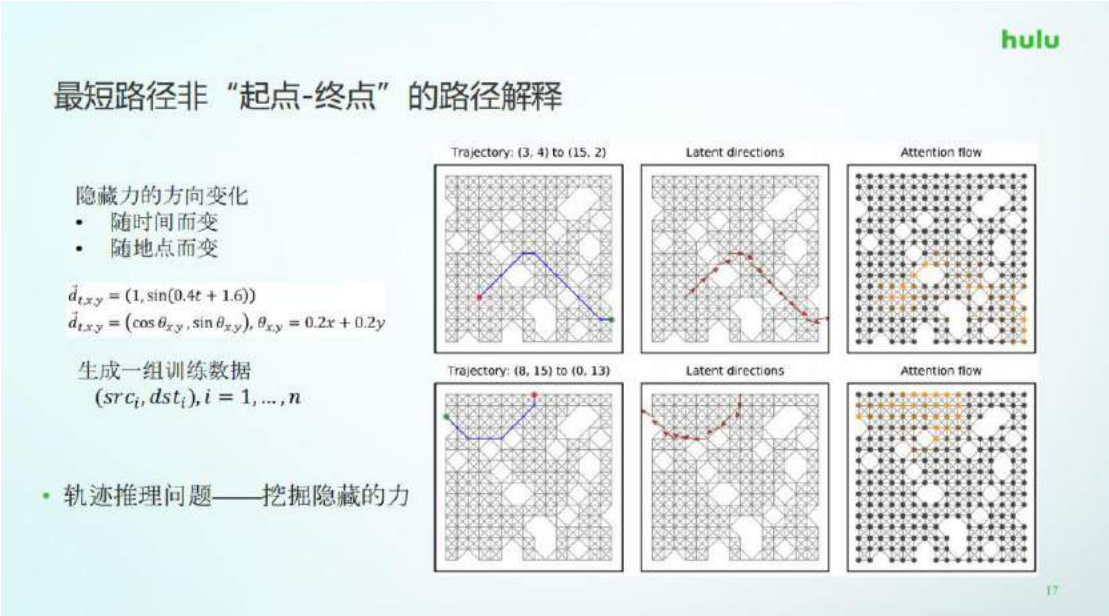
我们回到推荐推理这个任务中，我们要做的是给推荐结果提供解释，比如：已知某个用户喜欢《钢铁侠》，解释推荐系统为什么给他推了《蚁人》。我们把解释推荐结果的过程，抽象成一个在给定知识图谱上寻找从起点到终点的路径，作为我们解释语言的基础。在这个例子中，我们发现之所以给喜欢钢铁侠的用户推荐《蚁人》，是因为这两部电影都属于漫威的复仇者联盟中的超级英雄，也就是说，在这个知识图谱中，我们找到由三条路径构成的子图，一端连接着“钢铁侠”，一端连接着“蚁人”，中间分别经过了“超级英雄”、“漫威”和“复仇者联盟”。

DataFunTalk

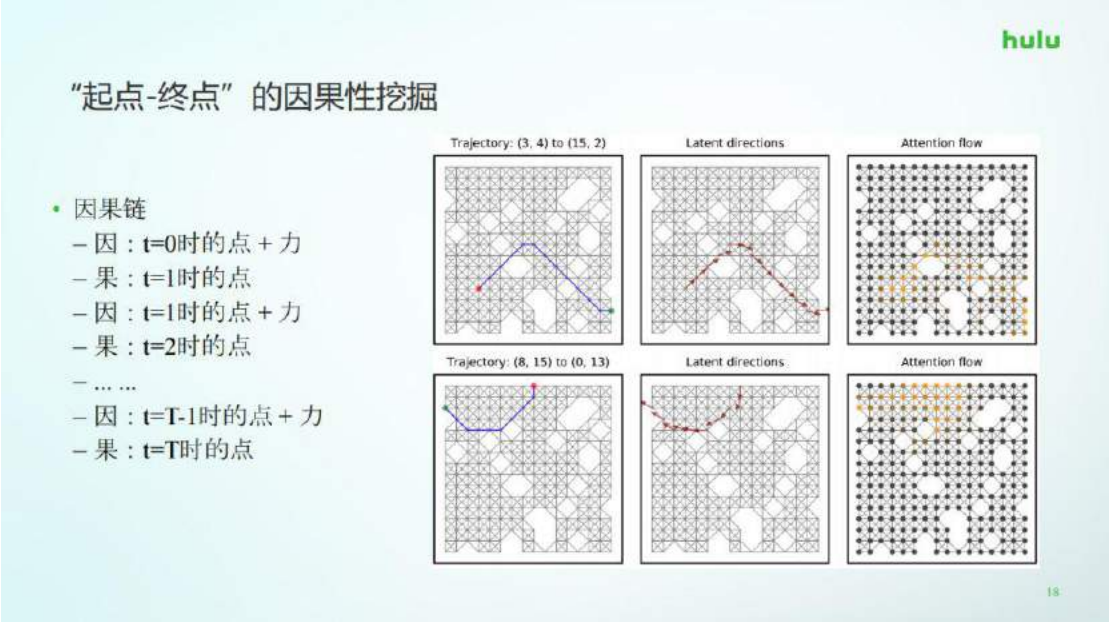


然而，不是随意连接“钢铁侠”到“蚁人”的子图都构成一个好的解释。判断解释好坏的标准有一定的主观性，但是大体上可以归结成三个指标，这里给出三个解释的坏例子。首先，是不关联的例子，这里“钢铁侠”和“蚁人”间没有被连接，“钢铁侠”虽然连着“富豪工程师”和“人工智能”这两个标签，但是这和“蚁人”没有半毛钱的关系，同样“蚁人”连着“盗窃大师”和“量子领域”，这和“钢铁侠”也没有任何关系。其次，是没重点的例子，虽然“钢铁侠”和“蚁人”被连在一起，而且由很多很多条路径相连，但是可读性极差，也没有抓住导致“蚁人”被“钢铁侠”推荐的关键原因。再次，是非重点的例子，这里举的是钢铁侠和钢铁侠 2 的例子，我们发现有很多小角色演员作为中间节点把两部电影连接起来，这当然不是我们想要的结果。

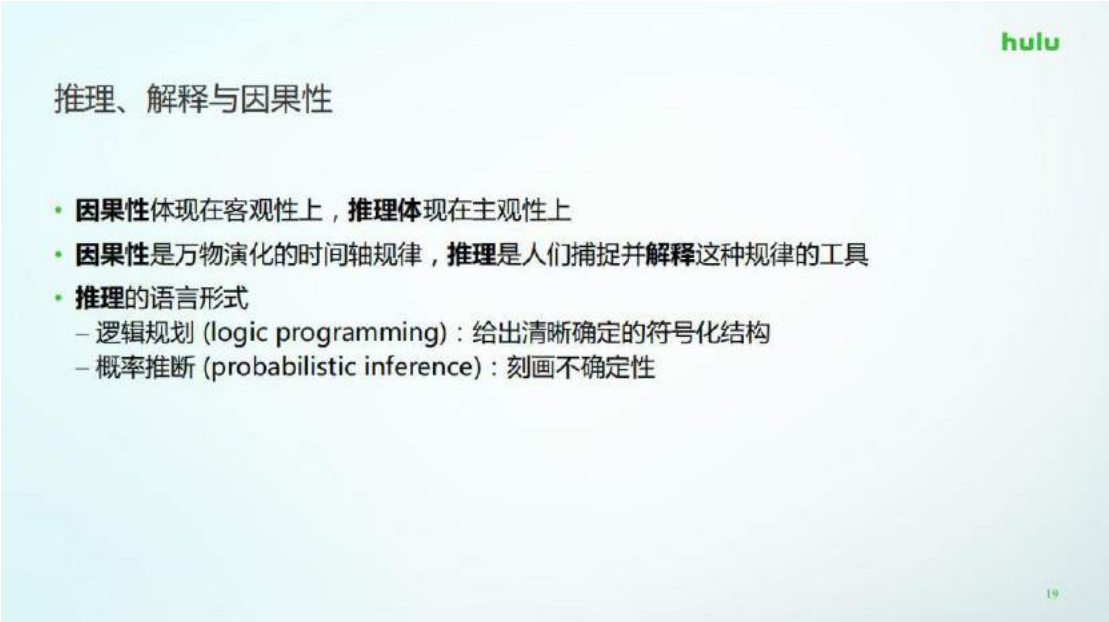
DataFunTalk



寻求“起点-终点”的路径解释，不能被简单地理解成图论中的关于图本身的算法问题，比如寻找最短路径是一个纯的图本身的算法问题，这种问题不是学习问题（Learning Problems），也就是说，给定一个图，算法立刻就能执行起来计算出结果，不需要训练。在我们的问题中，图只是一个环境，或者说只提供了一个场地。想像一下，在一个二维的网格图上，我们为这个二维世界提供一个看不见的力场，在外力的作用下，红色起点位置上的物体被一格一格地移动，经过一段时间后，转移到绿色位置上。这个力场随时间和地点都在变化。现在，假设我们手上的数据只有起点和终点，中间的轨迹信息丢失了，我们需要根据一组“起点-终点”的训练数据推理出物体的运动轨迹，从而探测到力场的变化。因此，图是我们执行任务所处的世界，而这个看不见的力场才是我们执行任务的目标。力场的变化独立于图的具体结构，力的方向可以随时间而不同，也可以随地点而不同。给定起点，导致终点的原因既包括图的结构，也包括力的作用。



进一步地，我们发现物体的运动轨迹实际反映了起点和终点间的因果性关系。在 t=0 时的点，由于力的作用，一个时间步后移动到 t=1 时的点的位置上；同理，t=1 时的点在力的作用下，一个时间步后移动到 t=2 时的点的位置上；以此类推，t=T 时我们得到终点的位置。推理轨迹的过程，实际上是通过模拟物体运动，寻找起点和终点间因果链的过程。



我们总结一下推理、解释与因果性这三者的关系。因果性体现在世界运行的客观性上，推理体现在人类认知的主观性上。因果性是万物演化中内在因素相互作用的时间轴规律，区别于它们在时间轴上的呈现（如相关性）。推理是人们捕捉并解释各因素相互作用内在规律的主观工具。推理不仅要捕捉，即很好地拟合时间轴上的呈现，更要提供解释，超越数据拟合的层面向内在延伸，试图模拟各因素相互作用的演化过程。因此，推理不仅要很好地反映数据呈现的客观统计规律，还要提供主观的世界观构建，作为提供解释的基础。推理的语言形式可以采用传统的逻辑规划或概率推断，也可开发新的基于图结构的方法。

挖掘因果性实际是在回答 why 而非 what 的问题，不是给出起点预测终点是什么的问题，而是给出起点回答为什么是这个终点的问题。这就引出了关于 why 的科学，一门关于解释而不仅仅是预测的科学。这里我推荐 Judea Pearl 的一本新书《The Book of Why》。Judea Pearl 是我们熟知的贝叶斯网之父，2011 年图灵奖得主。

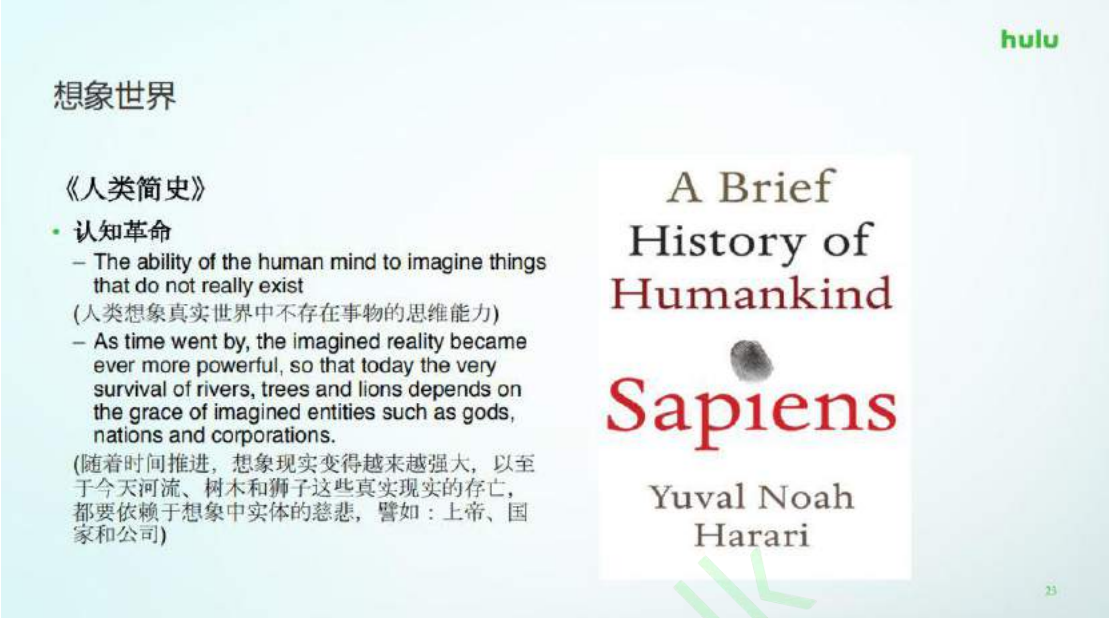
因果性学习和传统机器学习的区别在哪里？传统机器学习，通常认为是“learning from data”，即由数据驱动学习的过程。这种过程是被动的，训练模型的表现严重依赖于手头的数据，现在越来越多的学者认为这种学习只学习到了表明的统计规律（Superficial Statistics），属于感知层面的学习，尚未到逻辑推理、归纳抽象这种认知层面的学习。因果性学习与机器学习最重要的两点区别——干预（Intervention）和反事实（Counterfactual）。

什么是干预？这里举两个例子。第一个例子，为了验证某个猜测，我们设计实验时都会安排一个实验组和一个控制组，实验组就是我们干预的一方，控制组是未加干预的一方，这样做的意义是作对比，实验组中只修改某一个变量，维持其他变量不变，看看结果中实验组一方是否因这个变量的修改而与控制组产生差异，这不就是我们很多人熟悉的A/B test 吗；第二个例子，想象一下婴儿学习，是一个与周围环境不停交互并建立对世界认知的过程，婴儿这动动那试试，干预一下环境，看看会造成什么样的后果，是不是符合原来的预期，还是发现一个惊喜。

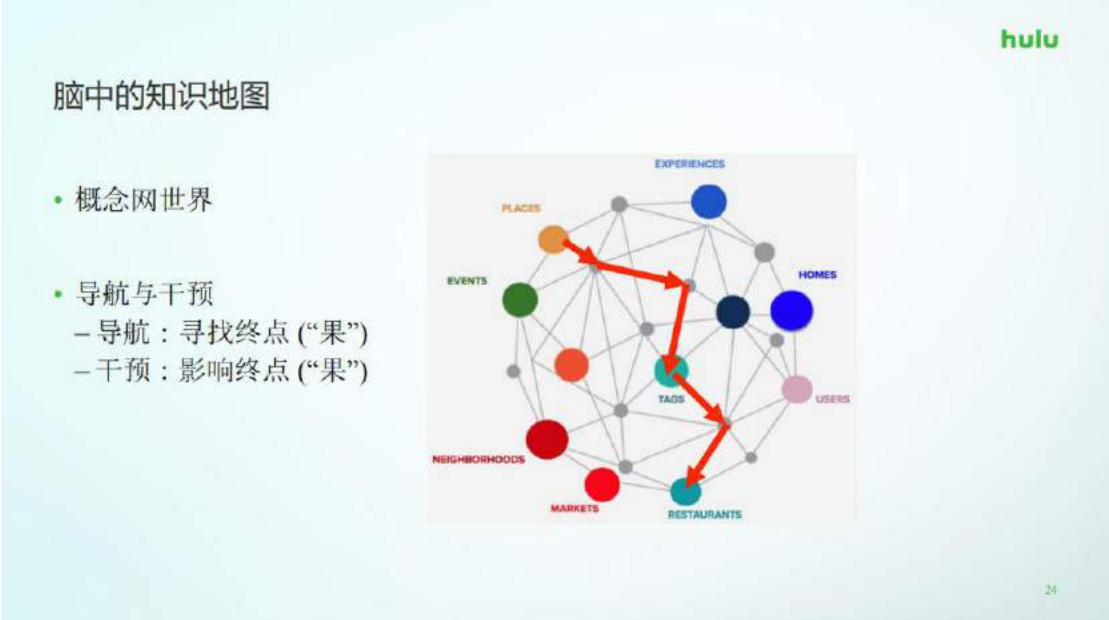
什么是反事实？简单地说，就是构建一个想象世界。这里问你：一个人从十米高的楼顶跳下来会怎样。你脑海里会想象出一个三、四层高的楼，一个人站在楼顶上，如果你是一个细心的人，你会注意到地面的情况，是土地？是沙堆？还是水泥地？但是，你不会为了得到答案亲自去做这个实验，因为代价太大了。为什么你会得出代价太大的结论？因为你已经在想象世界里完成了这次跳跃，得出非死即残的结论。然而，问题换作：一个人从一米高的台子跳下来会怎样。你很可能去亲自做这个实验，其实你已经在想象世界中快速做完这个实验了，得到的评估结果是——没问题最多崴到脚脖子，所以你才敢做这个实验。在想象世界里，我们可以做更多的事情，没有风险，也无需等待很长的时间，时间进度条任你拖动。说白了，在想象世界的这个过程就叫——思考，思考不同于感觉，是受制于逻辑约束的思维推演，可以进行各种假设性干预。

我们把干预和反事实放在一起，举一个例子，问：是公鸡叫醒了太阳吗？回答“是”的理由是，每次听到公鸡打鸣，然后就看到太阳升起，所以是公鸡打鸣造成了太阳升起。直觉上，这显然是一个错误的结论。怎么证明是错误的？我们可以用反事实干预来证伪。

假设有一天，我们把所有公鸡都吃光，当然这不可能发生，所以是反事实的，我们假设会发生，然后问自己：我们还会看到太阳吗？显然，我们知道，太阳会照常升起。



想象世界对人类来讲有多重要。这里，我可以推荐一本书《人类简史》，书中讲到人类文明的诞生源于人类的那场认知革命，认为人类从所有动物中脱颖而出的一次跃迁，就是对想象能力的掌握，可以想象那些真实世界中不存在的事物，那些抽象的或者虚无的概念，人类才有了从未有过的强大的组织动员能力。

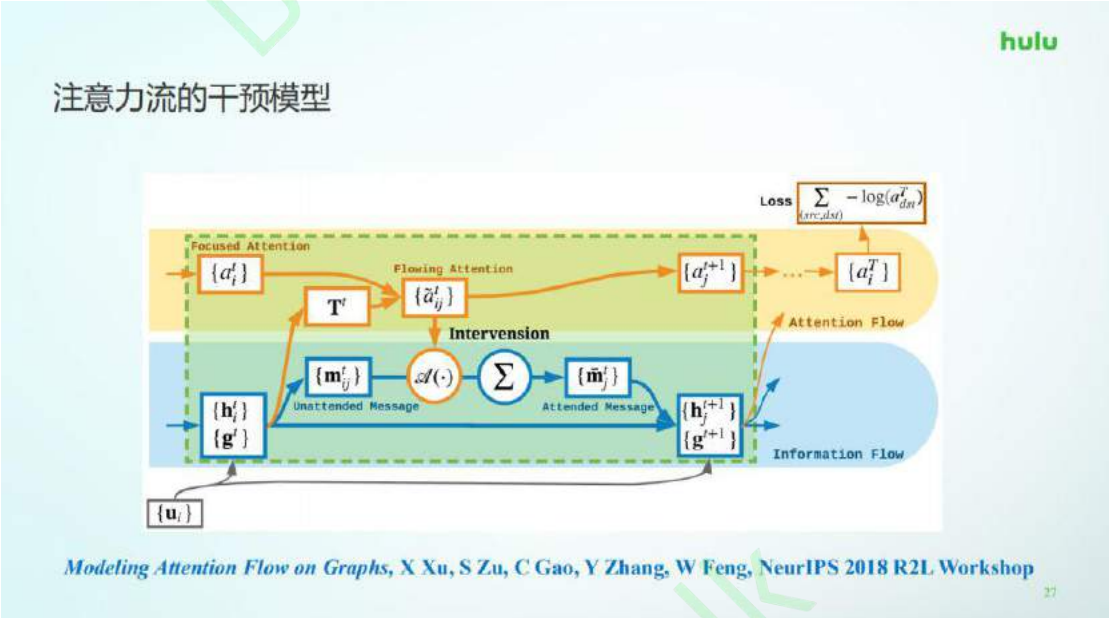


我认为想象世界与真实世界不同，真实世界是物理的、连续且可无限细分的（暂不较真到微观量子如夸克层面），想象世界是概念的、离散的和关系化的。因此，在人脑的想象世界里，应该有一份知识地图，由一个个概念节点连接而成，构成一个概念化的网络世界。注意，这个网络世界不同于大脑的神经网络，不一定存在物理意义上的对应，但是一定存在于某个抽象层面上。想象或者思考的过程，就是不断调用和检索这个知识地图的一些部分，持续在其中导航的过程，从一个节点（如起点）沿着概念网的边寻找另一个节点（如终点）。导航，从某种角度看，就是一个不断施加干预的过程，即不断做出一系列的选择，边选择、边计算、边观察，从而主动地影响终点的位置，获得预期的结果，或者验证某种选择下的结果。由于是发生在概念网这个想象世界里，这一系列选择的过程是虚拟的。

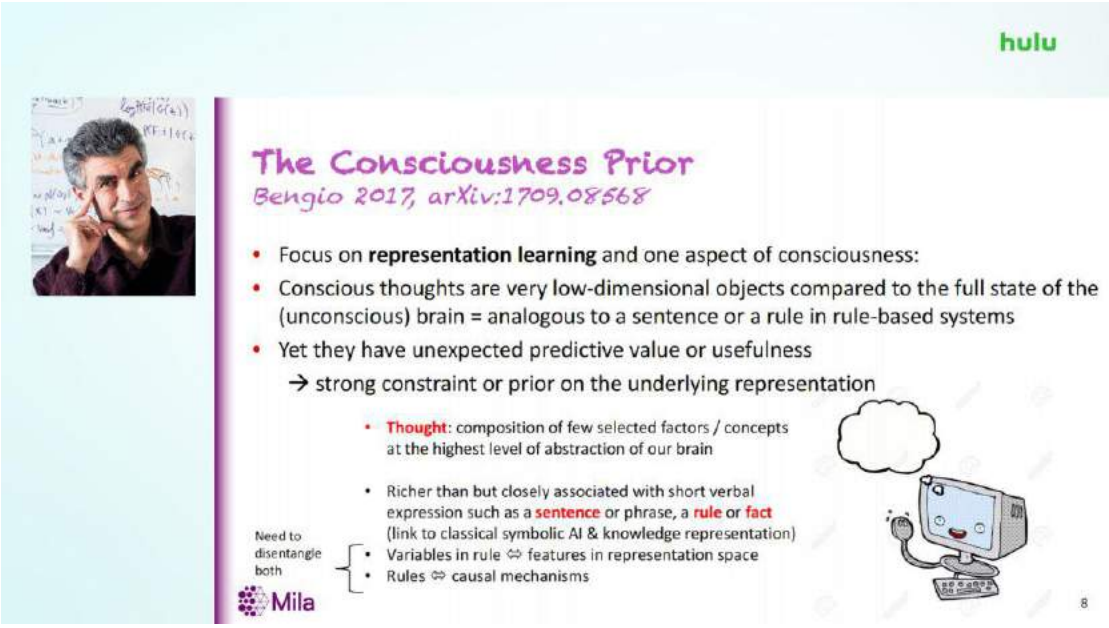
DataFunTalk

在知识地图中做导航，整个过程可以描绘成一连串指引方向的动作构成的序列。因为导航发生在想象世界里，想象中的动作，本质上就是我们的注意力，因为投放了注意力，所以我才有意识地这么想，完成想象的过程。进而，想象中的动作序列，构成了我们的注意力流，这样就引出了意识流的问题。

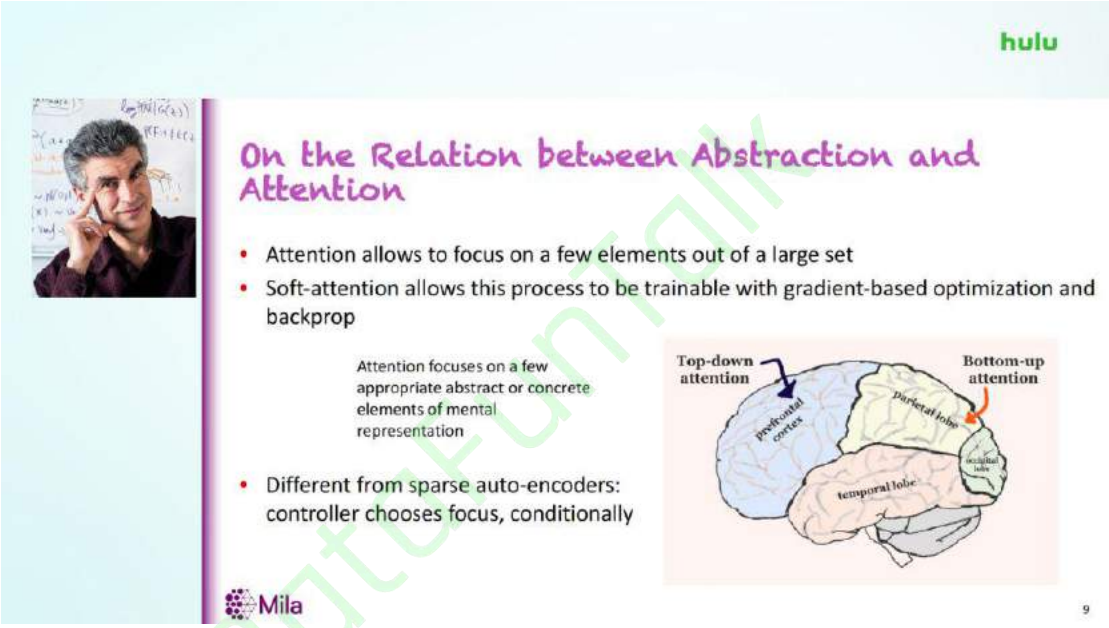
注意力和动作的关系，好比是一对双胞胎，一个发生在想象世界，一个发生在现实世界。在太极哲学中，有句关于太极拳的心法——用意不用力。这里的“意”，是意念的意思，指的就是我们的注意力；这里的力，是发力的意思，指的就是我们的实际动作。



实际的“发力”，实际的动作，会影响环境的变化，进而验证真实世界中的因果关系；而我们的注意力，既然扮演了想象世界中动作的角色，也能影响想象世界中的环境变化，这个环境就应该指我们当下的思考状态。在我最新的一篇论文中，我初步研究并验证了注意力流的干预效应。

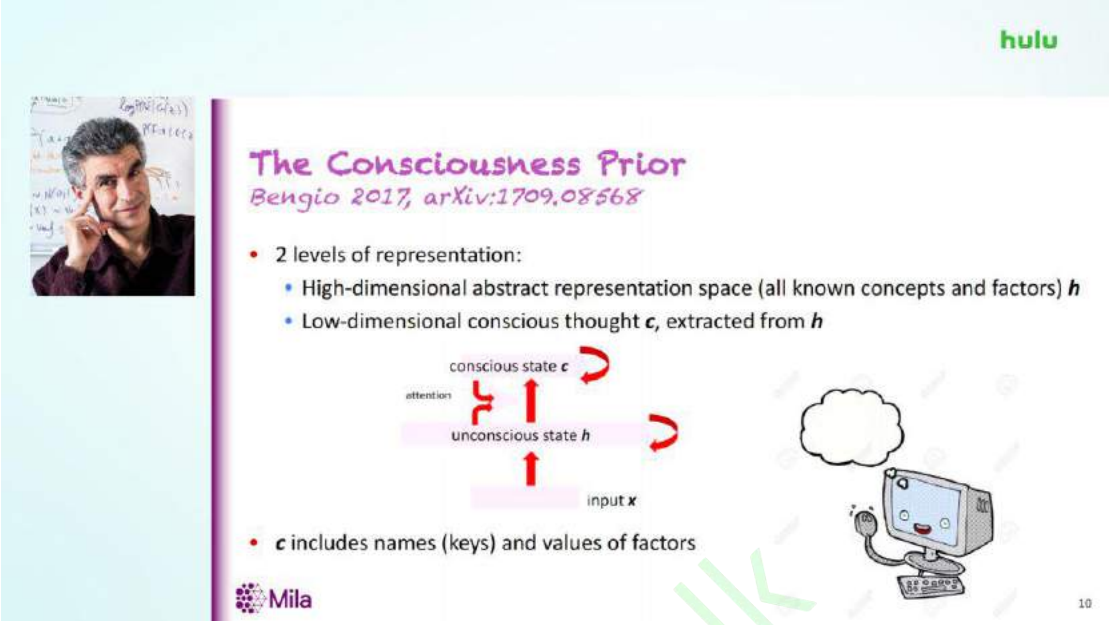


最后，十分推荐大家读读 Bengio 教授的一篇文章《The Consciousness Prior》以及他去年 11 月份来中国演讲的 PPT。他在表征学习与意识上做了长时间的深入思考，认为意识可能对应于大脑世界中那些低维且抽象的物体表示和约束，在表征学习的过程中逐渐抽象解离而获得的更短更精炼、富含语义的符号和概念，以及这些概念间的因果等关系。

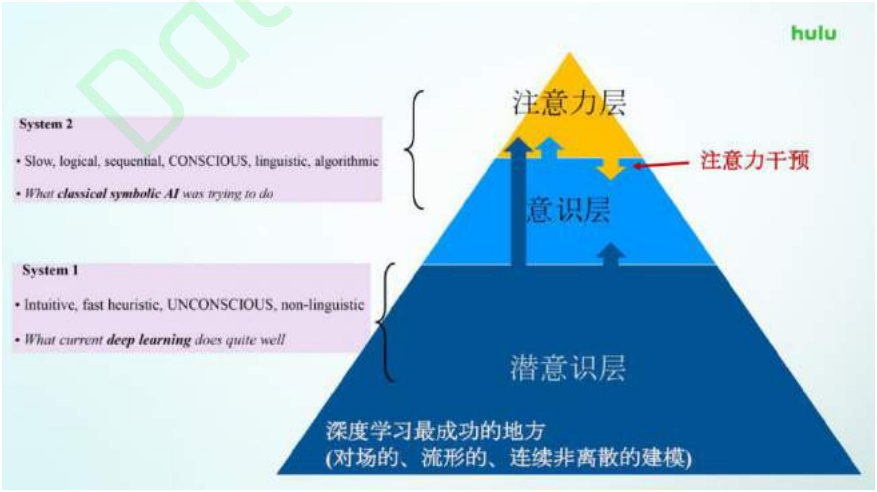


DataFunTalk

这些概念的形成和提炼，本质是根据当前上下文主动投放注意力的结果，从茫茫的表征空间中筛选出最具信息价值的表示并映射到某个符号，形成概念，构成我们的意识。

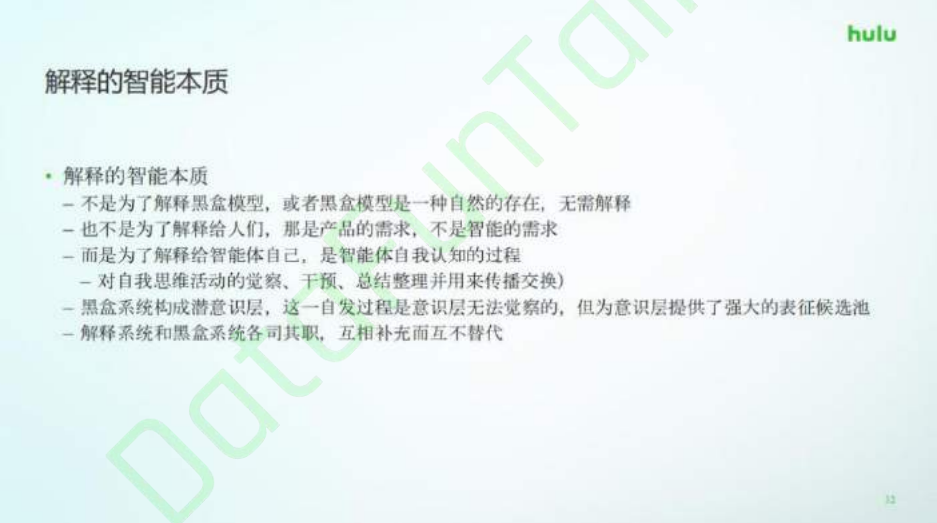


简单地归纳一下，在表征空间中存在着两个层面的表示：一个是处于下面一层的高维表示，蕴藏着丰富的潜在概念或因子，对应非意识的状态；一个是处于上面一层的低维表示，是施加注意力后对非意识状态的提取，形成了有意识的状态。形成注意力的上下文，既有非意识状态，又有意识状态。



我构造了一个金字塔重新作阐释，从下往上依次是潜意识层、意识层和注意力层。潜意识层和意识层都会影响注意力的形成，潜意识层为意识层提供丰富的信息候选，注意力层会干预并影响意识层活动的走向。潜意识层对应了系统 1（System 1），直觉的、快速的、无意识的、非语言符号作载体的，其实正代表了深度学习模型的特点，是对场的、流形的、连续非离散的建模；意识层和注意力层对应了系统 2（System 2），逻辑的、较慢的、有意识的、有语言符号作载体的。

最后，是我关于解释的智能本质的一点总结。



嘉宾知乎地址：https://www.zhihu.com/people/deepknower/activities

达观数据个性化推荐系统实践

嘉宾：于敬　整理：王展昭

本次分享将从以下四个方面介绍个性化推荐系统的构建：

- 智能推荐系统概述；
- 常见推荐算法和应用场景；
- 开发推荐系统的难点分析；
- 达观推荐架构实践和效果优化。

▪ 智能推荐系统概述

推荐系统已经成为市面上数据型产品的标准配置，如淘宝、天猫为不同用户推荐的不同首页的商品，网易云为不同用户推荐不同歌曲，今日头条依据每个用户不同兴趣推荐的

相应视频、资讯等。手机应用中的“推荐”按钮，这种模式都是个性化推荐模式，除此之外在文章、视频详细页都有一个相关推荐，或者百分之多少的用户买了这种商品还买了其他商品等都是推荐系统中的一个场景，叫关联推荐。第三种就是热点推荐，比如今日头条的“热点”频道等，这就是推荐系统常见的三种形式。

推荐系统对于用户的价值而言，它与搜索有点差异，是一种沉浸式的融入体验模式设计，对于用户不明确自己需求时提供的一种个性化推荐。推荐系统可以依据用户的偏好进行不断的信息挖掘，捕捉到用户的兴趣爱好后做个性化推荐，在下拉刷新后依据你的行为实时反馈，如点击、实时收藏等行为，个性化推荐你感兴趣的东西。

个性化推荐对于企业而言，也带来了很大的价值。如电商巨头亚马逊据说有超过 30% 的收益来自个性化推荐引擎，而对于全球最大的视频网站 YouTube，依靠推荐算法，每年视频点击增幅达 50%，而目前很多推荐算法都是从 YouTube 公开论文或算法加以应用的。LinkedIn 也提供机器学习，为公司带来数十倍的稳定增长。

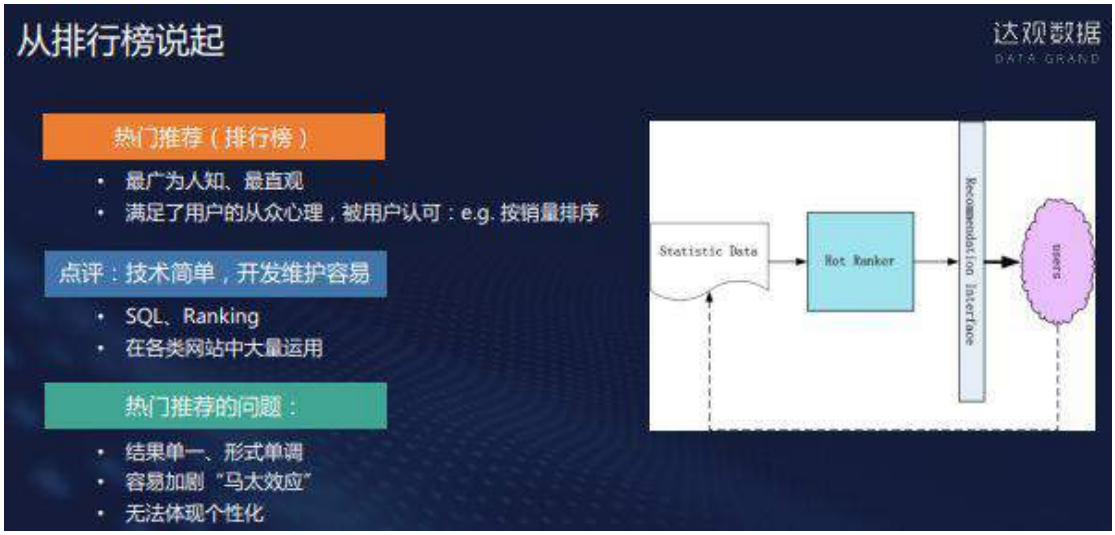
个性化推荐就是一种个性化的感知，架起用户与内容间的桥梁，能让用户找到自己感兴趣的东西，同时能让物品能有更多的曝光机会。对于企业和 APP 来说能够增加流量、增加用户粘性。推荐系统产生主要是解决量大问题：

- 信息过载，就是在一个场景中信息超过你的选择时，很难找到自己感兴趣的东西；
- 长尾问题，很多产品，如视频文章很多都是缺少曝光机会。

推荐系统设计目标主要有三点：

- 功能：功能上要全面些，包括相关推荐、个性化推荐、热门推荐等还包括混合推荐，如一个详情页会出现无限下拉的过程，其实在这种相关推荐做了个性化推荐；
- 效果：效果在不同领域有差异，如在直播领域关注送礼物、打赏收入等，而资讯行业较关注人均点击数量、用户停留时长等；
- 性能：性能在不同领域也是有差异的，但是必须是快速、稳定的，不允许出现推荐位置的留白，也就是你的推荐系统可以效果不好但是不能空白，在高并发时要求性能稳定快速。其实在实际业务场景中这三者是相互影响，权衡利弊的。

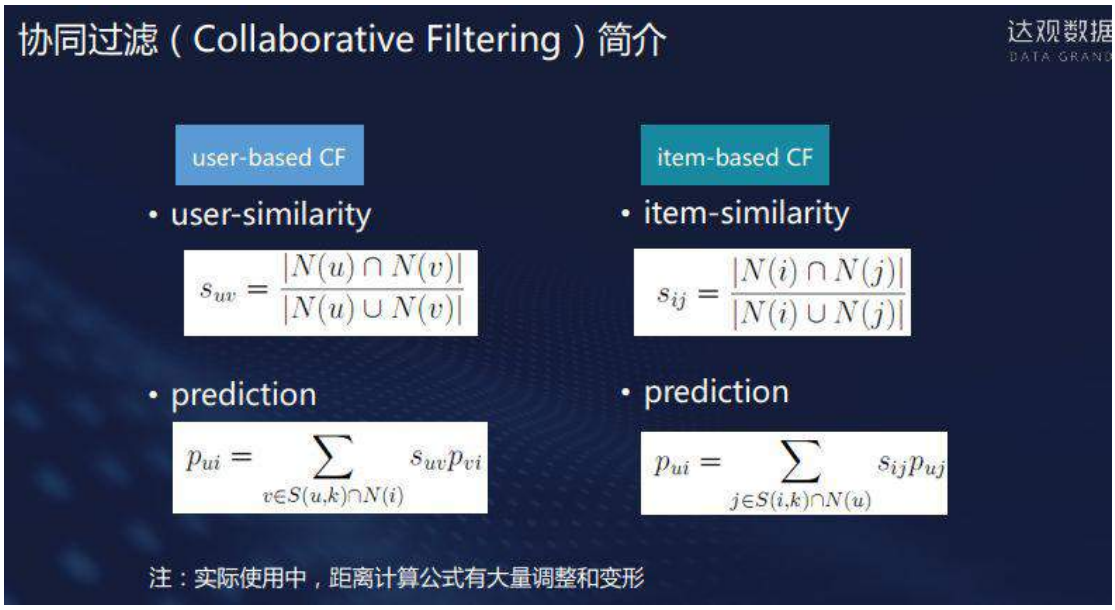
▪ 常见的推荐算法和应用场景



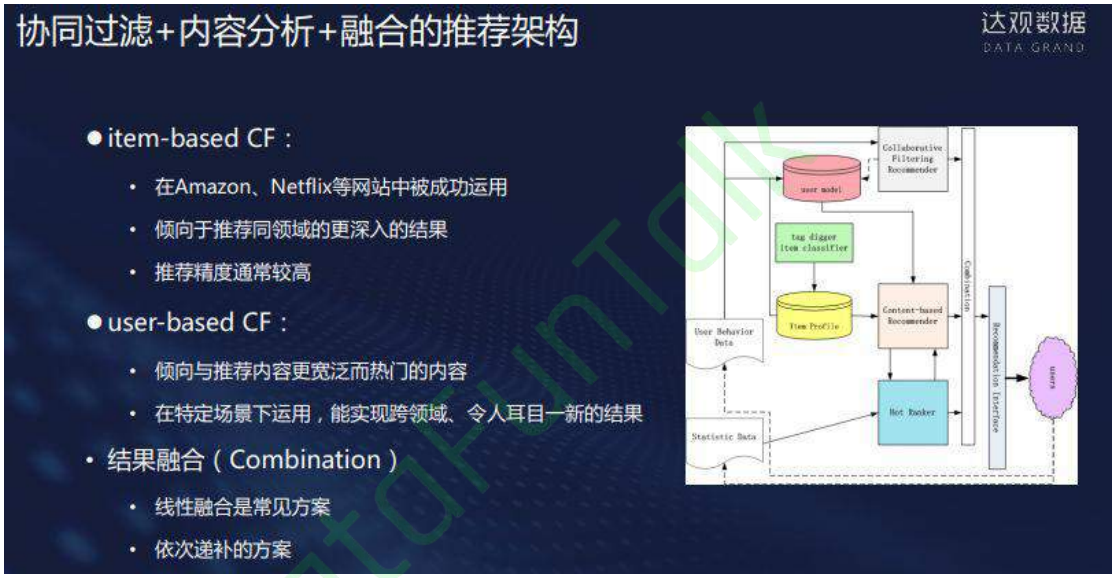
首先，从排行榜说起，这里其实还有简单的人工运营，就是人工每天筛选一部分优质信息进行展示，这种虽然千篇一律，但是排行榜很容易广为人知，主要是满足了很多用户的从众心理，这种方式技术比较简单，运维也比较方便，只需通过简单的 SQL 语句就

能进行 Ranking。这种方式存在的问题就是结果比较单一，缺乏个性化，但是这种方式确实带来了不少的收益，不少用户确实希望看到高质量和热门的东西。

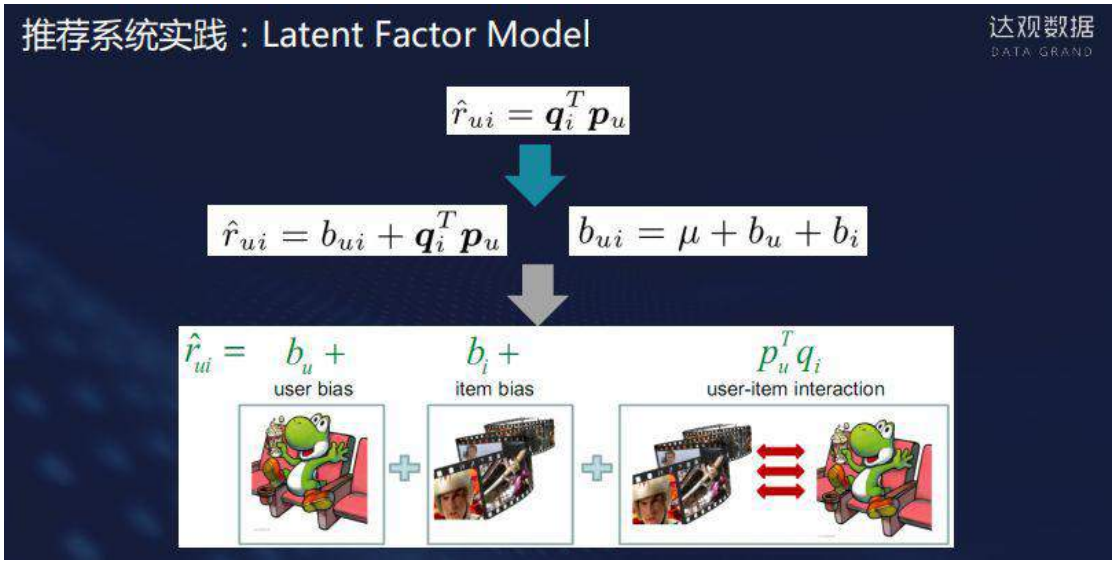
再进一步优化时可以进行 item 内容分析，这里面包含很多信息，如文章有标题、分类、标签、作者、发布者等很多有用的信息，因此可以依据这些物品信息做一些基于内容的推荐。然而仅仅基于文本内容的推荐效果也会打折扣，因此还有其他一些信息的提取需要依靠自然语言处理技术来进行处理。与 NLP 部门结合，除了自身待推荐物品的信息之外，还会做文本聚类分析，包括本体、内容打上不同的主题，做内容方面的召回。当看到一篇文章或视频后，基于内容的推荐可以提高相关性，这种推荐思想比较直观。接下来更进一步基于用户模型的推荐，主要是做用户画像，如果需要做更精准的个性化推荐首先需要对用户做一个全面的用户画像，对用户打各种各样的标签，对用户模型做更加精细的刻画。

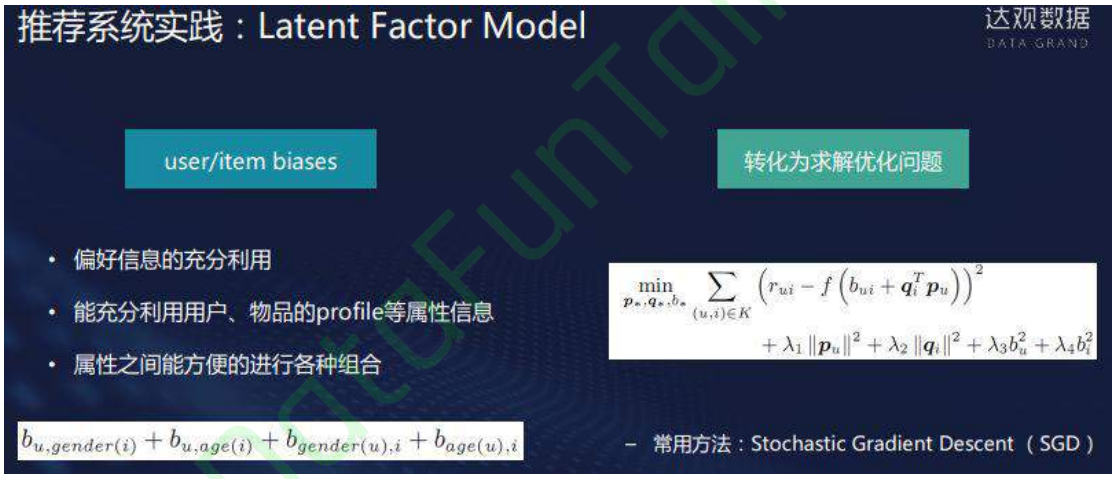


协同过滤是推荐算法中必不可少的算法。协同过滤一种是基于用户的，一种是基于物品的。基于用户的协同过滤主要是基于群体智慧，如想看电影的时候，会去咨询下朋友或者去豆瓣看看电影评价。其过程是首先找到与你兴趣相似的用户，基于相似用户喜欢什么再给你做推荐。基于物品的协同过滤就是基于用户行为过滤做协同过滤，如 1000 个人看了 A 视频或商品，同时 1000 个人又看了 B，很明显 A 和 B 是有关联性的，将样本扩大就能找到相似物品，基于这种关联规则的挖掘就可以得到相似物品进行召回就可以做预测，基于距离的相似度计算在实际应用中有很多变形，比如基于点击行为、购买行为等，也要考虑物品的冷热不均等。



在我们实验中，一般情况下引入协同过滤比单纯基于内容推荐的推荐效果会高出一倍以上。其算法本质上是基于群体智慧，用户不断地下拉及相关行为其实是对算法的结果进行选择，机器算法能够学习到这种选择。这种方式能够挖掘潜在的相关性，如购买了手机后还会买一个手机配件，通过用户购买记录可以推荐给用户更好地选择。实现过程就是首先要生成用户的行为矩阵，然后生成用户与 item 关系矩阵，然后计算用户与用户的相似度、item 与 item 相似度，对个性化推荐结果进行预测。





除了基于相似度的协同过滤，还有一种效果很好的隐语义模型。这是完全基于算法来实现的，每个用户都有自己的评分，形成用户评分矩阵，通过矩阵分解，将最初用户行为矩阵的空白处填补起来，并将预测的评分从高到低排序取 topN，生成推荐结果。这种算法虽然推荐效果比较好，但实际应用并不是很多，主要问题是可解释性比较差。该算法在在实际效果调优中除了单纯的评分，还有很多其它因素考虑的，比如当用户评分的时候，会存在用户和 items 本身 bias 的影响，比如用户有的倾向于打高分有的倾向于打低分，有的物品倾向于得高分，有的物品倾向于得低分，需要将两者结合进行效果调优。

在此基础上还可以做进一步的优化，如考虑到年龄、性别等特征，做特征的交叉组合，进一步的提升效果，而对于模型求解的过程就转化成一个最优化问题。

- **开发推荐系统的难点分析**

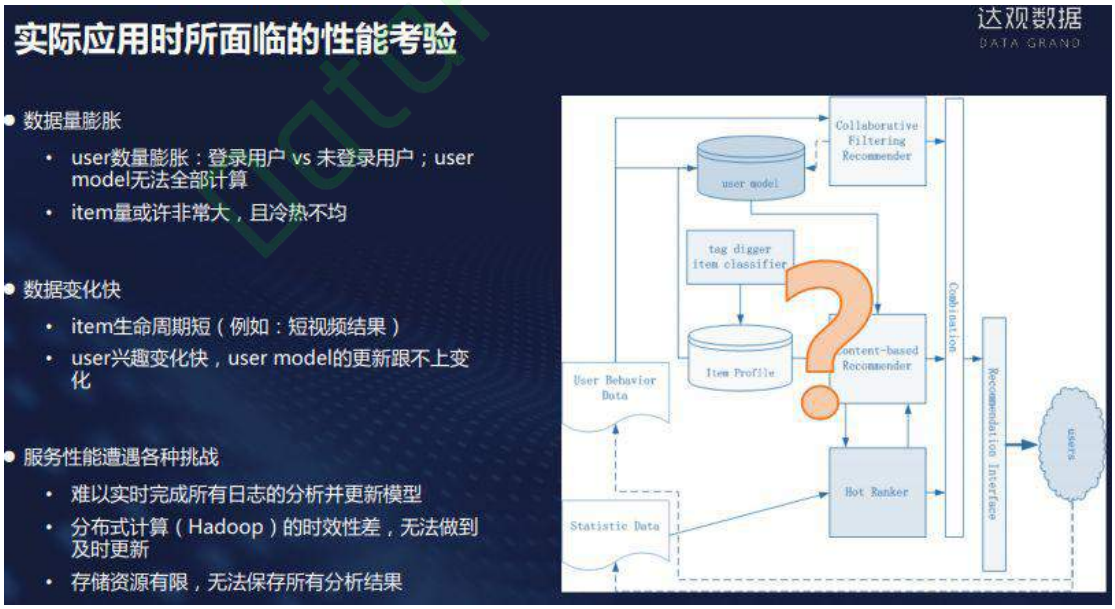
达观数据为众多不同的行业提供推荐服务，包括媒体、视频、直播、电商等，客户行业不同，数据量巨大，同时需要兼顾性能和效果，在此也遇到一些挑战，接下来跟大家分享下：

首先的难点就是用户信息量比较大，实际中用户场景变化也是比较大，如在做母婴电商时，开始比较关注于奶粉这一块，接着可能会买一个小推车。用户的信息是在不断变化的，尤其是在用户量信息比较大的情况下，还有用户的兴趣也是随着时间不断变化的(如欧冠开始的时候关注欧冠的新闻，西甲开始的时候关注西甲），这就对用户画像提出了更高的要求。再就是用户画像的多样性，如内容数据、好友关联数据、更丰富多样的行为数据，这就对构建用户兴趣化模型提出更高的要求。还有就是基于各种算法的召回，不同模型对用户兴趣开发的不同粒度如何去生成结果，对几十万的用户画像推荐的结果如何进行个性化的处理。

DataFunTalk

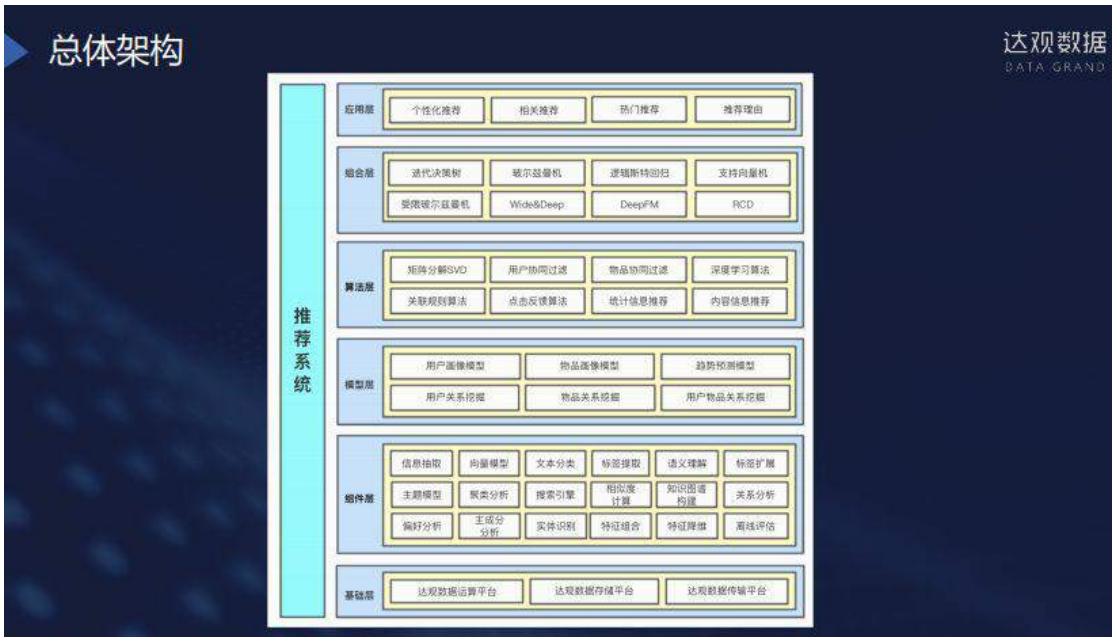
其次就是冷启动的问题，这是推荐系统中比较难的问题。主要就是一个新用户到来之后如何生成推荐结果，尤其在当下对于很多公司引入新用户的成本是非常高的，如何能让用户快速的停留下来并进行转化是需要重点关注的。还有就是对于新物品来说缺乏曝光机会，质量是参差不齐的，如在发布的众多视频中如何让好的视频能够快速呈现出来，再自动的逐渐过滤掉和筛选，这对算法来说有很大的挑战。因此冷启动是推荐系统中比较核心的问题。

然后就是推荐结果的单调性和重复性问题，比如“回声效应”，我喜欢体育然后推荐的全部是体育方面的内容。在 APP 首页呈现的是推荐系统推荐的东西，有可能不是你喜欢的东西，如果推荐系统不知道你喜欢什么，那么你永远看不到你喜欢的东西。



最后就是性能方面，首先数据量非常大，面对不同行业的不同客户，用户差别大，又有海量数据需要分析挖掘，并实时精准得进行个性化推荐，性能压力可想而知。第二个就是数据量变化比较快，如短视频每天产生的数据有几十万。还有一个就是服务器性能，需要几十毫秒返回推荐结果，对于算法模型也有更高要求。另外就是对于我们服务的客户，首先会有一个 A/B 测试环节，进行很多 PK 达到客户的要求，然后才会买单，这比单纯的搭建一个推荐系统的要求要高很多。

- **达观推荐架构实践和效果优化**



上面是系统层次结构图。

基础层，对于服务多家客户来说首先是基础运算平台，全部基于 Hadoop 和 Spark。基础存储平台是基于 HBase、MySQL、Redis、HDFS 等，传输平台是 DgIO，主要基于消息队列的方式。

在组件层有各种各样的组件和算法库，实现多个产品服务都可以复用。对于这些组件也有相应的研发团队进行升级和维护，如文本分类、标签、语义理解都是由文本组处理，对于搜索引擎性能、相关性等的优化升级是由搜索组完成，组件都是共同使用共同维护。

组件层有一系列小的组件，基于组件可以做一些模型层的事情，比如推荐相关的做用户画像，因为对于不同行业的用户画像有不同的标准，我们拿到的就是用户 id 和行为数据，刻画用户画像主要基于向量方式。物品画像主要解决流向，就是物品来了如何及时曝光，这时就需要依据其初始信息进行预估打分，对于已经曝光的物品会记录一段时间

DataFunTalk

的收益情况（点击率、收藏数据等）形成物品画像做一些过滤信息。趋势分析主要是物品曝光后接下来是怎么样的，用户关系主要是基于用户行为分析的，主要做社交关系的推荐。物品关系主要是做算法方面的处理。

算法层主要是包括基于内容的推荐、矩阵分解、协同过滤、深度学习等。基于内容推荐如标签召回、热门召回、内容召回，深度学习各行业都在使用。

接下来就是一个组合层，对各种单一推荐算法的召回结果，使用机器学习的方式进行融合，以达到推荐效果的最优化。

最上面是应用层，目前提供三种推荐，同时还有推荐理由，就是可解释性。

目前个性化推荐引擎提供以下几种服务：

- 数据管理模块：包括数据采集、预处理；

DataFunTalk

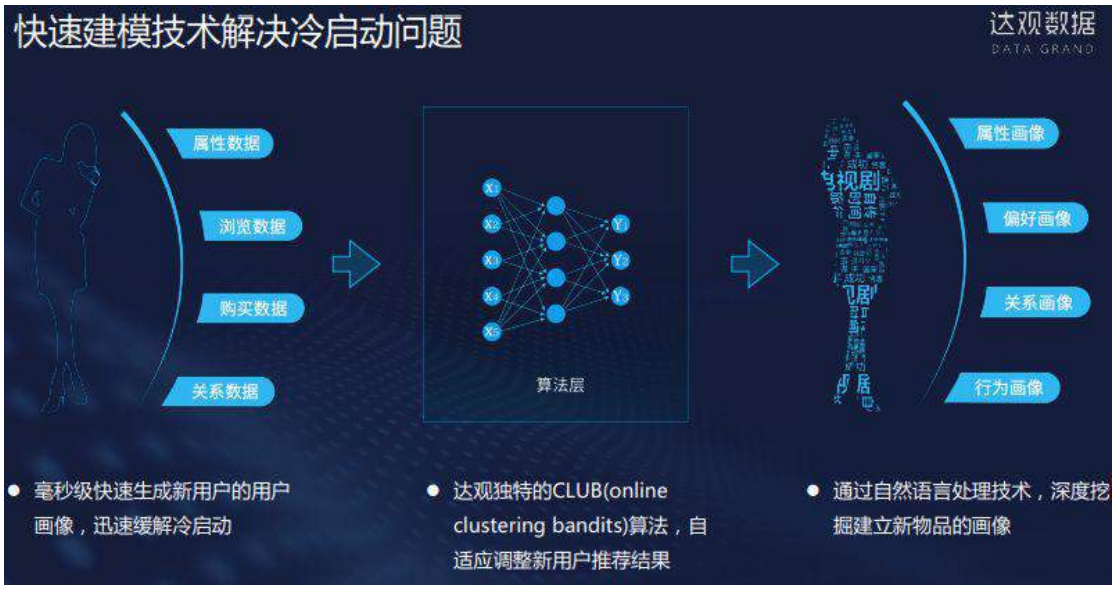
- 语义分析：推荐系统很多与语义理解是分不开的，主要依赖于 NLP 基础组件的服务，如标签的提取、分类，还包括情感分析；
- 推荐算法：包括基于内容的、标签的、深度学习的，还有 CLUB 冷启动推荐算法；
- 用户画像：会有群体画像、个体画像，
- 服务化接口和可视化配置平台：对于大型客户而言他们有自己的产品人员、运营人员，使推荐平台不是一个黑盒子方式，提供一个配置让产品、运营参与进来干预推荐结果。

任何一个推荐系统数据是第一位的，需要拿到足够丰富的数据。包括前期数据采集，再做效果调优时需要各种各样的行为数据，推荐一个东西展示出来如何实现一步步的转化需要数据采集做到非常精细准确。第二块就是语义分析，除了直播外如视频、文章等都有各种各样的文本信息、标签、内容，这都可以做很多语义方面的处理。文本处理方面

主要是做 NLP 处理，如拿到一个文本可以做标签的召回、提取，直播领域中的弹幕、评论与主播的内容以及客户流失是有很大关系的，标签和标题都是动态变化的，分词会产生很多无用的分词，如果做简单的召回是不符合实际的，效果也很差。

用户画像主要是用于推荐服务的，主要是基于行为数据、属性等进行深入的分析挖掘，得到用户不同维度的多种信息，比如使用机器学习模型进行预测（高价值用户、价格敏感型等），同时画像信息可以做沉睡用户分析、流失用户分析等，同时企业也可以做流失预警、沉睡唤醒等服务。

DataFunTalk

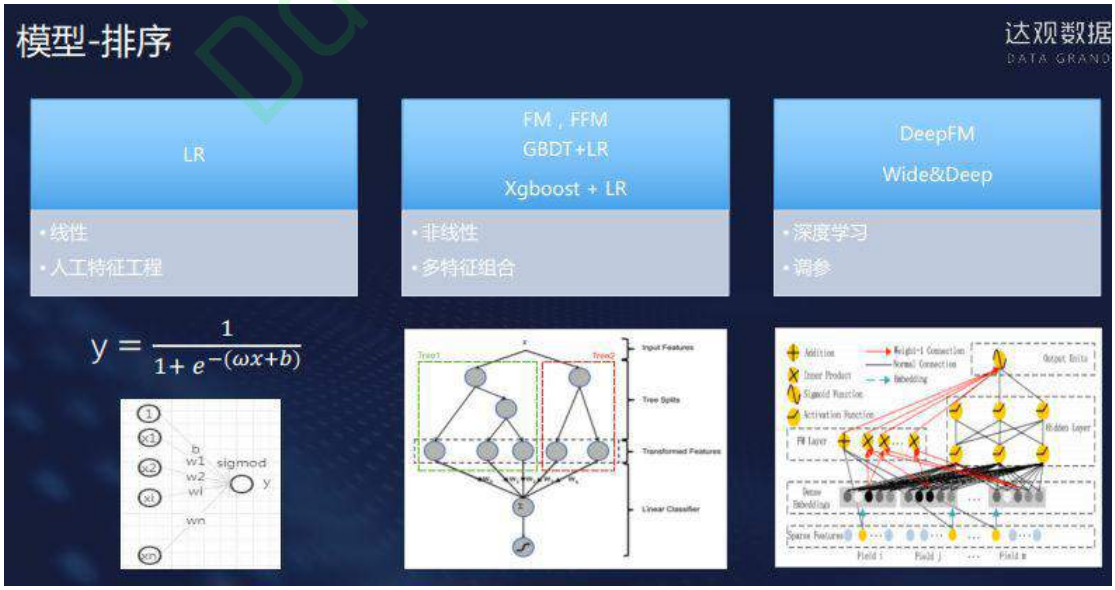


冷启动问题，我们的解决方案首先是基于物品画像，主要基于物品属性来做预测，通过机器学习来进行初始打分，接着就是 CLUB 构建探索再利用机制，将用户划分为不同群体，当新物品来到实时提供曝光机会，会有一个收益指标来评估变化，收益预期是否达到要求，通过自动调优的方式来决定是否增大或者减少曝光。还有会通过用户来选择一些感兴趣的标签来解决冷启动，以及基于本身用户属性信息（性别、地域、年龄）来作为冷启动的依据，通过反馈不断学习提高精准度。

在优化过程中相似度计算的时候，置信度是需要考虑的，优化方法是对同类型用户计算相似度在进行聚类，还可以进行用户分类，如专家用户，专家喜好会影响大家的喜好。还有 Item-based CF 是对物品与物品之间先做一次聚类，先区分相似物品再对相似物品使用协同过滤，这种效果有很大提升。

Embedding 处理主要是做内容多样性，不同内容可能会带来单一结果，可以用标签扩展的机制，如喜欢 C 罗的用户当 C 罗新闻过少时，可以推荐尤文图斯相关新闻或者欧冠或者梅西相关的新闻。还有就是类别扩展，当用户喜欢体育类的文章，尝试推荐科技类或者军事类的文章，这样就可以解决多样性问题。再一个是 item embedding，我们用 word2vec，将用行为的历史数据做一个队列，形成标签扩展得到相似的 item，这主要通过深度学习实现实现相似 item 的召回。第三种就是特征工程，人工特征组合有很多局限，通过特征组合形成特征向量放入机器学习模型，能够很好地实现模型泛化。

模型处理分为离线、近线、在线，算法方面分为召回、粗排、精排。涉及的算法有重量级的算法（机器学习、协同过滤等），利用大数据集群，响应时间在小时级，数据结果在千级。轻量级算法主要针对于粗排，基于机器学习做 ctr 预估，主要是在内存里面计算，有时会和 mysql 进行交互，运算时间在秒级。接下来就是精排，主要是基于业务规则，运算时间在毫秒级，很多时基于内存或者 redis，得到的结果在十条量级。



在排序模型方面很多基于 LR 模型，现在很多都是基于深度学习来做，不同模型都有不同的应用场景，并不是单一使用一种场景。LR 模型利用人工特征工程，相对于深度学习的优点是可以感知的，是可以 debug 的。LR 模型对于特征处理是线性的，利用 Xgboost+LR 或者 GBDT+LR 由线性向非线性转化，能够做到多特征组合，对推荐效果也有不同程度的提升，目前还有利用 Wide&Deep，可以从特征工程中解放出来，在特征选取方面不需要做很多工作，但是在调参方面工作量比较大。

达观数据推荐系统平台提供的是推荐技术服务，通过数据上报接口的方式拿到客户数据，然后通过推荐接口服务提供给客户推荐结果，完完全全将客户解放出来。

▪ 结束语

最后心得体会就是做推荐优化时要依据用户行为出发，避免过多的客户干预。避免本末倒置；搭建一个推荐系统不难，难在如何持之以恒的提升效果；没有坏的方法，只有坏的用法，需要依据合适的场景应用模型。

DataFunTalk

Embedding 技术在民宿推荐中的应用

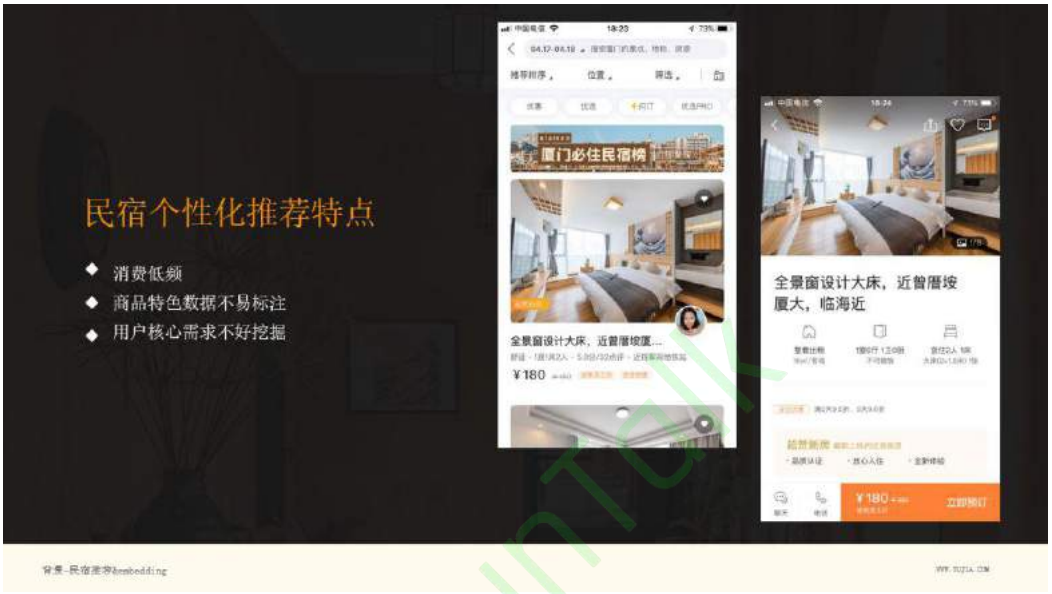
嘉宾：周文彪　编辑：HohXil

民宿越来越成为旅游消费的新热潮，作为国内最大的民宿平台，途家在应用 AI 技术进行个性化商品推荐方面进行了不少探索。民宿行业的数据，具有用户消费频次低，用户兴趣点不好描述等特点，基于内容和普通协同过滤的方法效果都不明显，受 graphic embedding ，word2vec 的启发，我们运用 Embedding 技术有效建立了商品之间的相似关系，运用在物物相似推荐场景通过 AB 测试效果明显胜出。这里对这方面的主要工作进行一个介绍和分享：

- 背景：民宿推荐 & Embedding
- 方案：业务中 Embedding 如何落地
- 改进：落地时的摸索和技巧
- 后续：未来计划

一、背景

民宿在国内还算是比较新的行业，发展比较快，在国外民宿已经发展很多年了，沙发客文化在国外比较流行。



图中右侧部分，为途家 APP 的界面，左边为 list ，右边为点击第一个房间的详情，如果没有用过民宿的同学可以感受下。民宿列表页和携程、去哪儿酒店有明显区别，他们的图是比较小的，而民宿看起来图片很大，什么原因呢？其实民宿是非标住宿，个性化较强，住酒店的话，可能你想好几星，位置，价位，就可以选择了，但是民宿，人们很多时候，会想住一个山景房，或者一个木屋，住在海边，或者是繁华地段，需求方面会注入更多个性化因素。

总结下，民宿个性化推荐特点：

1. 消费低频，它是旅游需求，相对低频，且用户行为中位置选择占主导，这导致如果应用传统的协同过滤方法，不容易出效果，即使得到的结果也仅仅体现出位置相近的关系。

2. 商品特色数据不易标注，民宿的商品比较特殊，除了房子基本信息，还有很多潜在信息，不容易表达和标注。

3. 用户核心需求不好挖掘，当给出能住几个人，在什么位置，是别墅还是农家乐等信息，用户往往还是无法做出选择，还是会详细的看看房子的图片，或者和商户再进行一些沟通，询问些问题，比如是否适合小孩入住，是否有宠物，因此用户的核心需求也不好挖掘。

这就是民宿推荐上的一些特点。

二、方案

为了在民宿推荐上引入更多的个性化，我们考察了三类个性化推荐的方案：

1. 基于内容的推荐

依赖的数据：这个方法很直接，如果我们可以把房屋的各个维度的数据描述出来，都打上标签，我们就可以进行推荐，比如装修风格，图片怎么样，是否有些有趣的特色。

特点：可覆盖新品，需要大量前序数据挖掘和标注工作，潜在特性不易表达，数据描述的全面性和准确性难保证。

2. 基于协同过滤的 item-to-item 相似推荐

在这里我们不用 user-to-user ，因为它是低频消费的场景，用户之间的相似度很难挖掘，所以我们这里重点考察 item-to-item 相似推荐。

依赖的数据：用户行为日志

特点：不需要数据标注，从高层角度进行推荐；无法覆盖新品，在酒店民宿上应用会陷入以地标相似为主的囧境，弱化了其他个性化特征。

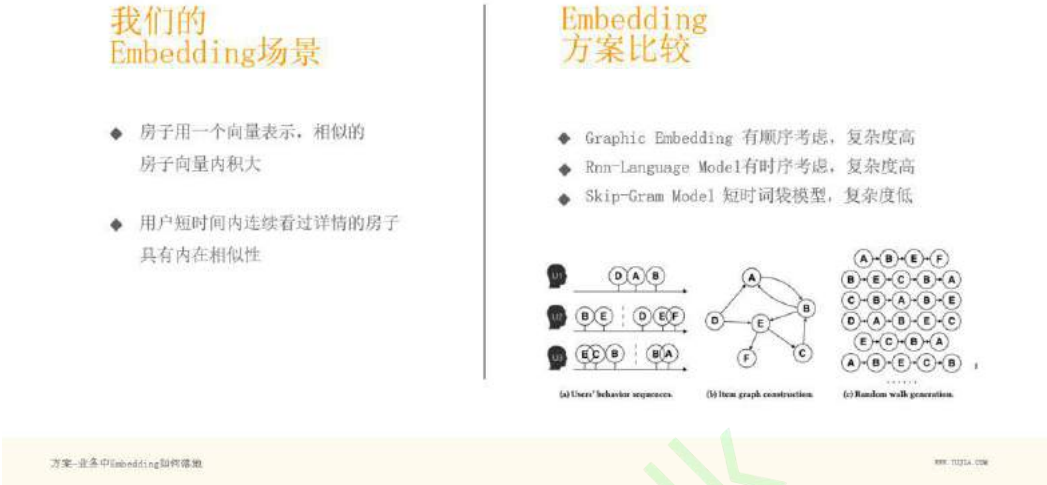
3. 基于 embedding 的 item-to-item 相似推荐

Embedding 可以挖掘商品多维度的潜在个性，通过向量之间的内积，发现物品之间的相似性。

依赖的数据：用户行为日志

特点：不需要数据标注，从高层角度进行推荐；无法覆盖新品，可以挖掘除了地标特征外的多维度相似性。

Embedding



下面介绍下 Embedding：

Embedding 是指把超高维 one hot 离散型变量，表达成一个相对低维的向量（比如：从 100000 维降到 64 维）。

在民宿场景下，把房子用一个向量表示，这个向量的内积越大，就表示相似度越高。

这种向量可以表示房屋的原理是：

我们收集了大量的用户日志，用户在短时间内会浏览很多房子，我们认为，用户在下单前，他的搜索都是基于同一个需求的，所以在短时间内浏览的房子是具有内在相似性的。

Embedding 方案比较

我们考察了几种 Embedding 方案：

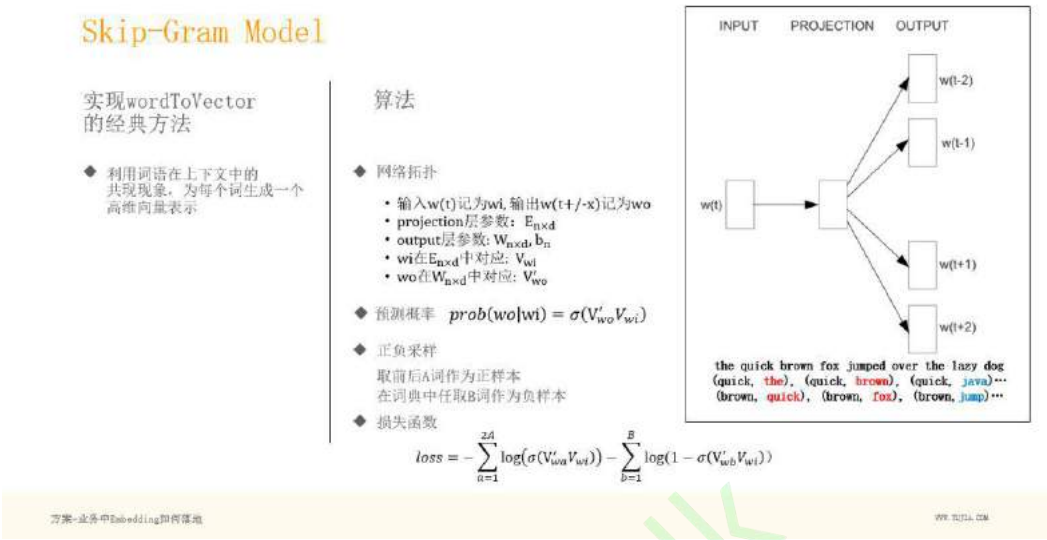
Graphic Embedding：大家可以看到上图（a）：用户搜索时的时序，图（b）：把时序用网络来表达，其中 u1 用户的行为 DAB ，就可以把 D-A-B 连线，多个用户的连线放在一起，就形成了网络。这种方式考虑了用户点击行为之间的顺序，和不同用户行为序列的交叠， 这在我们的场景中没有必要性，且会增加很多计算量。

Rnn-Language Model：这个模型复杂度高，有顺序考虑，在数据集不大时，收益也不明显。

Skip-Gram Model：回归到简洁的模型，其实模型的复杂度是跟我们的数据和真实的场景有关的，有时候需要复杂的模型，有时候可能简洁的模型更好。这个模型简单来说是一个短时词袋模型，复杂度低，很适

合对大量的语料进行建模计算。2013 年 Mikolov 等人正是回归到简单的 Skip-Gram 模型开启了 NLP 领域的新阶段。

Skip-Gram Model



简单介绍下 Skip-Gram Model：

它是实现 word to vector 的经典方法，利用词语在上下文中的共现现象，为每个词生成一个低维向量表示(几百维)。

我们可以看右边的网络图，非常简洁，输入为 w(t) ，中间层为高维的 projection 层，最后为输出层，可以采用 sotfmax 、也可以采用二分类，Skip-Gram 采用了二分类，通过负采样的方式，避免了对大量输出的分类。

下面为一个语言的例子，输入为 quick ，输出为 the 或者 brown ，然后做了负采样 java ，这就构建了正负样本，可以进行学习了。

网络参数：

Projection 层：E_{nxd} ，这个就是我们说的 Embedding ，n 指 n 个词，d 指我们想把这个词表达成多少维的向量。

Output 层：W_{nxd} ，也是每个词都有 d 维的参数，然后还有一个偏置 b_n

我们用 wi 表示输入，wo 表示输出，V_{wi} 表示 Embedding 向量。

预测概率：

可以看到这里我们要预测某个词出现在上下文中的概率，就是求解在给定输入词 wi 的时候，wo 的概率是多大。

正负采样：

这里 Skip-Gram 采用的是正负采样，取前后 A 词作为正样本，在词典中任取 B 词作为负样本。为什么取负样本？因为不能只去学习正样本，两个词之间不是所有属性都相近，需要负样本来学出来到底哪些属性相近，哪些属性不相近。

损失函数：

$$loss = -\sum_{a=1}^{2A} \log\left(\sigma(V'_{wa}V_{wi})\right) - \sum_{b=1}^{B} \log(1 - \sigma(V'_{wb}V_{wi}))$$

Skip-Gram Model 实现 HouseToVector

接下来把 Skip-Gram 落地到民宿场景中，把一个个民宿表达成向量。

如右图，左边为自然语言的一个句子，可以看到白框 The cat plays with the dog 是一个正常的句子，后面俩个黄色的词其实和这个句子无关 ,我们把它采样进来做一个负例。学习的时候 The 作为输入 ,cat 和 plays 作为正样本，car 和 driving 作为负样本，这样构成一个学习样本去学习它的 Embedding 。在我们的业务中，这个序列其实就是

一个个房子，就是用户在列表页看到的房子，并且是点击过的房子，黄色为用户跳过的房子，这样就和我们的语言模型对应上了，也有正样本和负样本，图中蓝色的框为用户的 book ，即用户的下单。可以看出，用户的潜在需求与短期内点击过的房子具有相似性，与跳过的房子不相似，与下单房子的相似性具有高权重。

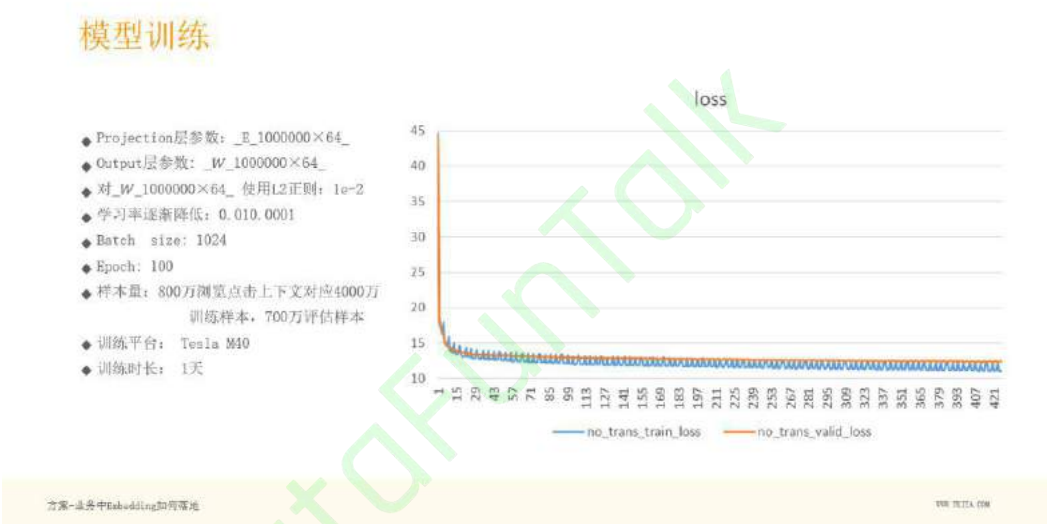
可以看下，上图左边的概述，就是我们如何把 Skip-Gram Model 如何映射到民宿场景中的：

- 连续的浏览点击行为作为单一需求上下文 w(1),w(2)...w(t)....
- 若两个行为间隔在半小时以上，则分到两个不同需求上下文中
- 一个上下文中点击的房子作为正样本 wa ，跳过的房子作为负样本 wb
- 当天下单的房子作为当天所有上下文的正样本 wa ，且权重更大(当作 5 个点击)
- 正样本中任何一个房子可以作为输入 wi
- 其前后 2 个正样本及下单房子作为输出中的正样本 wa
- 采样一个上下文中跳过的 8 个房子作为输出中的负样本 wb

- 补充采样上下文中目的地(可能多个)的若干个房子作为负样本 wb

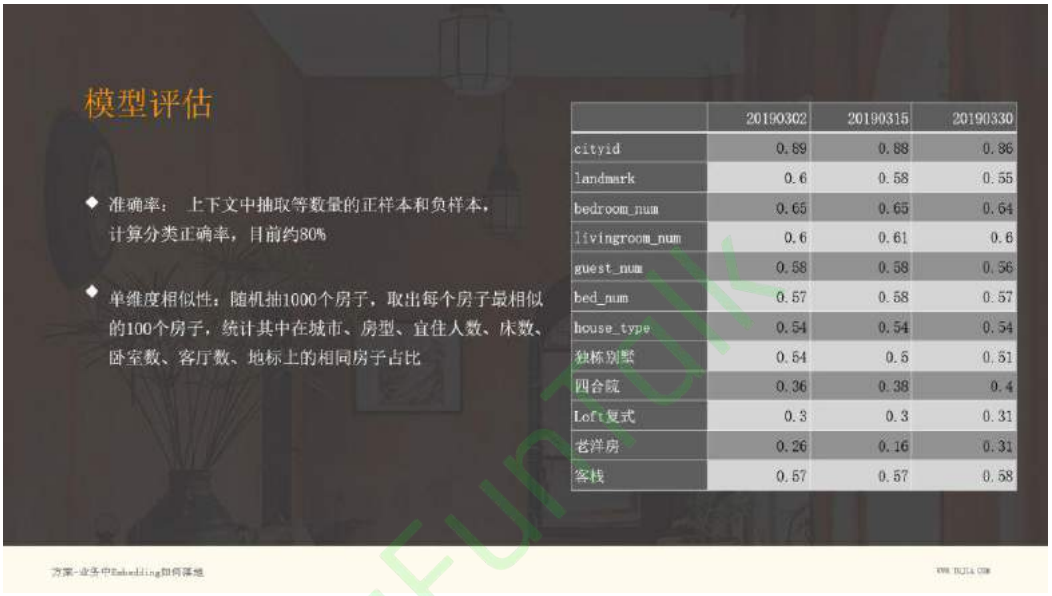
- 一个训练样本包括，输入：1 个房子，输出： 64 个房子

模型训练



说一下我们的模型训练，百万级的房源，用 64 维的向量来表达，参数其实就是 E 矩阵和 W 矩阵，对 W 矩阵进行 L2 正则化，Batch size 为 1024，使用了 100 个 Epoch ，训练了 1 天，训练的样本量是由 800 万浏览点击上下文对应的 4000 万训练样本，700 万评估样本，采用 Tesla M40 训练平台。

可以看下我们的 loss 曲线，黄色为评估集，蓝色为训练集，可以看到这个曲线是比较符合预期的，没有太大的抖动，说明这样的方法合适民宿 Embedding 建模。



	20190302	20190315	20190330
cityid	0.89	0.88	0.86
landmark	0.6	0.58	0.55
bedroom_num	0.65	0.65	0.64
livingroom_num	0.6	0.61	0.6
guest_num	0.58	0.58	0.56
bed_num	0.57	0.58	0.57
house_type	0.54	0.54	0.54
独栋别墅	0.54	0.5	0.51
四合院	0.36	0.38	0.4
Loft复式	0.3	0.3	0.31
老洋房	0.26	0.16	0.31
客栈	0.57	0.57	0.58

模型评估除了刚刚的 loss 曲线评估，我们还需要一些产品维度的评估，包括：预测准确率，单一维度相似性。右图为单一维度相似性的对比图。

我们再说个具体的例子，就是途家 app 列表页的一个变化。上图中红直线标出了几个房子位置发生了上移，原因是和作图中用户点击的房子具有相似性，直线上小字是相似度。

这个是我们在线 ABtest 的一个效果，可以看到蓝色是我们的上线日期，上线之后，有一个明显的转化率的提升。

另外我们还做了一个工具提供给供应链的同事使用 ,用来进行联想搜索,左上角是输入的房子，下面是根据 Embedding 最相似的若干房子：

三、改进

1. 落地中的一些技巧

关于训练集和训练参数：

- 过滤掉停留时长太短的点击行为
- 过滤掉点击数量太多的用户行为
- 上下文不能太长：+/-2 个点击， 行为间隔<30 分钟
- 下单参与到用户当天所有上下文中

负采样：

- 一定要采样用户跳过没点的房子，数量不能比正样本多太多
- 一定要采样同目的地的其他房子，数量和用户跳过房子相当

学习率：

- 足够小，自适应降低

L2 正则化：

- 针对 W ，采用较强的约束，batch-related：只把当前 batch 里出现的房子加入到 L2 正则项里。

2. 应用 Embedding 中的一些技巧

数据后处理：

- 预测前需要对所有 Embedding 向量规范化
- 过滤掉在训练集中总出现次数小于 10 的房子

推荐方案：

- 对用户一天内最近 5 次点过的房子向量求均值，再求与候选房子的相似度

冷启动问题：

- 找到与新上房子最接近的房子小集合，对小集合中的 Embedding 向量求均值作为该新房子的 Embedding 向量
- 小集合确定方法：在距离，房型，价格，图片分，面积，人数等已有数据方面尽量接近

3. Embedding 的迭代更新

说下比较重要的一点，就是训练一次生成百万级向量之后，过了一段时间，由于民宿增长非常快，每天都有很多新的民宿进来，过一段时间，我需要重新训练，这个代价是比较大的，就是说如果用新的训练样本去训练，可能以前的房子最近没什么人看，那么它的向量可能会丢失，我们的做法：

收集最近 2 月的用户行为日志，生成新训练样本

预加载上一版模型参数：

$$E_{n\times d}, W_{n\times d}, b_n$$

新增的 p 个房子补充到参数矩阵末尾随机初始化：

$$E_{(n+p)\times d}, W_{(n+p)\times d}, b_{(n+p)}$$

只用新训练样本进行训练，导出 Embedding 矩阵同步到线上推荐模型。

这样方式的特点：

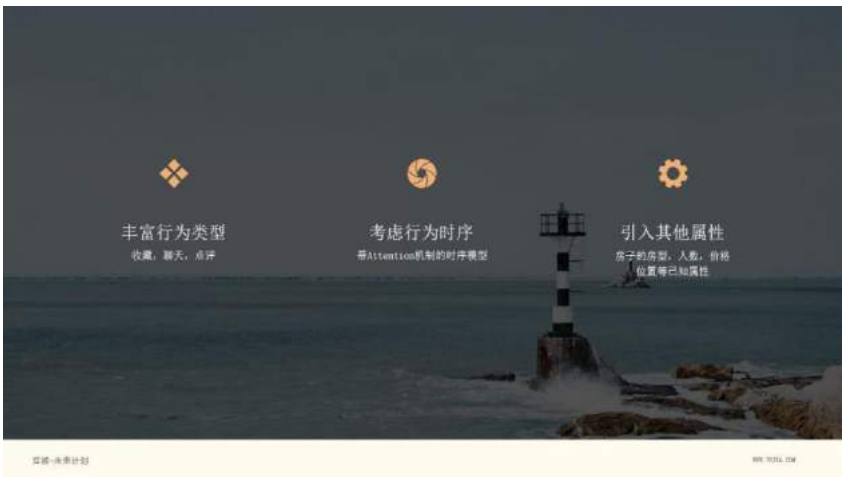
- 老的房子向量可以存留和持续更新
- 及时加入新的房子向量

4. 预加载和重新学习的效果对比

对比维度	预加载后训练	重新训练
cityid	0.929	0.59
landmark	0.442333333	0.225
guest_num	0.489	0.39433333
bed_num	0.482333333	0.38533333
bedroom_num	0.567666667	0.44033333
livingroom_num	0.599	0.53366667
house_type	0.479	0.374
普通公寓	0.614	0.52033333
联排别墅	0.214333333	0.073
Loft复式	0.202	0.087
别墅单间	0.104333333	0.08033333
客栈	0.515333333	0.38733333
度假村	0.124666667	0.059
木屋	0.006	0
独栋别墅	0.367666667	0.20266667
酒店式公寓	0.306	0.21766667
叠拼别墅	0.068333333	0.0395

然后我们对比一个房子，假设这个房子在新的训练集中不存在，只在老的训练集中存在，通过右图，可以看到，第二列是预加载后训练的模型，第三列是重新训练（不预加载参数进行训练），发现，如果把原来老房子的向量加到重新训练的 Embedding 矩阵中，其实没意义，可以看到第三列的相似度差了很多，基本上像在随机乱猜。可见预加载训练很有必要。

四、后续



我们优化的未来计划：

- 引入丰富的行为类型，包括收藏，聊天，点评；聊天其实是比较重要的，用户下单前往往会和商户简单聊几句，如果发生聊天，我们会认为这是一个更重的正样本。
- 考虑用户的时序行为，引入 Attention 机制。
- 引入其他属性，作为副信息和 Embedding 一起学习，顺便把其他属性的 Embedding 也提取出来。

基于行列式点过程的推荐多样性提升算法

嘉宾：陈拉明　编辑：李岩哲

一、推荐系统目标

推荐系统的目标主要包含两个方面：**Exploitation** 和 **Exploration** 。

在 **Exploitation** 中最重要的是 **Relevance**（相关性 ）的计算，其根本思想是根据用户浏览、观看和收藏的内容等用户行为数据推测该用户可能采取的行动。常见的推荐算法大多是基于针对该目标的优化而展开的。

然而用户行为数据在现实中很可能过少、不足以全面地体现用户的兴趣。这一现象在冷启动等场景中很常见。此时推荐系统还有责任挖掘用

户尚未表现出的兴趣，并且避免由于现有行为数据过少而导致推送内容相似性过高的情况。这就需要引入 **Exploration** 。

Exploration 主要有三个方面：

1. 覆盖度：被推荐给用户的内容占全部内容的比例应该较高，特别是新的内容能够有机会展现给用户。

2. 惊喜：推荐的内容并不与用户之前的行为明显相关，但又是用户所喜欢的。这能很大程度提升用户体验，但却难以给出衡量指标。

3. 多样性：在短时间内不要过多地向同一用户推荐同一类型的内容，而是混合各种类型的内容推荐给用户。衡量这一指标主要通过三个方面，接下来将逐一介绍。

二、如何衡量推荐内容的多样性？

2.1、Temporal Diversity（ 时间的多样性 ）

Temporal Diversity

- Recommendations vary over time
- Measured by number of different items per time window
- Our efforts
  - Item-based CF
  - Behavior time decay
  - Impression discount

推荐结果应随着时间的迁移发生改变，其衡量的指标是在固定的时间间隔内推荐不同类的内容的个数。比如一个推荐系统在一段时间内给用户推荐了 10 个内容，那么这 10 个内容中属于不同类别的个数，即可衡量推荐系统的多样性。

对于这个指标的提升主要有三个方式来提升这个指标：

第一个类似于 Item-based CF 的思想，预先根据所有用户的历史偏好数据计算内容之间的相似性，然后推荐与该用户的喜好相类似的内容。

第二个是针对用户的行为做一个时间上的衰减，这样能够针对老用户增大他观看新类型结果的变化。

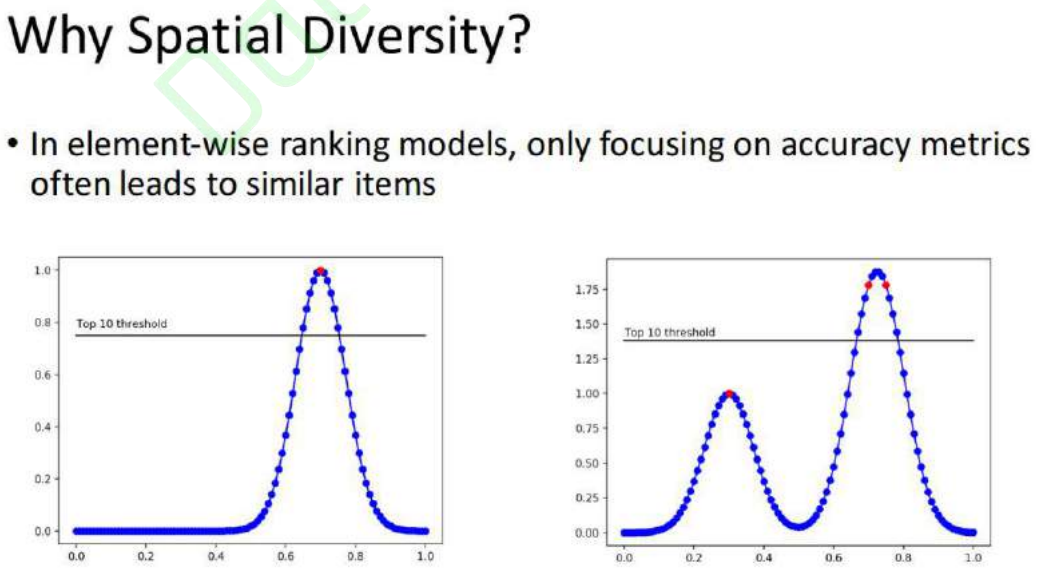
第三个是 Impression discount（ 印象折扣 ），统计所有推荐给用户的内容中哪些是用户没有观看的类型，降低该类型的曝光度，从而给其他类型的内容增加更多的曝光机会。

2.2、Spatial Diversity（ 空间的多样性 ）

它的衡量指标是单个推荐列表中物品之间的差异程度，可以通过计算在同一个推荐 list 中两两 Item 之间的相似度的平均值来进行衡量。

接下来我们将详细介绍该方面内容：

首先我们为什么关注这样一个指标呢？这是因为在推荐系统中我们只关注准确性指标的话，那么会导致推荐出来的内容大部分都相似。

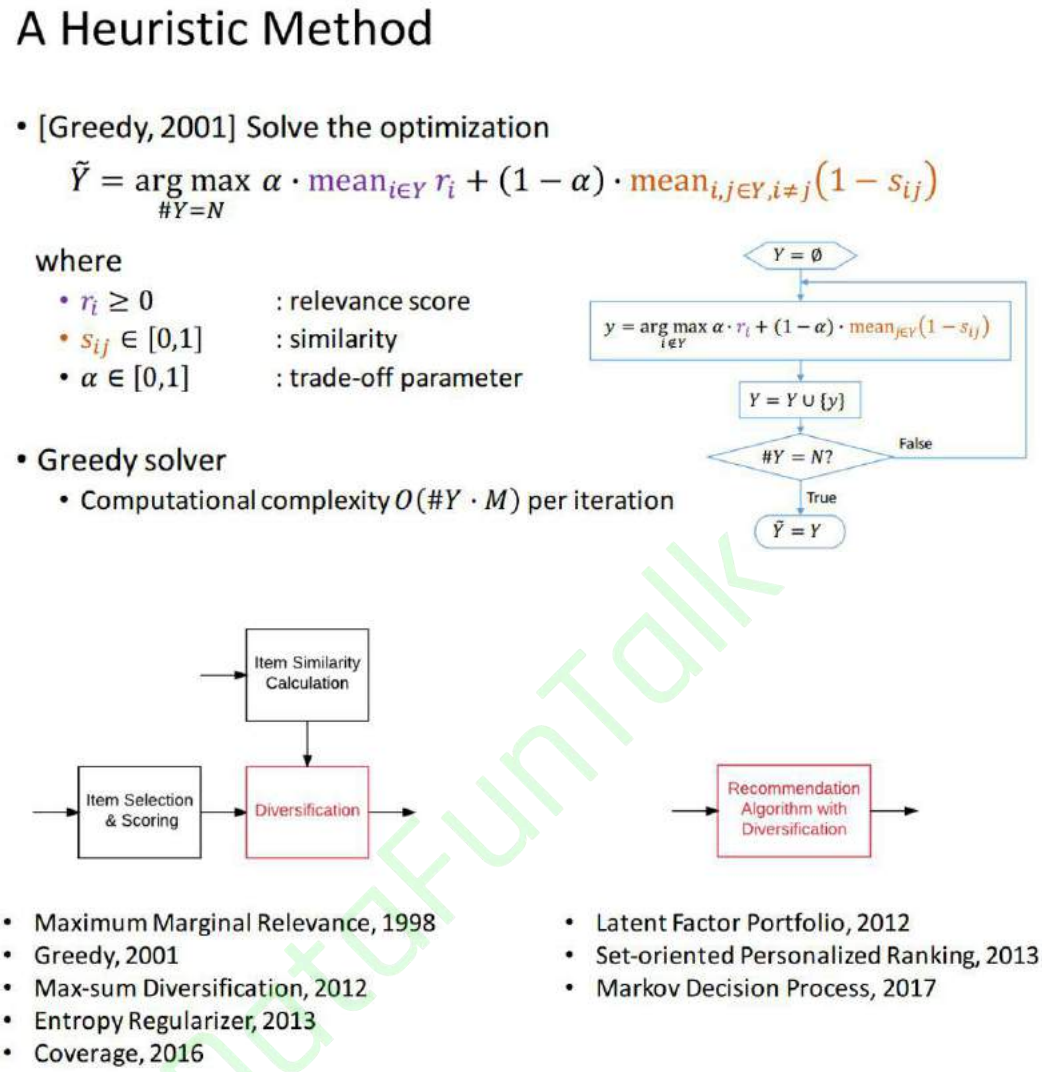


在上面这幅图中，每一个点代表一个 Item ，横坐标表示物品之间的相似性，横坐标越近表示物品的相似性越高，纵坐标表示推荐系统对 Item 的打分。

在左图中有个用户观看了一个 Item 用红点表示，那么推荐系统会根据这个行为推荐 10 内容给用户，那么这 10 个内容和这个 Item 相似度非常高。

在右图这个例子中，一个用户观看了三个内容，其中 Item1 和 Item2 的相似度非常高，和 Item3 的相似度非常低，那么推荐系统根据用户的行为推荐 Top10 的内容的话，会发现这 10 个内容都与 Item1 和 Item2 相似。这是个很常见的例子，比如一个用户很喜欢看漫威的电影，也喜欢看一些文艺类的电影，其中用户观看漫威的电影比较多一些，看文艺类的电影少一些，那么推荐系统很容易造成推荐的时候只推荐漫威类的电影。

用户多样性的问题也已经被广泛研究过。传统上使用启发式的方法，它会在多样性和相关性之间用一个加权平均的方法来获得一个总体的优化目标，然后两两之间比较当前推荐的差异性，然后试图最大化这个总的平衡了之后的优化目标，用穷举的方法。



提升推荐系统多样性的方式主要有两类。第一类主要是依靠 Item 相似度模型和 Item 排序模型。如图 5 左侧所示。另一类主要是直接去修改推荐算法，使得这个推荐算法推荐出来的 Item 尽可能的不相似，如图 5 右侧所示。

三、行列式过程推荐多样性提升算法

DPP 的构造

行列式点过程（Determinantal Point Process，DPP）是一种性能较高的概率模型。DPP 将复杂的概率计算转换成简单的行列式计算，并通过核矩阵的行列式计算每一个子集的概率。DPP 不仅减少了计算量，而且提高了运行效率，在图片分割、文本摘要和商品推荐系统中均具有较成功的应用。

DPP 通过最大后验概率估计，找到商品集中相关性和多样性最大的子集，从而作为推荐给用户的商品集。

行列式点过程刻画的是一个离散集合 Z={1,2,3....,M} 中每一个子集出现的概率。当给定空集合出现的概率时，存在一个由集合的元素构成的半正定矩阵 $L \subseteq R^{M \times M}$,对于每一个集合 Z 的子集 Y ,使得子集 Y 出现的概率 $P(Y) \propto \det(L_Y)$ ，其中，L_Y 表示由行和列的下标属于 Y 构成的矩阵 L 的子矩阵。可以看看下面的例子：

Determinantal Point Process (DPP)

- For DPP, there exists a PSD $M \times M$ matrix $\mathbf{L}$ such that

$$\mathcal{P}(Y) \propto \det(\mathbf{L}_Y)$$

- Example: $M = 2, Z = \{1,2\}, \mathbf{L} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$,

$$\begin{array}{ll} \mathcal{P}(\emptyset) & \propto 1 \\ \mathcal{P}(\{1\}) & \propto 2 \\ \mathcal{P}(\{2\}) & \propto 1 \\ \mathcal{P}(\{1,2\}) & \propto 1 \end{array} \Longrightarrow \begin{array}{ll} \mathcal{P}(\emptyset) & = 0.2 \\ \mathcal{P}(\{1\}) & = 0.4 \\ \mathcal{P}(\{2\}) & = 0.2 \\ \mathcal{P}(\{1,2\}) & = 0.2 \end{array}$$

由于矩阵 L 是半正定的，因此存在矩阵 B ，使得 $\mathrm{L} = \mathrm{B}^{\mathrm{T}}\mathrm{B}$ ，并且

$$P(Y) \propto \det(L_Y) = Vol^2(B_i), i \in Y$$

这是因为行列式为方阵中的各个列向量张成的平行多面体体积的平方。

为了将 DPP 模型应用于推荐场景中，考虑将每个列向量 B_i 分解解为 $\mathrm{B_i} = \mathrm{r_i f_i}$ ，其中：ri 为 item i 与 user 之间的相关性，且 $\mathrm{r_i} \geq 0$ ；

$\mathrm{s_{ij}} = < f_i, f_j >$ 为 item i 与 item j 之间的相似度度量，且 $\| f_i \|_2 = 1$ ；

那么：

$$L_{ij} =< B_i, B_j >=< r_i f_i, r_j f_j >= r_i r_j s_{ij}$$

$$P(Y) \propto \det(L_Y) = \prod_{i \in Y} r_i^2 \det(S_Y)$$

从矩阵 L 的构造可知，商品与用户之间相关性越大，且商品之间多样性越丰富，则矩阵 L 的行列式越大。因此，我们可以建立如下最优化问题：

$$Y_{map} = \text{argmax}\{\det(L_Y)\}$$

但是，直接求解该优化问题是 NP 难的，陈拉明团队则利用贪婪算法，提出了一种能加速行列式点过程推理过程的方法。

首先，DPP 取 Log 后的函数是满足次模函数的：

$$f(Y) = \log\det(L_Y)$$

次模函数是一个集合函数，随着输入集合中元素的增加，增加单个元素到输入集合导致的函数增量的差异减小。即，对于任意

$$i \in Z, X \subseteq Y \subseteq Z \setminus \{i\}$$
$$f(X \cup i) - f(X) \geq f(Y \cup i) - f(Y)$$

直观解释为，小集合和大集合增加同样一个元素，小集合带来的收益大于大集合的收益。

因此，可以将上述优化问题转化为贪婪的形式：

$$y = \arg\max\{f(Y \bigcup i) - f(Y)\} = \arg\max\{\log(\det L_{Y \bigcup i}) - \log(\det L_Y)\}$$

即，每次选择收益最大的 item ，直到满足条件为止。

DPP 模型求解

求解该优化问题时，每次迭代的计算复杂度来源于行列式的计算，而求行列式的计算复杂度与该行列式长度的三次方成正比，即 $O(My^3)$ ，这一结果显然不适用于实际线上实时性较高的场景。下面，叙述论文中所做的改进：

首先对子矩阵 L_Y 做 Cholesky 分解，使得：

$$L_Y = VV^T$$

其中，V 是一个下三角矩阵。对于任意 $i \in Z \setminus Y_g$ ，对子集 Y 添加一个元素 i 之后的子矩阵做 Cholesky 分解，使得：

$$\mathbf{L}_{Y \cup \{i\}} = \begin{bmatrix} \mathbf{L}_Y & \mathbf{L}_{Y,i} \\ \mathbf{L}_{i,Y} & \mathbf{L}_{ii} \end{bmatrix} = \begin{bmatrix} \mathbf{V} & \mathbf{0} \\ \mathbf{c}_i & d_i \end{bmatrix} \begin{bmatrix} \mathbf{V} & \mathbf{0} \\ \mathbf{c}_i & d_i \end{bmatrix}^T$$

其中，有以下等式成立

$$\begin{aligned}\mathbf{V}\mathbf{c}_i^T &= \mathbf{L}_{Y,i} \\ d_i^2 &= \mathbf{L}_{ii} - \|\mathbf{c}_i\|_2^2\end{aligned}$$

两边取行列式后再取 log ，可得：

$$\log\det(\mathbf{L}_{Y\cup\{i\}}) = \log\det(\mathbf{L}_Y) + \log d_i^2$$

应用 Cholesky 分解后，每次迭代只需要计算 $\mathbf{c}_i$ 即可。而为了得到 d_i ，先需要求解线性方程组：

$$\begin{aligned}\mathbf{V}\mathbf{c}_i^T &= \mathbf{L}_{Y,i} \\ d_i^2 &= \mathbf{L}_{ii} - \|\mathbf{c}_i\|_2^2\end{aligned}$$

求解得到 $\mathbf{c}_i$ 后，再带入 $d_i^2 = \mathbf{L}_{ii} - \|\mathbf{c}_i\|_2^2$ 得到 d_i 。此过程的计算复杂度来源于求解线性方程组，虽然求解线性方程组的计算复杂度也是三次方，但是系数矩阵 V 是下三角矩阵，因此，每次迭代的计算复杂度可降到二次方。

即使计算复杂度降到了二次方，但是相比于目前主流的算法，可能依然没有优势。因此，作者又考虑每次迭代也用增量的方式更新 $\mathbf{c}_i$ 和 d_i ，从而避免了求解线性方程组带来的计算复杂度。具体过程如下：

For any $i \notin Y \cup \{y\}$,

$$\begin{aligned}\mathbf{V}\mathbf{c}_i^T &= \mathbf{L}_{Y,i} \\ d_i^2 &= \mathbf{L}_{ii} - \|\mathbf{c}_i\|_2^2\end{aligned} \quad \Longrightarrow \quad \begin{bmatrix}\mathbf{V} & \mathbf{0} \\ \mathbf{c}_y & d_y\end{bmatrix}\tilde{\mathbf{c}}_i^T = \mathbf{L}_{Y\cup\{y\},i} = \begin{bmatrix}\mathbf{L}_{Y,i} \\ \mathbf{L}_{yi}\end{bmatrix}$$
$$\tilde{d}_i^2 = \mathbf{L}_{ii} - \|\tilde{\mathbf{c}}_i\|_2^2$$

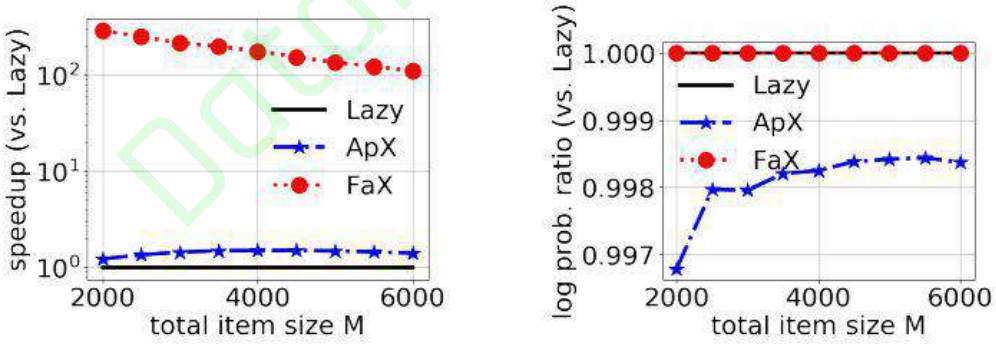
将带入上式中，推导可得：

$$\tilde{\mathbf{c}}_i = \begin{bmatrix} \mathbf{c}_i & \left(\mathbf{L}_{yi} - \langle \mathbf{c}_y, \mathbf{c}_i \rangle\right) d_y^{-1} \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{c}_i & e_i \end{bmatrix}$$
$$\tilde{d}_i^2 = \mathbf{L}_{ii} - \|\mathbf{c}_i\|_2^2 - e_i^2 = d_i^2 - e_i^2$$

因此，每次迭代的计算复杂度进一步降低至一次方。

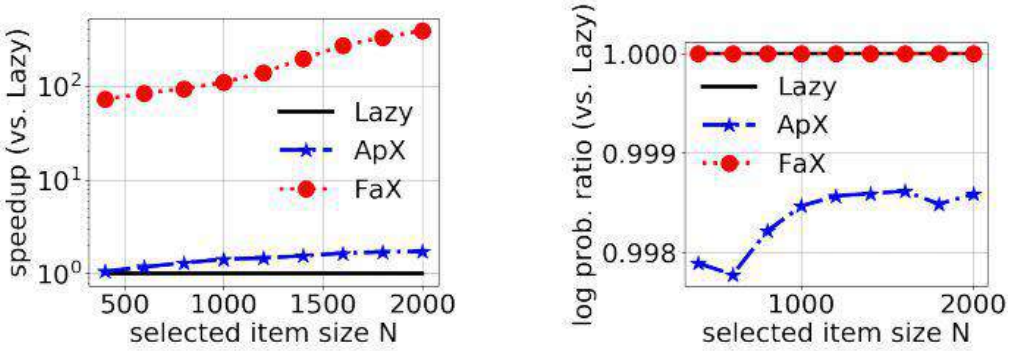
四、实验结果

- Efficiency comparison, $N = 1000$



我们假设给用户推荐 1000 个内容 ,假设物品的数据是从 2000 到 6000，我们可以看到我们的加速效果可以达到 100 倍量级 ,并且没有任何性能损失。

- Efficiency comparison, $M = 6000$



另外一个是我们固定总的 item 数到 6000，来变化 N 从 500 到 2000，加速效果也是在百这个量级，性能损失也是没有的。

- By varying the trade-off parameters, algorithms are compared on
  - Netflix Prize
  - Million Song Dataset

  in terms of
  - Mean reciprocal rank (MRR)
  - Intra-list average distance (ILAD)
  - Intra-list minimal distance (ILMD).

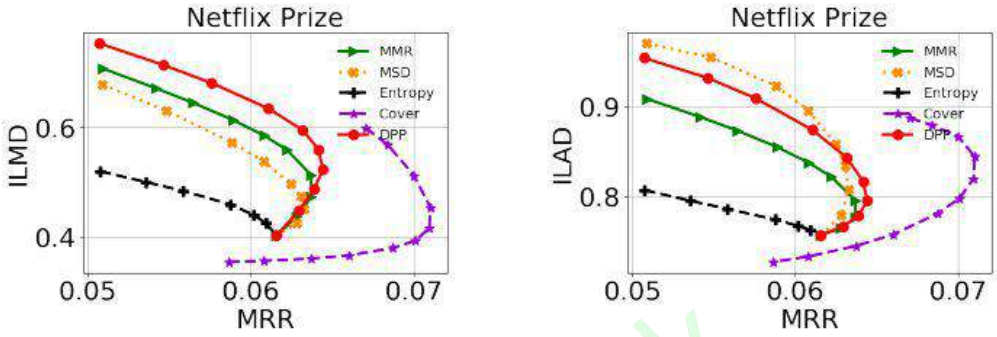
这个是我们另外的一个实验，在俩个数据集上跑我们的模型，并且对比 MRR、ILAD、ILMD 三个指标。

- Comparison of average / upper 99% running time (in milliseconds)

Dataset	MMR	MSD	Entropy	Cover	DPP
Netflix Prize	0.23 / 0.50	0.21 / 0.41	200.74 / 2883.82	120.19 / 1332.21	0.73 / 1.75
Million Song Dataset	0.23 / 0.41	0.22 / 0.34	26.45 / 168.12	23.76 / 173.64	0.76 / 1.46

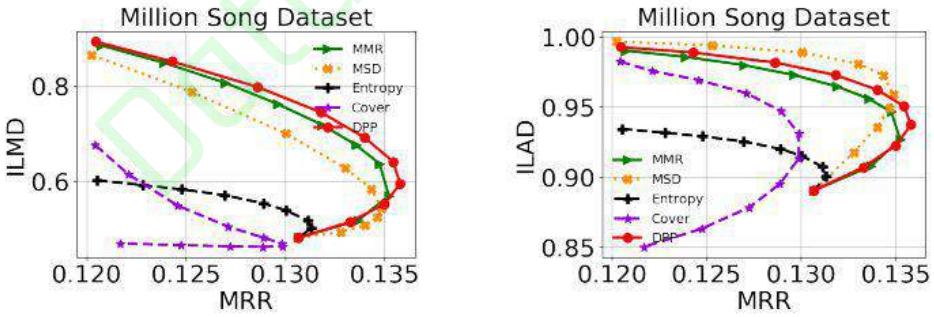
上图为各算法在运行时间上的对比。

• Accuracy vs. Diversity



横坐标代表相关性，纵坐标代表多样性，在这个数据上我们的算法优于其他三个算法，而 Cover 的性能是表现最好的。

• Accuracy vs. Diversity



而在这个数据集上 Cover 算法性能又非常差，而我们的算法在这个数据集上表现还是不错的。

五、总结

基于行列式点过程的推荐多样性提升算法使用贪婪算法推理最优的行列式点过程，并利用 Cholesky 加速行列式点过程的推理。该算法在推荐领域具有较好的应用，在丰富推荐多样性和相关性的同时，大大提升了计算速度。

DataFunTalk

网易新闻推荐：深度学习排序系统及模型

嘉宾：薛海霞　编辑：郑智献

大家好，今天分享的主题是：网易新闻客户端信息流的个性化推荐中，深度学习排序系统及模型，这也是我们团队在研发实践中的一些经验总结。

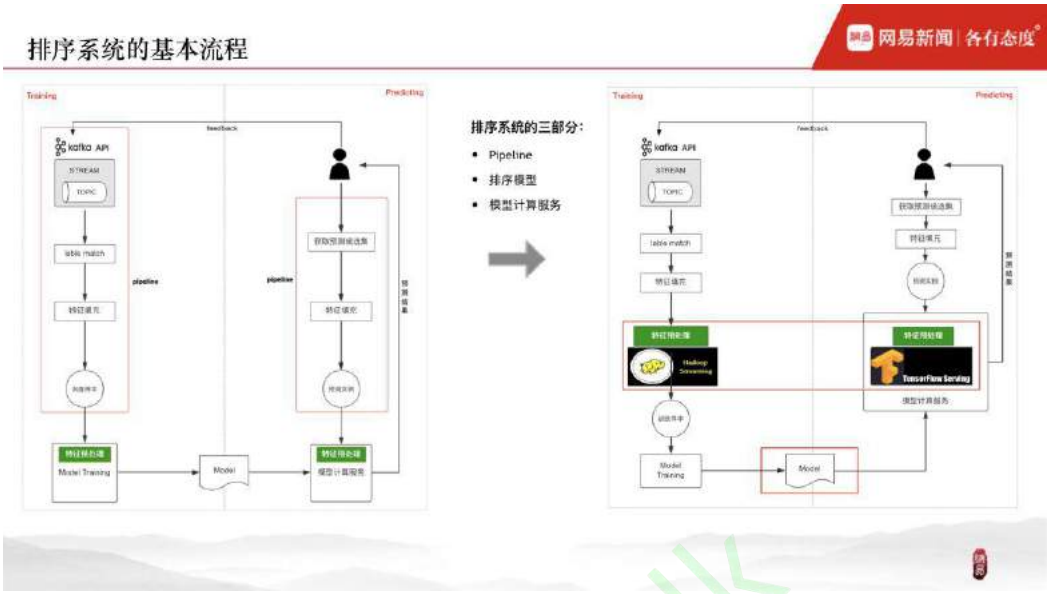
首先看一下在信息流场景中，个性化推荐的产品形态。左边是网易新闻的头条频道，右边是短视频频道，在经过召回、排序、重排之后信息流的最终呈现。作为流量较大的两个频道，我们在对它构建排序系统和排序模型的时候，通常会根据文章、视频、图像相关的内容特征，用户画像，场景，关注关系等对内容候选集进行排序。接下来，问题来了，如何衡量排序的效果呢？线上优化指标通常包括：CTR、阅读时长、留存、刷新次数等，其中最容易建模的就是点击率了，如果要用一个范式进行表示，就是图中这个概率公式。

接下来考虑一个问题，当面临一个具体的推荐业务场景时，如何高效地构建排序系统和排序模型？

一般来说，排序系统由三部分构成：pipeline、排序模型、模型计算服务。

线下阶段，pipeline 将客户端传过来的实时反馈日志，按照不同 session 进行聚合，曝光和点击数据进行 label match，然后经过特征填充生成原始样本，接着特征预处理后，生成用于模型训练所需的训练样本。线上阶段的 pipeline 跟线下基本相同，不同之处在于各自依赖的上游数据源，一个来自客户端的实时反馈，一个来自召回请求及预测候选集。

下图为排序系统的基本流程。然而，在具体实践中，处于对各模块的计算平台、流程一致性、组件通用性以及模型训练性能等各方面的考虑，我们需要对这个 basic 的排序系统做对应的改造。而我们的改造主要集中在两块：特征预处理 & 模型。



为什么要对基础流程进行改造以及具体如何改造？这里需要了解一下整个排序系统研发过程中我们所面临的核心问题和挑战。

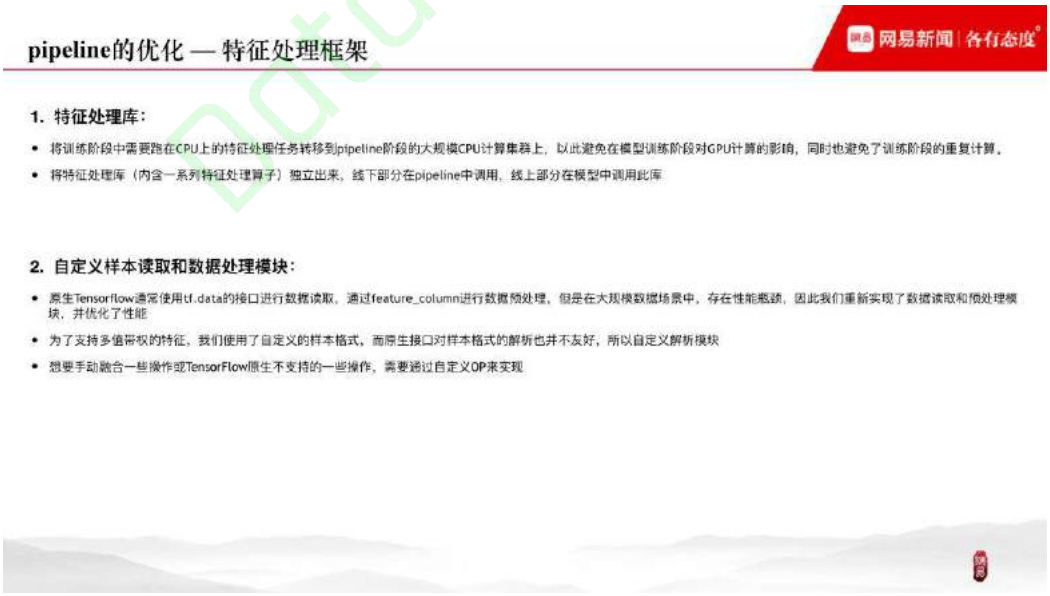
首先关于 pipeline，我们知道，排序模型在训练和预测之间的差异会对模型的准确性产生很大的影响。而差异的产生来源于：特征、特征处理。只有保证 pipeline 线上线下的一致性，才能保证线上模型能够得出正确的预测结果。特征方面，我们将进行在线预测时的特征保存下来，然后填充到线下的训练样本中，这样就保证了训练和预测阶段的特征是一致的；特征处理方面，实践中一般采用 tf.data 和 feature_column 进行读取样本数据和特征处理，这种做法虽然避免了线上线下一致性的问题，但是这种基于原生 TensorFlow 的实现也带来了线上计算和线下训练的性能问题。如何在特征处理模块即保证线上线下的一致性又能同时保障系统整体的性能，这是我们所面临的一个挑战。

然后对于深度学习排序模型，我们需要支持网易新闻不断增多的业务需求，这就要求排序模型框架有足够的通用性和可扩展性，以支持模型的快速迭代和迁移。同时要求模型有足够好的灵活性，以支持业务定制化。

针对 pipeline 中的特征处理模块以及排序模型框架，我们做了如下优化：

首先特征处理库，将模型训练阶段跑在 CPU 上的特征处理任务转移到 pipeline 阶段的大规模计算集群上（Haddop Streaming），以免影响模型训练阶段对 GPU 计算的影响，同时也避免了训练阶段的重复计算。基于这样的思想，我们将特征处理库从模型中独立出来，线下阶段在 pipeline 中而非模型中调用此库，线上阶段在模型中调用此库，但是这样会带来线上线下不一致的问题。我们的解决方案是将特征处理库独立出来之后，线上和线下调用对应的我们自定义 OP（样本读取和数据处理模块）即可。

关于自定义 OP，我们主要对样本读取和数据处理模块进行了对应的改造。原生 TensorFlow 通常使用 tf.data 进行数据读取，通过 feature_column 进行数据预处理，但是在大规模数据场景中，存在性能瓶颈，因此我们重新实现了数据读取和预处理模块，并优化了性能；另外，为了支持多值带权的特征，我们使用了自定义的样本格式，而原生接口对样本格式的解析也并不友好，所以自定义了解析模块；或者想要手动融合一些操作或 TensorFlow 原生不支持的一些操作，也需要我们通过自定义 OP 来实现。



完成 pipeline 的底层改造之后，如何快速使用特征处理框架呢？很简单，在配置文件中将特征处理流程描述出来即可。

在配置文件中，对于每个特征，描述它对应的特征处理流程。如下图，圆形代表原始特征值和经过各个算子处理后的特征中间值，方形表示特征处理库

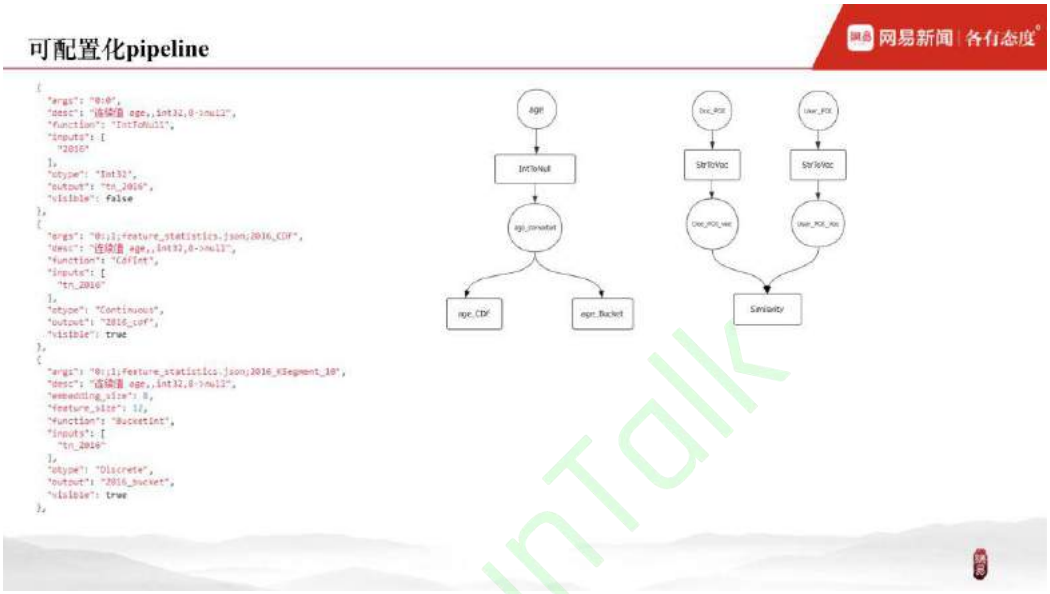
内的算子，所以特征处理流程可以视为各个 DAG 图。右边的两个例子是用 DAG 图来描述特征处理流程的相关配置。

1）特征 age，先经过校正算子（将上游传入为 0 的 age 特征字段校正为 Null）得到中间值 age_corrected，再经过算子 CDF 或者 Bucket 处理得到可以作为模型输入的特征值 age_cdf 和 age_bucket。

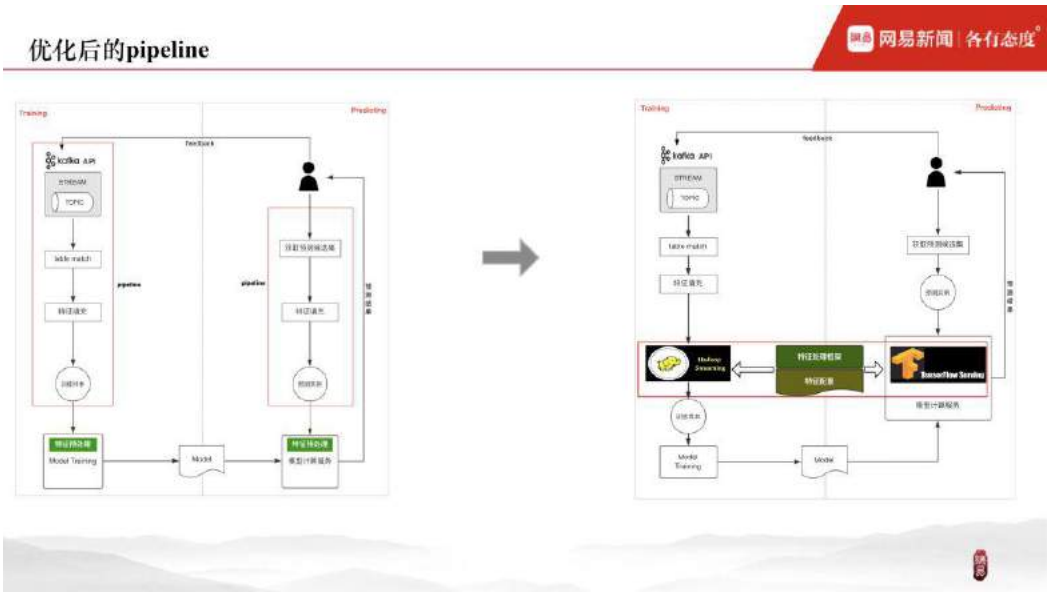
2）特征 Doc_POI 和 User_POI，经过算子 StrToVec 处理之后再经过相似度算子，得到二者的相似度。

以上经过一系列算子处理后的特征将作为排序模型的输入。若在训练样本中增加特征，只需要在特征处理框架中实现对应的算子，并在配置文件来描述新增特征的处理流程。

我们使用 DAG 图来描述特征处理过程的思想，源自于 TensorFlow，先实现底层算子 API，然后将对特征的处理流程构建 Graph，再通过上层 API 接口 session.run 启动运行 Graph。而解析配置文件的过程就相当于解析 DAG 图。

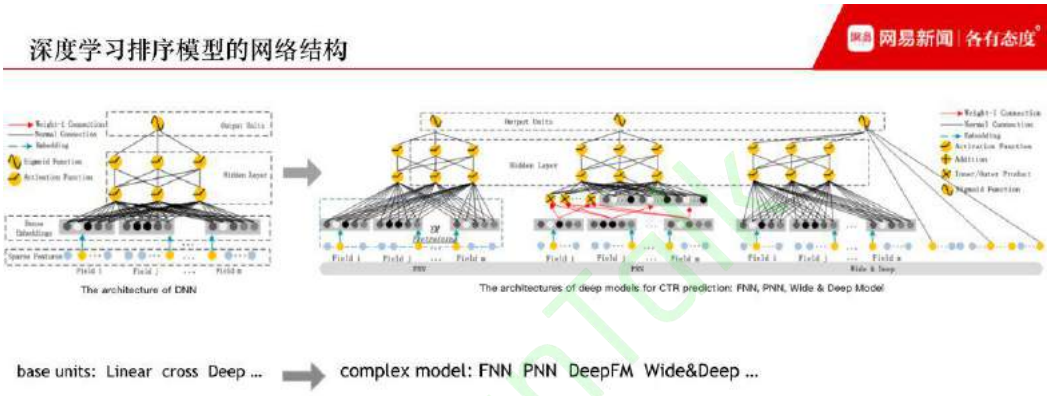


至此，完成 pipeline 的优化。回看一下，排序系统经过改造之后的模样。排序系统变成由一个特征处理框架进行特征处理，而这个框架既可以跑在线下阶段的 hadoop streaming 上，也可以跑在线上 TensorFlow 深度学习框架之上。



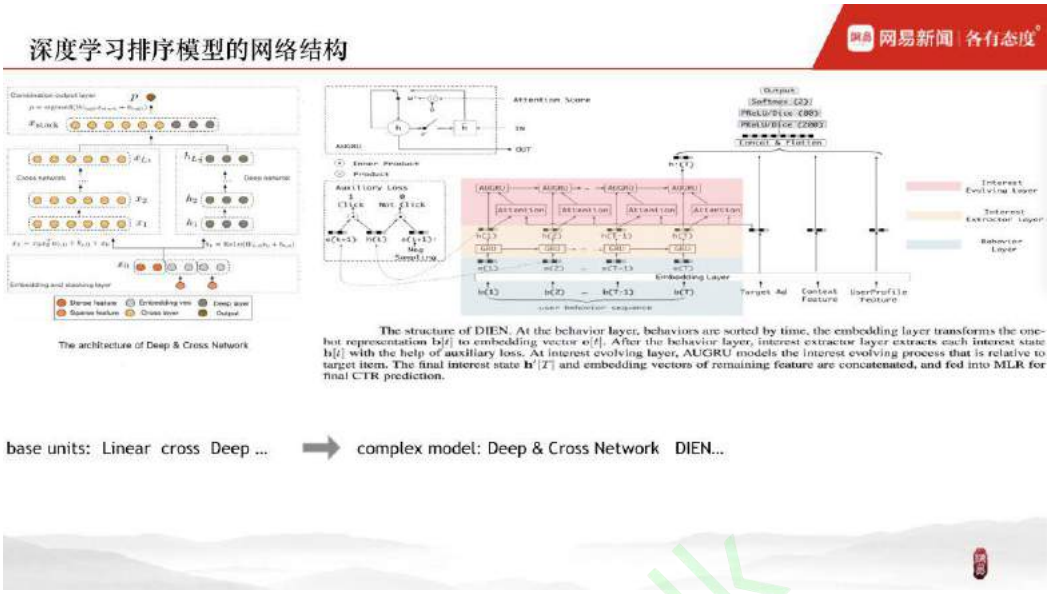
深度学习排序模型方面，同样面临一系列的问题。如何构建通用 & 可扩展的推荐算法库框架，来支持新的业务场景，保证模型的快速迭代？如何保证框架的灵活性，根据变化的业务需求对模型做定制化？如何通过高度可配置的方式来构建模型？

先看几个经典的深度学习排序模型的网络结构，左边是 DNN，右边为它的几种变体：FNN、PNN、Wide&Deep。虽然 4 种模型结构各异，但抽象一下会发现共通之处，基本单元包括：linear、cross、deep 模块，从层次结构来看包括: 特征表征层、特征交叉层、全连接层，其中特征交叉既可以通过人工方式（Wide 部分 ）也可以通过内积或矩阵乘法的方式来进行。经过以上分析，其实提供了一种思路：我们可以将深度学习模型的各个基本单元进行模块化来构建模型。



沿着以上的抽象和分析，接下来用两个网络结构相对复杂的深度学习排序模型来进行验证。

下图左边为 DCN（Deep & Cross NetWork）模型，右边为 DIEN（Deep Interest Evolution Network），可以观察到，层次结构同上：特征表征层、特征交叉层、全连接层，与上面不同的是：特征交叉不再受限于人工交叉或各种方式的 2 阶交叉，而是让 NN 网络去进行更高阶的特征交叉表达。网络的基本单元仍然包括：linear、cross、deep。

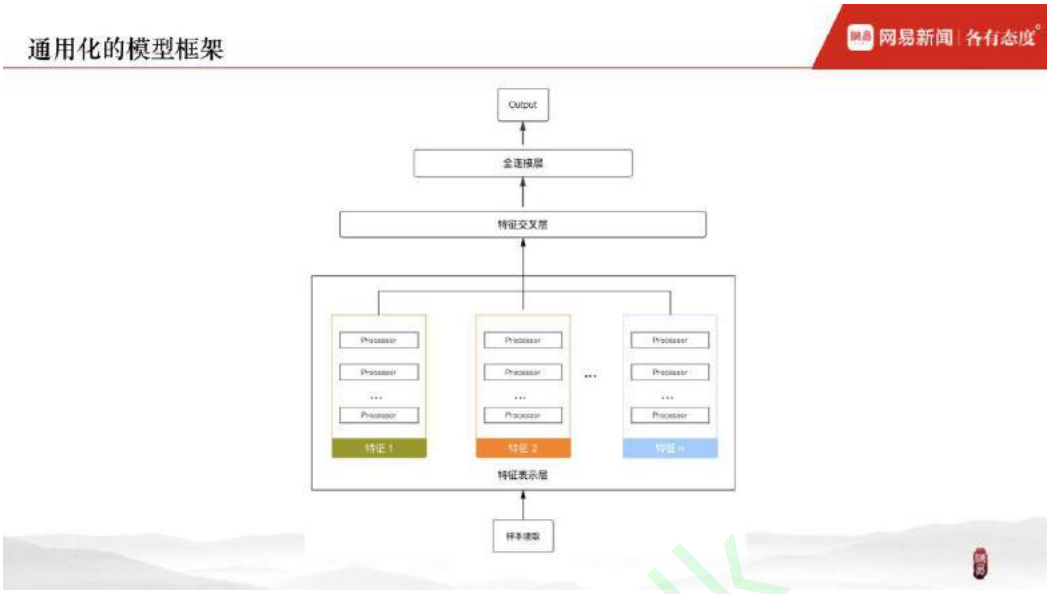


根据前面对几种深度学习排序模型的网络结构的分析，总结起来如下表所示：

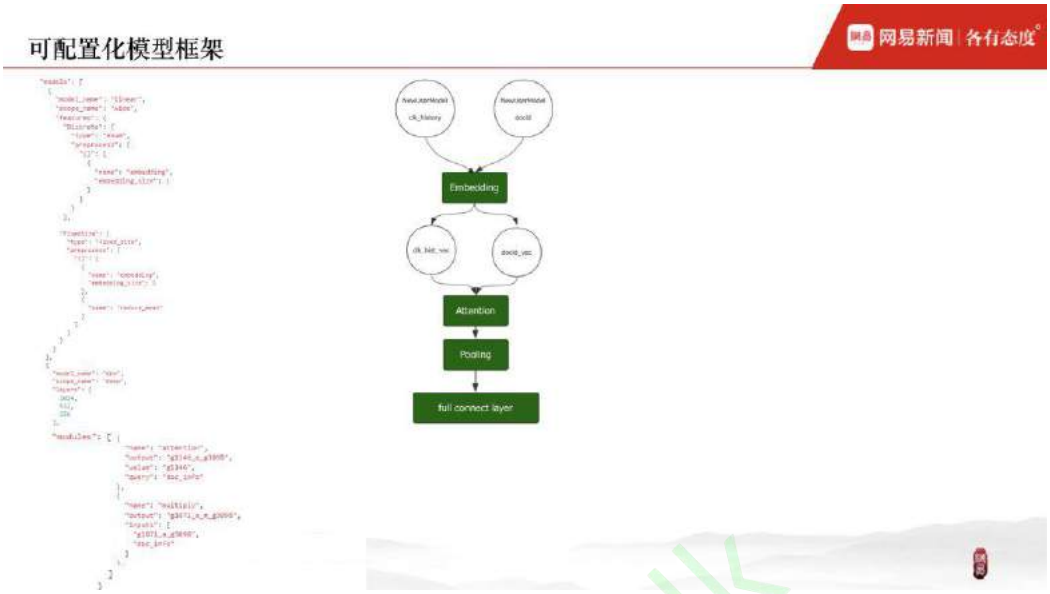
输入层到 Embedding 层，该过程把大规模的稀疏特征通过 embedding 操作或其它表征方式映射为低维稠密的 embedding 向量；对于生成的 embedding 向量，我们可以对它进行包括特征交叉在内的其它各种操作，比如：concat、sum or average pooling 等操作，大部分排序模型改造主要集中在这一层；最后将经过各种处理之后的 embedding 向量输入至全连接框架中。

有了深度学习排序模型的通用范式，我们便针对网易新闻推荐业务设计了通用模型框架。

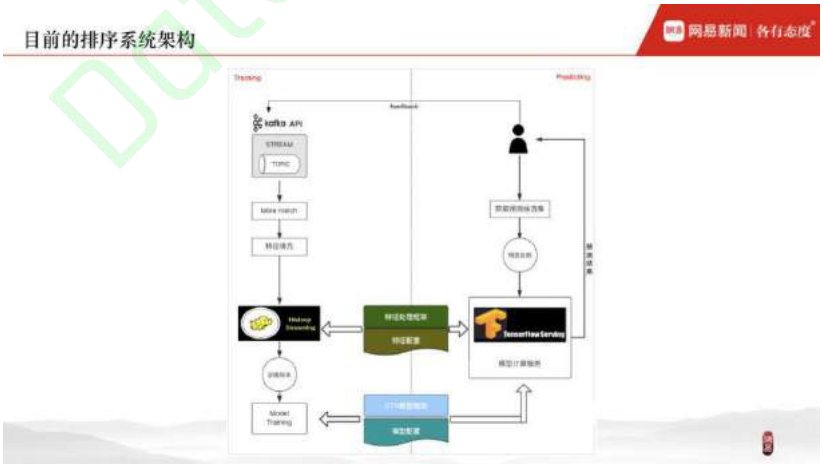
用层次结构图来表示的话，即下图：



与特征处理框架的思路相同，这里对模型框架也采用子模块可配置化的方式。比如输入特征为用户点击历史 clk_history 和目标 docid，首先两个特征经过 Embedding 后得到对应的向量表示，再经过 Attention 得到权重分配，随后 Pooling 操作将变长向量转化为定长，最后输入全连接网络。底层实现各个模型子模块，具体的业务场景对应的模型只需在其配置文件中将模型结构描述出来即可。



经过对 pipeline 及模型框架的优化，我们的排序系统最终能够很好的解决了 pipeline 线上线下一致性、系统整体的性能问题、深度学习模型框架的通用性&可扩展、灵活性等问题。



同时模型效果在各个推荐业务中相对 baseline 也有比较显著的提升。

模型效果

网易新闻 | 各有态度

经过AB测试，深度学习排序系统目前已在网易新闻头条的多个推荐场景扩量上线。在上线后一周，其中新用户排序业务的Deep Learning实验线相比baseline点击率提升9.854%，人均浏览时长提升8.0766%。

DataFunTalk

低频少样本长验证周期场景下的算法设计

嘉宾：张惟师　编辑：高勇

本次分享的内容提纲：

业务背景介绍

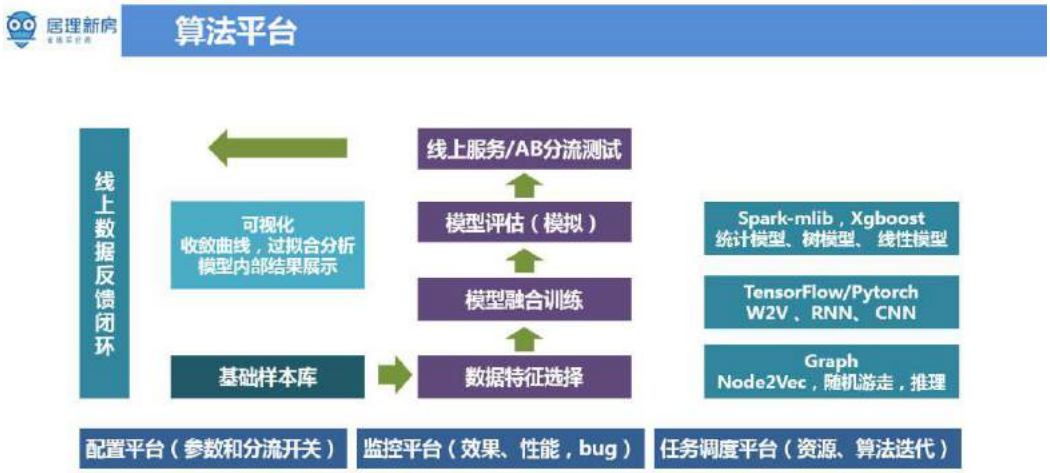
DataFunTalk

工业界中存在算法友好场景，也存在算法不友好的场景。算法友好场景中用户交互信息频繁，样本数量充足，模型验证周期短，常见的应用场景有新闻信息流、广告推荐、快消类商品，搜索排序等；算法不友好的场景中恰好相反，面临用户交互信息少，样本数量不足，模型验证周期长等多种问题，如金融风控、招聘推荐、二手车点电商、房产电商等场景。居理新房业务场景中属于后者，是在产业互联网中一种客单价极高，频率极低的典型算法不友好场景。

居理新房打造了独特的线上、线下的闭环生态系统。线上部分通过投放效果类、品牌类广告（如 SEM， 信息流等），在广告落地页上进行产品展示，App 下载等方式来引流客户。客户在使用相关产品线的产品时会产生一些线索信息。这里的线索可以理解为用户的相关信息（如手机号码等）。线索信息较为混乱，真假不一，因此需要在线客服人员对客户留下的线索进行筛选确认，将具有真实购房意愿的名单筛选出来给线下咨询师。由咨询师联系客户，带领客户预约看房， 客户满意的话，带领客户继续认购、签约等后续环节。从上述环节中可以看到线上获客到线下签约数据量呈漏斗式层层过滤，最后签约量较少。在整个流程中，广告自动投放和线索质量质量评估都有相关算法，这里主要介绍线索质量相关的算法。不同线索的质量不同，对于线索质量高的用户，如果能在恰当的时间点为预测出客户的购房需求，可以提高用户的转化率，对于线索质量低的客户，例如暂时处于观望或者是虚假信

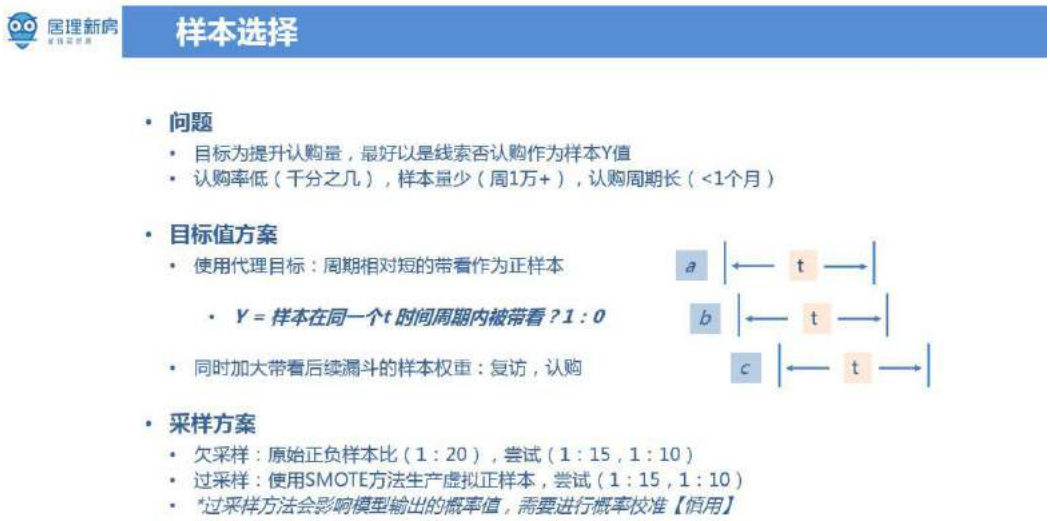
息，如果能尽早的识别出来这一部分群体，可以降低线上客服审核成本，进而提高咨询师的工作效率。这样比较自然的引入用线索质量二分类的算法，将好的线索直接给咨询师，不太好的线索给客服进行审核。但是这种场景下的分类算法面临多种挑战，首先用户交互频次极低，一个人一生平均购买一到两次房，也就几乎不存在历史购买行为信息，其次是样本量少，举个例子，北京市 2019 年上半年新房成交量在 3w 套左右，其中居理新房的成交比例在 2%左右，整体用于训练的样本不多。另外客户决策周期长，从平台首次接待客户到客户完成交易的周期业内均值在 2-3 个月，即使居理新房周期短一些，流程周期也要在 20 天到 30 天，换句话说是跟单周期长，算法上线后，至少跟单一个月才能观察算法效果。客户相关信息较少，目前主要有客户渠道来源（ 头条创意，百度渠道 ），以及客户在 App 上的历史浏览，点击，搜索等行为信息，还有一些线下获取的大量非结构化信息（ 如咨询师与客户的线下交互行为 ）,但这部分数据线上获取不到，一般用来验证算法的有效性。

算法平台



目前居理新房搭建了复用性和扩展性都比较好的算法平台，可以通过一些灵活的配置，实现相关监控，任务调度，模型校验，可视化页面以及 AB Test。算法平台底层支持多种算法引擎包括机器学习模型平台（Spark-MLlib，Xgboost），深度学习平台（Tensorflow，Pytorch）以及图相关模型。可以通过 pipeline 的方法整合数据流和算法引擎。

样本选择



如何选择正负样本？比较直观的做法将最终是否发生认购行为作为正负样本的评估依据。但在当前场景下，由于房屋交易业务转化率低，从线上访问 UV 到最终房屋认购，比例在万分之几左右，只从中间线索角度来看，这个比例也在 2%-3%左右。为了解决样本稀疏问题，这里设置了一个代理目标，将是否发生带看行为作为正负样本的评估依据。带看行为发生在认购行为之前，发生认购行为占带看比例中的 10 分之一左右，周期也可从一到两个月缩减到两周左右。那么样本在一个周期 T 内，将是否被带看作为正负样本的评估依据。另外可以后续模型训练时，提高具有多次带看行为或者发生认购行为的权重。在一个时间周期 T 内，可能存在跟单不完全的情况，但这部分比例在 10%以内，可以忽略。由于正负样本比例差异较大，在样本量较大的情况下，这种比例可以接受，但在样本量较少的情况下，正负样本比例差异导致模型学习困难。因此在训练模型前可以先对样本进行采样预处理。常见的样本采样方法有欠采样和过采样。欠采样是保持数据集正样本数量不变，根据一定

比例去随机抽取负样本,过采样是通过已有正样本来构造虚拟正样本，来减小正负样本差异。常见的过采样方法有 SMOTE 等。但是采样方法会影响数据集中的正负样本分布，在关注概率值的分类等业务场景下，需要对模型输出的概率进行校准。

特征工程

在特征工程中，特征类型主要有以下三类，信息来源类（ 渠道/线索来源、多设备、多账号，刷量识别 ），用户 App 行为类（ 核心页面点击行为，转换行为，留存行为 ），用户粘性类（ 前 n 日活跃频次，活跃时长，最近一次活跃时间，访问深度、广度)。其中信息渠道来源是指用户是来自百度、Feed 还是广点通等平台，多设备、多账号等主要是用于衡量用户是否发生过作弊行为。另外用户 App 行为类特征是占比最大的一类特征，主要是指用户在

留电话号码前，在 App 上点击，浏览、搜索等行为。用户粘性类特征是一些抽象统计特征，其中访问深度是指 App 页面用户访问最长的路径，广度是指 App 中用户使用的功能的数量。

在 Deep Learning 的场景中，一般来说用 Embedding 就 ok 了，不需要较多的特征处理，但在当前场景下特征处理相对来说比较重要。常见的数值处理有 IV（离散化）、页面比率（页面转化）。什么是页面转化？用户搜索 query 词后 App 显示楼盘信息，如果用户搜索五次后发生一次点击行为,那么对于这个用户来说，该用户的搜索点击的转化率就 20%。特征组合主要有主成分分析（PCA），树模型决策路径，以及完整的功能点击路径。另外还进行了特征有效性分析，重要包括方差，覆盖率分析，互信息，独立性分析，离群值分析等。离群值分析在当前场景中影响较大，因为在样本量较少的情况，离群点容易误导模型。下图是数据集中覆盖率前 10 的特征，包括渠道表示（op_type），城市（city），用户设备数量（user_id_cnt），来源（source），渠道数量（channel_cnt），最近一次渠道来源（last_channel），上一次留电话距离现在的天数（leave_phone_days），App 中 banner 点击的数量（banner_click_cnt），App 启动次数（app_start_cnt）等。

模型调优

模型主要采用的传统模型 LR，RF，XGBoost，LightGBM，也尝试了使用 Deep Learning 等，但实际效果不足传统模型好。另外在当前场景下样本数量少，而且版本更新迭代较快的时候，常遇到数据分布不一致的问题。产品迭代了新的版本，但离线模型训练用的还是老的数据，而线上用的新版数据，这种信息不一致将拉低模型效果。当前采用的方案是时间窗口滚动的方式来训练模型，并尽量剔除一些可能因为版本导致数据不一致的特征，努力将影响降至最低。在无论是深度学习还是传统机器学习，参数调优的方法大同小异，主要是网格搜索（Grid Search）、人工参数调优以及分城市阈值调优。网格搜索调优虽然不能一定找到最优解，但是花费时间较短。另外也尝试了一些贝叶斯优化的方法，它和 grid search 有的区别就是它会根据你上一轮做算法的参数结果去帮你去选择最有可能产生最优算法参数方向去优化这个参数，换句话说做了几轮参数实验并获取对应结果，利用贝叶斯优化自动帮

忙寻找参数优化方向。贝叶斯寻优容易陷入局部最优，需要多进行几轮贝叶斯优化，手工选出里面的极大值。分城市阈值调优是遇到的另一个更严重的一个场景，由于居理新房在全国十多个一二线的城市有业务，而且每个城市有自己不同的特点，用户的行为都不一样，在模型分类时，每个城市应采取不同的阈值。在本身整体数据量就不多情况，每个城市的数据量更少，这种情况下要么将城市信息加入特征或者每个城市都分别训练一个模型。

在做算法时，评估的值不一定是一些常用通用指标如 AUC，Top N 的点击率等，而是与业务比较契合的指标。采用通用指标来衡量算法刚开始可能还可以，但是一段时间后发现对业务帮助不大，指标虽然显示比较高，但最后并没有增加用户的收入，也没有提升用户的留存。只是说单独的就是算法本身还不错，所以很多时候更多需要的是找到真正跟业务结合的优化目标。经过多次思考和试验后，最后选了两类线索的带看召回率（Recall@30%、Recall@70%）的评估指标。模型输出概率需要选取阈值来判断线索是好的还是坏的。可以通过阈值选取，将不同比例的线索标成为好线索。比如说有一百个线索，从里面拿出来 30 个线索就 30%，最后这 30 个线索被带看或成交的比例能达到 60%。就相当于 30%里面有 60%的比例是被召回出来了，这阈值选取的还是不错的。因为在现有算法中不可能保证每一个线索都是好的，通过某个阈值大于 70%的线索是好线索。那么在这 70%的线索里边，应该至少能召回 90%的带看或者认购的线索，才能说明这个算法才是有意义的。

另外就是 early stop 的调优方法，比如说这个模型在训练的时候，模型应该什么时候停止训练。一般来说会用到的方法是学习曲线（learning curve），模型在训练的时候可能迭代一百次、迭代一千次的时候看模型的学习曲线有没有有很大的这种变化，如果在进行更多次迭代时发现测试集上效果在在下降，但是在训练集上其实在上升，这就是过拟合现象。从最好结果看，模型在 recall 为 30%的时候，差不多有 60%被带看或认购的比例，在 recall 的为 70%时能达到 91%的这样一个比例 ,从结果上来看还是能达到预期的结果。

Model	欠采样	Params	Recall@30%	Recall@70%
LR	1：15	penalty=L1 C=1	0.541	0.862
LR	1：10	penalty=L1 C=1	0.552	0.883
RF	1：15	max_depth=8 n_estimators=800	0.556	0.891
RF	1：10	max_depth=8 n_estimators=700	0.577	0.896
XGB	1：15	learning_rate=0.02 max_depth=5 num_boost_round=300 early_stopping_rounds=50	0.593	0.901
XGB	1：10	learning_rate=0.02 max_depth=5 num_boost_round=260 early_stopping_rounds=50	***0.605***	***0.912***

可解释性分析

接下来主要讲一下可解释性的分析，因为整个房产行业的客单价都比较高，相当于说每个客户都特别的宝贵。以北京为例，平均每单利润在 12 万到 15 万。若将好线索误分为坏线索，没有为客户好好服务将会导致较大的损失。对于业务人员（ 客服，咨询师 ）而言，模型预估结果会与他们原有的一些工作模式和习惯矛盾，需要向业务人员解释模型预估结果。当算法去服务于业务团队的时候，这种可解释性显得尤为重要。通过 xgboost 计算得到的特征重要性（feature importance）不一定是完全可解释的，它和特征在决策森林中出现的次数相关。但特征在决策森林里面出现的次数越多并不能说明特征越重要。这里采用的是 SHAP 来进行可解释性分析，SHAP 计算的是一个特征加入到模型时的边际贡献，考虑了该特征在所有的特征序列的情况下不同的边际贡献。在 SHAP 图中，纵坐标是特征列表，横坐标是从负数到正数的取值范围，表示对模型输出值的影响。留电口、渠道特征是从 SHAP 方

法来看是最为重要的特征。一般来说通过搜索渠道来的用户，购房的意向比较强烈，这个也和基本认知符合。另一个比较显著的特征 pv_p_select_city，表示切换城市的动作越多，用户质量越差。举个例子：假设有北京和天津的都有业务，客户可能一会看看北京的房子，一会又选择天津的房子，就中间各种摇摆，客户可能也没太想好到底在哪个城市要买，这种情况下需要花费很多精力去说服客户，用户转化成本计较高。留电天数越多，稍微比越低，就相当于一个月之前留的线索和昨天留的线索对比来看，明显是距离现在时间近的线索质量会高一点。如果客户上午留的电话，在下午的时候咨询师就能给客户提供服务，这种情况下转换概率会很大。但并不是所有的特征影响与我们对模型的预期是相同的，比如说特征 history_view_pages（ 用户在 APP 的上 PV 贡献度 ），PV 贡献度越大，线索质量反而比较差。但如果从购房意愿来看也是可以理解的，对于购房意特别强的客户，浏览页面比较集中，更能体现用户的关注点，虽然只使用了不到 App 1% 的特征，但转化率比较好。反而是那些楼盘看的特别多的用户在行为上表现的流程和浏览广度都不错，但质量确实比较低。这种聚焦观点不仅适用于当前场景，也可以推广到产品上的其他场景。

实际效果

从模型效果来看，客户认购量提升了十七个百分点，基本达到了算法预期目标。从认购到带看的目标变化，将周期从两个月缩减到了两周，后续希望能找到一个更好的指标来代替带看，进一步缩短模型周期。另外居理新房还做了很多线下数据的累积。比如说咨询师与客户的电话录音，微信记录，交通行为等。通过这些离线数据能大概分析出咨询师和客户的行为，比如说通话频率，通话时长，微信验证，聊天频率，客户需求响应时长等多种详细指标。目前不同的城市数据累积量不同，等数据量积累到一定程度，可以为不同的城市设置独立的模型。另外模型融合（stacking）是后续优化的方向，看能不能做出更有意思的效果。目前的模型是基于无线数据，PC 数据相对无线来说，用户行为比较少，下一步是跨站整合 PC 和无线的数据。

阿里零售通智能导购推荐技术实践

作者：零售通算法团队　编辑：Hoh Xil

零售通是阿里巴巴 B 类事业群针对线下零售店推出的解决方案，通过与品牌商、本地经销商紧密合作，为零售店提供品质好货，升级社区零售店的便民服务能力，肩负着“共建智能分销平台”和“让百万小店拥抱 DT 时代”的重要使命。在这种 B2b2c 新型的分销模式中，智能导购推荐不仅提供一站式 APP 进货方式，帮助小店主高效便捷地从平台批发进货，进而服务小店周边 C 端消费者，同时承载品牌商推广及自营销，赋能品牌商通过小店高效精准地触达 C 端消费者，通过智能化导购推荐技术，实现 B 端供给和 C 端需求的高效链接和精准匹配。

业务背景

零售通 APP 导购推荐主要包括大促会场、日常频道和猜你喜欢 3 大类核心场景，如下图 1 所示。左侧图为 618 会场入口，包括榜单主会场入口以及行业分会场入口，榜单主会场为小店主提供当季该区域最适合的商品进货清单，供小店主采购决策，行业分会场是品牌商的营销阵地，通过品牌券和红包等营销方式触达更多的小店和 C 端消费者，实现品牌商数字分销渠道下沉；右侧上图为日常频道，包括羚羚发、网红美食、进货洋货，以及精选频道和限时特价等，通过差异化的频道定位，满足不同小店主个性化的进货诉求；右

侧下图为猜你喜欢，具体场景包括 APP 首页、进货单页、凑单页、商品详情页、支付成功页等推荐区块，猜你喜欢贯穿在小店主购前、购中、购后整个采购流程，提供个性化交互式沉浸式采购体验。

图 1 零售通 APP 导购推荐场景

从图 1 介绍可以看出，零售通导购推荐涉及的业务领域比较广泛，通过一篇文章讲清楚整个业务领域所有的问题以及技术实践背后的思考可能不太现实，因此本文后续部分围绕零售通猜你喜欢重点展开介绍，一方面是因为猜你喜欢产品形态在诸如电商购物、本地生活、旅游出行、文化娱乐、在线教育等行业的 C 端 APP 场景比较成熟，大家阅读起来易于理解，更为重要的是我们想与大家一起分享探讨以零售通为代表的 B2b2c 猜你喜欢具有哪些业务特点？遇到了哪些技术挑战？以及我们的一些实践和经验总结。

零售通猜你喜欢贯穿在小店主购前、购中、购后整个采购流程，如下图 2 所示。当小店主打开零售通 APP，首页猜你喜欢提供精准流量分发入口，意图

明确的小店主通过 Query 推荐引导到搜索页，意图模糊的小店通过精选爆品、品牌专辑、类目榜单等逐步引导到商品详情页；通过购前流量引导，小店主逐步进入了购中流程，此时小店主可能在商品详情页找同款，或在凑单页凑足营销活动门槛以及 388 包邮，亦或者在进货单页找相似品，然后在进货单页下单；商品下单后进入了购后流程，支付成功页推荐复购券引导小店再次购买。

图 2 零售通猜你喜欢场景

在零售通 B2b2c 业务模式下，猜你喜欢具有如下业务特点和挑战：

1. 小店主采购心智：一方面区别于 C 端用户偏冲动消费，B 端小店主比较理性，采购行为通常具有计划性和周期性，不同小店主有各自的采购深度、进货习惯，需要采用差异化的推荐策略，针对深度小店需要采用个性化推荐，

基于小店主周期进货习惯以及小店内商品的库存消耗情况，为小店精准推荐补货清单，针对浅度小店，需要结合小店属性和协同过滤，为小店混合推荐潜在商品；另一方面 B 端小店主盈利诉求较强，对平台营销活动和商品的价格比较敏感，在推荐算法设计中，我们需要综合地考虑平台营销门槛以及价格因素对成交转化的影响，例如在包邮、满减满赠凑单推荐中，如何在带有约束条件下实现推荐的商品单 UV 价值最大化。

2. 线下零售特点：线下新零售是一个巨大的市场，小店主进货渠道较多，既可以通过线上 b2b 平台（零售通、新通路、易酒批…），也可以通过线下经销商批发进货，目前各个渠道覆盖的行业类目有差别，提供的商品也有特定的售卖范围，商品的区域化售卖特点比较明显，另外小店进货目的是为了服务周边 C 端消费者，其所在地理位置很大程度决定了小店能辐射的服务范围。如何结合阿里巴巴大数据能力，挖掘小店周边 C 端消费者需求，为小店提供合理的补货建议，基于 LBS 属性的小店商品推荐是一个充满挑战、值得深究的问题。

技术实践

1. 算法架构

零售通猜你喜欢技术架构如下图 3 所示。用户打开零售通 APP，产生的实时数据存在 ABFS，TPP 请求 ABFS、UPS 分析用户意图，通过推荐引擎 BE 召

回商品返回给 TPP，TPP 接着调用 RTP 打分服务进行精排和重排，最后展现给用户。离线和实时排序模型训练和参数调优通过 PAI 和 Porsche 完成，排序模型支持 T+1 和实时更新。

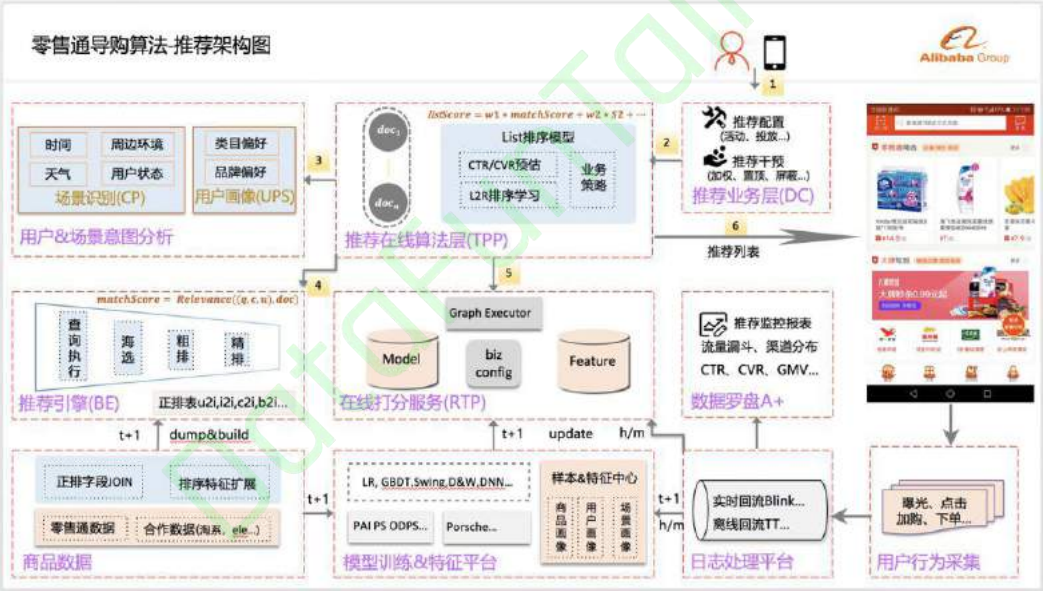


图 3 零售通猜你喜欢技术架构

推荐服务 TPP（The Personalization Platform），阿里个性化推荐平台，TPP 依托阿里中台强大技术架构能力，简化了数据流的调用过程，使得各种推荐算法、数据、模型能够积累、对比和复用，支持实时 AB 报表，支持算法快速迭代上线。

基础特征服务 ABFS（Ali Basic Feature Server），ABFS 提供统一的特征服务，实现用户基础行为特征和统计特征的收集和处理，用于实时训练反馈，实现用户特征共享和特征价值最大化。

推荐引擎 BE（Basic Engine），推荐场景定制的召回引擎服务，支持不同类型的召回数据在线 Union、Join、Filter 等，深度集成和定制了 KNN 库，支持向量索引的分布式构建和查询，提供向量召回能力。

在线打分服务 RTP（Real time prediction），RTP 接收到 TPP 传入的商品列表、上下文特征以及用户特征后，在线 FG 和打分输出，支持多种机器学习模型（PSLR、GBDT…）、深度学习模型（WDL、CNN、RNN、MultiTask…）。

模型训练 & 特征平台，Porsche 在线机器学习训练平台，基于 Blink 的实时机器学习平台，提供了日志处理、特征计算和实时建模，为算法提供端到端的秒级延时的实时模型训练能力。

2. 匹配模型

零售通匹配模型经历了从热门召回，到协同过滤，再到 DeepMatch 模型等一系列优化实践。热门召回虽然比较简单，但具有实践意义，一方面可以直接应用于热门排行榜推荐，另外也为协同过滤、DeepMatch 的算法迭代优化提供 Baseline，我们可以评估算法每次迭代相对 Baseline 的收益大小。

协同过滤主要基于图结构，现实世界中的图主要分为 user-user 同质网络（社交网络、Facebook…），以及 user-object（电商、视频…）二部图网络两种。零售通导购推荐 CF 主要基于 user-object 网络，分为 item-CF

和 user-CF 两种，item-CF 侧重于挖掘 item 之间关系，通过用户历史共现行为计算 item 之间相似度，离线计算方便，应用较为广泛，其中图的相似度计算有很多种[1]，例如 Cosine、Adamic/Adar、Jaccard、PageRank、simRank...，目前零售通 item-CF 主要采用阿里巴巴自主研发的 eTREC 和 Swing，算法原理如下图 4 所示，线上实践证明，相比于传统 item-CF，eTREC 和 Swing 抗噪声更强，并且通过调参能一定程度抑制“哈里波特”和长尾现象。

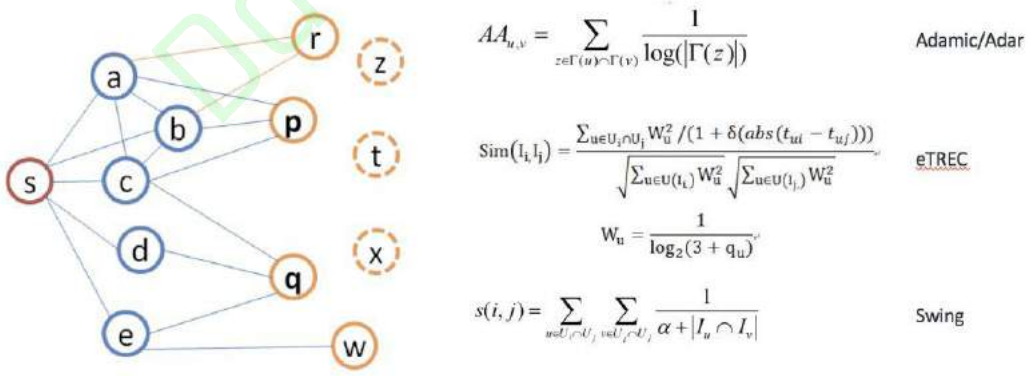


图 4 Item-CF 相似度计算

上述 Item-CF 在零售通进货单相似商品推荐、APP 首页猜你喜欢基于用户实时行为触发 U2I2I 召回等场景，取得了不错的业务效果，但都是基于用户共同行为关系挖掘的 I2I 算法，召回限定在有用户历史行为的商品候选集中，推荐的商品的发现性偏弱，容易导致“越推越窄”的问题，因此我们尝试了 Graph embedding I2I，在 User-Item 二部图加权采样（带权重的随机游走），采用 Skip Gram 模型（DeepWalk[2]、Node2Vec[3]...）训练出商

品的 Embedding，使用语义距离计算得出商品之间的距离，截取 TOPK 相似度商品作为商品的 embedding I2I，目前已经在进货单推荐场景上线。

在 Graph Embedding I2I 的基础上，我们目前正在尝试加入商品的 Side Information，进一步提升 Embedding 的泛化学习能力，更好地解决长尾以及商品冷启动问题，如图 5 所示[4]。

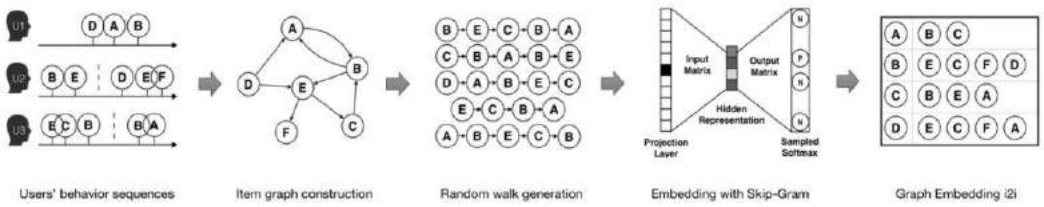


图 5 Graph embedding I2I

前面介绍了 item-CF 在零售通导购推荐场景的应用，同样 user-CF 在零售通导购推荐也有相关业务实践和结果，我们知道 user-CF 多用于挖掘有共同兴趣的小团体，第一节业务背景中已经谈到小店的 LBS 属性非常明显，下面我们介绍如何将传统的 user-CF 与小店 LBS 属性相结合，深挖小店 U2U 相似度模型，为小店主推荐的新颖性更好的商品。

零售通小店有哪些关键 LBS 属性呢？如图 6 所示，我们通过小店 APP 采购行为分析以及线下小店走访调研发现，小店商圈（学校、医院、高端住宅...）和面积（20 平米、50 平米、100 平米...）是影响小店采购的关键因素，商圈刻画了小店周边 C 端消费人群和消费目的，也是商品售卖和品牌营销重要的区域粒度，小店面积则反映小店满足消费者需求的能力，因此同一商圈、

面积差不多的小店在消费需求、商品结构、供给能力、采购习惯等方面相似度较高。

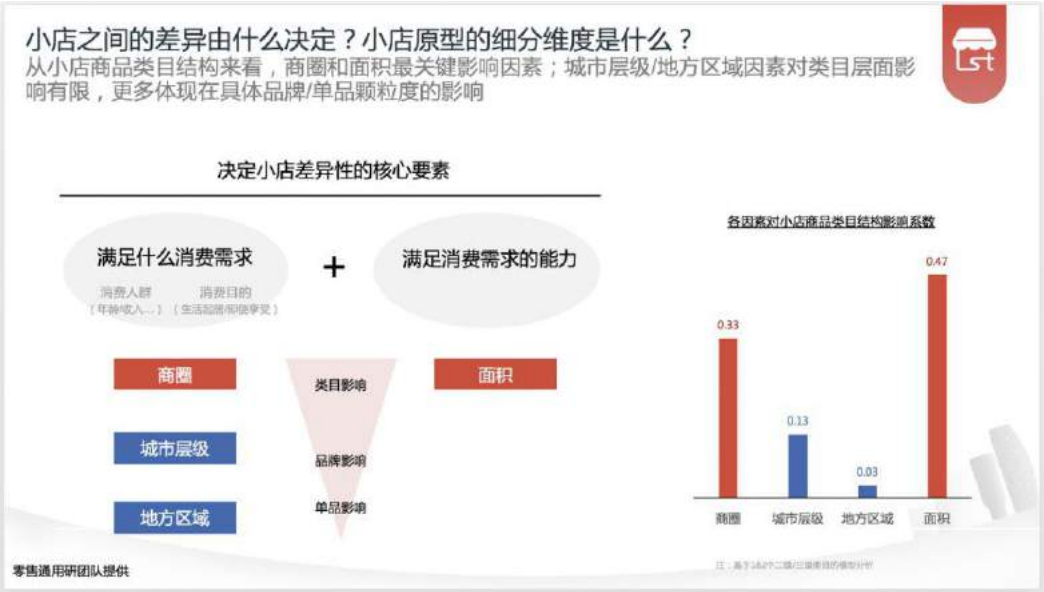


图 6 小店 LBS 属性信息

user-CF 核心是小店相似度计算，首先我们通过小店商圈、城市层级、区域街道、面积等因子，计算小店间 LBS 属性相似度，其次基于小店主在零售通的采购商品序列，通过 Doc2vector 训练小店主的向量表示，通过 simHash 计算向量距离，得出 APP 采购行为相似度，最后综合考虑 LBS 属性相似度、APP 采购行为相似度两方面计算得出小店相似度得分。

$$User_{embedding} = Doc2Vector(seq_i, seq_j, \dots seq_n)$$

$$User_{similarity} = K_1 Score_{embedding-sim} + K_2 Score_{LBS-sim}$$

图 7 小店相似度计算

基于上述思路，我们开发了一套基于小店属性的 user-CF 混合推荐模型，如下图 8 所示，该模型综合考虑了不同生活习惯及气候因素的地域差异、不同消费能力的城市层级差异、同城市不同街道区域的差异，以及不同小店群的进货习惯，在 APP 猜你喜欢取得了不错的业务效果。

图 8 基于小店属性的 user-CF 推荐

前面介绍了零售通导购推荐热门召回、协同过滤相关技术实践，在深度匹配模型方面，我们也正在进行了一系列尝试，如图 9 所示[5,6]，深度匹配对用户和商品在同一空间进行向量表达，通过用户向量在线召回 TopK 个与其近邻的商品向量，得到用户感兴趣的商品,增加 Side Information 能进一步提升用户和商品的 High-Level 表达能力。

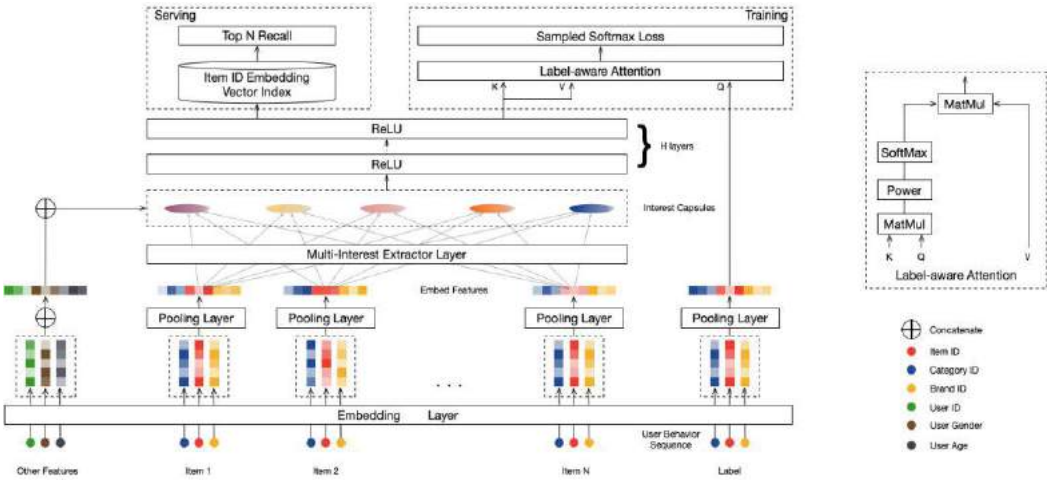


图 9 Multi-Interest Network with Dynamic Routing（MIND）模型

3. 排序模型

零售通导购推荐排序包括粗排、精排、重排三个阶段，由于召回商品集数量较大，考虑到性能因素，需要通过粗排截断选取 topN 商品传递给精排，精排对 topN 商品针对特定的业务目标精准排序，重排主要实现业务策略及干预，下面我们重点介绍精排模型。

深入理解业务目标、扎实的特征工程以及细致的模型选型和调参，对于排序模型迭代至关重要。零售通导购推荐业务目标为 CTR、CVR、SKU 宽度等，基于上述目标，我们重点展开在特征工程和模型演进方面的相关工作。

排序模型的特征可以归纳为以下几大类：

离散特征：小店 id 类，如小店 id、区域 id 等；商品 id 类，如商品 id、类目 id 等；上下文类，如星期几、天气等；以及上述 id 类特征的交叉组合。

连续特征：主要包括小店、商品的基础统计特征，小店-商品的交叉统计特征，以及小店周边 C 端消费者的统计特征。

序列特征：小店周期性采购的商品序列；小店实时搜索 Query 序列、点击的商品序列等。

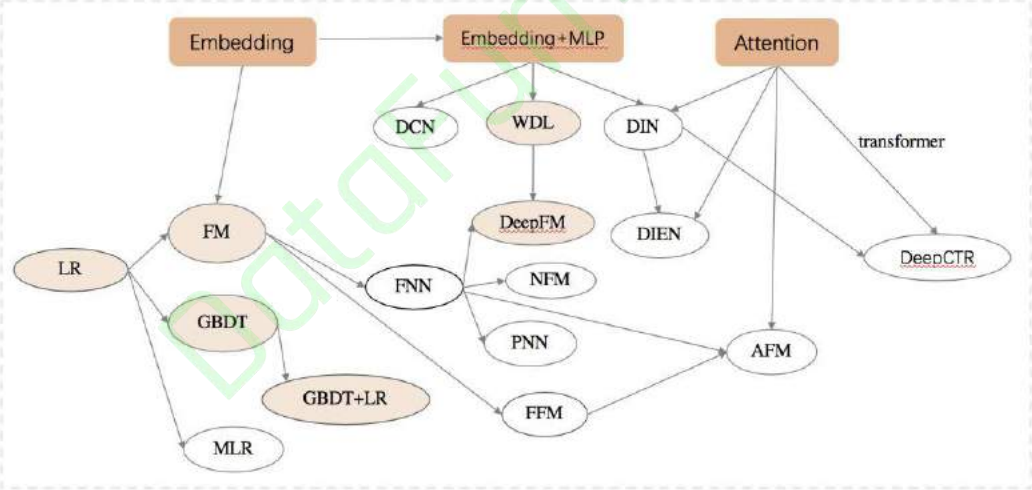


图 10 零售通导购推荐排序模型演进

零售通导购推荐排序模型演进如图 10 所示，传统的机器学习模型（LR、GBDT、GBDT+LR...）通过特征工程持续迭代，目前在已经取得了较好的业务效果，目前作为排序模型迭代的基线。

传统的机器学习模型需要大量的人工经验构建原始特征以及组合交叉，人力成本较高，随着 Deep Crossing[7]、 Wide&Deep[8] 等相继在 2016 年提出，深度学习模型在特征学习表达、模型效果等方面有诸多优势，基于此我们上线了基于 TensorFlow 的深度排序模型（ FM[9]、 WDL、DeepFM[10]... ），挖掘小店主、商品、C 端消费者、以及天气、时间、营销活动等深层次隐含特征关系，提升排序模型的表达能力和泛化能力，另外我们正在开展 Attention、Transformer[11] 在排序模型的相关实践与应用。

特色探索

零售通 APP 货架具有新零售 B2b2c 导购推荐特色，APP 端货架真实还原线下小店内货架的商品结构和位置摆放，通过一站式货架商品组合售卖，提供品类级别的分销解决方案，提升线下小店进货效率，最大化小店经营坪效，同时通过货架链接品牌商资源，打造成为品牌商面对线下消费者营销的主阵地。

零售通 APP 货架导购技术核心是基于 LBS 的精准组货推荐和商品成列算法，基于阿里巴巴大数据中台，结合小店周边消费者线上数据（ 支付宝、淘宝、饿了么... ）和线下数据（ 大润发、POS... ），以及小店主在阿里经济体全域行为数据，运用大数据和算法的力量，推荐最适合线下零售小店盈利增收的商品组合货架，实现“让线下没有难做的生意”。

图 11 零售通导购推荐 - APP 货架

总结思考

本文介绍了阿里零售通智能导购推荐技术实践，从业务背景、算法架构、匹配模型、排序模型、特色探索等方面，我们重点概述 B2b2c 推荐的特点、算法迭代中的思考以及实践经验的总结，当然受限于篇幅，推荐算法其他部分，诸如意图识别、文案展示、营销表达、多物料混排等，我们没有一一展开详述，有兴趣的同学可以线下深入讨论，我们希望借着分享线下新零售推荐算法的机会，与大家一起交流学习，共同进步。新零售面向线上线下亿万消费者，充满各种技术挑战，大家一起努力加油。（报名方式见文末海报）

参考文献

[1] The Link Prediction Problem for Social Networks[2004]

[2] Deepwalk: Online learning of social representations (SBU 2014)

[3] Node2vec: Scalable Feature Learning for Networks (Stanford 2016)

[4] Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. (Alibaba 2018)

[5] Multi-Interest Network with Dynamic Routing for Recommendation at Tmall (Alibaba 2019)

[6] Behavior Sequence Transformer for E-commerce Recommendation in Alibaba(Alibaba 2019)

[7] Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features (Microsoft 2016)

[8] Wide & Deep Learning for Recommender Systems (Google 2016)

[9] Factorization machines (ICDM 2010)

[10] DeepFM: a factorization-machine based neural network for CTR prediction (HIT-Huawei 2017)

[11] Attention Is All You Need (NIPS 2017)

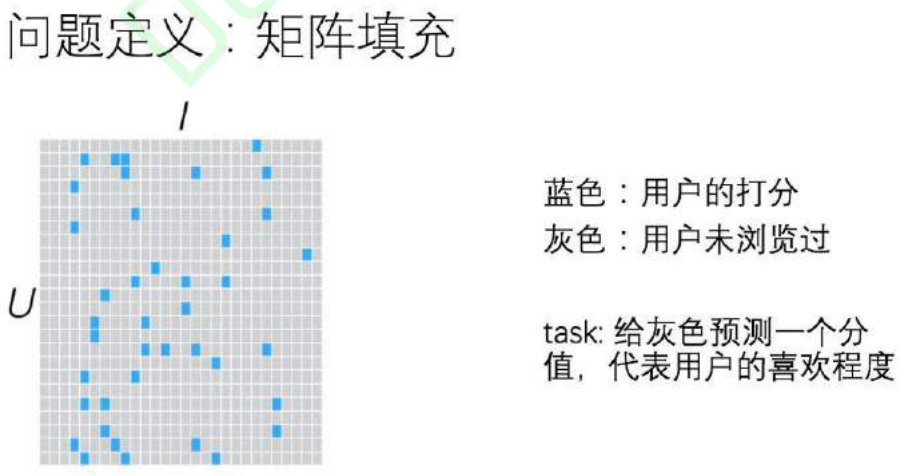
DataFunTalk

推荐技术随谈

嘉宾：翦浩　　编辑：蒋权

大家好，今天想和大家讨论下相关推荐技术通用的特点及在实践中的改进点，这也是我们团队在研发实践中一些经验总结。

首先，推荐解决主要问题是给用户在没有行为的新闻上预测一个偏好概率，然后通过概率值由高到低排序推荐给用户。如图简单地表达了用户在新闻上的稀疏行为矩阵，通过我们会基于用户的画像特征、内容特征、上下文场景特征、用户对新闻的各种操作行为特征等对内容候选集进行排序打分。其中很直观的解决办法是通过计算用户之间相似或者商品之间的相似来推荐。



那么，协同过滤中 UserCF 模型和 itemCF 模型是最简单的办法。这里着重对两者进行对比：

1. itemCF 比较适合个性化推荐场景（比如，商品、电影、短视频等各种场景），它偏重当前用户的当前行为，从而能推荐与当前相关新闻的内容，而 UserCF 更偏群体性（比如，新闻推荐场景），计算当前用户的 K 个相似用户时，当 K 值偏大，UserCF 推荐更偏热点；

2. itemCF 兴趣扩展比 UserCF 弱，因为 itemCF 会根据你当前的行为推荐与当前相关的内容，不像 UserCF 能够学习到多个用户不同的偏好；

3. itemCF 能解决用户的冷启动问题，只要新用户对一个新闻产生行为，就能够推荐和该新闻相关的内容，相反 UserCF 能解决 item 冷启动问题，因为用户相似度计算通常是离线计算的，新闻上线后一段时间，一旦有用户对新闻产生行为，就可以将新上线的新闻推荐给和对它产生行为的用户兴趣相似的其他用户。

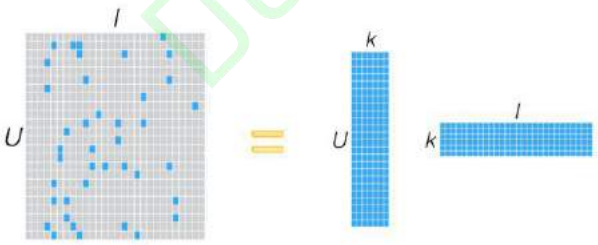
这里，用数学的角度去解释协同过滤为什么能够在推荐中可行。如下基本假设换成数学解释就是，一个矩阵向量，如果行向量部分相同，那么行向量其余部分也有可能相同，代数上解释就是向量线性相关性，那么线性相关带来的是向量低秩，即向量是冗余的，冗余的向量带来的信息和消耗的计算资源是完全可以避免的，那么如何进行矩阵压缩并且保证不影响信息的丢失是推荐技术中的另一个关键。

问题: 协同过滤为什么能够work

- 基本假设：如果一个人A在一个问题上和另一个人B持相同观点，那么对于另外一个问题，比起随机选择的一个路人甲，A更有可能同B持相同观点。
  - 行向量部分相同，其他部分也可能相同
  - 线性代数的术语：低秩
- 低秩->冗余，冗余->压缩，大矩阵可以表示成小矩阵

比如说，矩阵分解。如下一个比较大的稀疏矩阵（低秩）能够拆分成两个比较小的矩阵相乘。公式 E 是由经验损失函数和结构风险函数组成的损失函数。

矩阵分解



$$E=\sum_{i,j:r(i,j)=1} e_{i,j}=\sum_{i,j:r(i,j)=1} y^{(i,j)}-\sum_{k=1}^{K} p_{ik}q_{kj}+\frac{\lambda}{2}\sum_{i,j:r(i,j)=1}\sum_{k=1}^{K}(\|P\|^2+\|Q\|^2)$$

经验损失　　结构风险（正则化项）

其实，基于前面的假设，协同过滤和矩阵分解两者具有很紧密的内在联系，只是两种不同的表达形式。我们需要弄清楚一个问题：超大维度矩阵表式

形式带来的是信息的不相似 ,及稀疏数据计算的相似结果很有可能是不置信并且过拟合的，这样会导致信息准确性不足问题。在特征表示方面 CF 是维持了原有的元数据形式，能够捕捉更细粒度的局部特征，但这样带来的计算开销也是巨大的，所以，实际工业上出现了很多的变体方法。

协同过滤和矩阵分解比较

比较项	CF	LFM
特征维度	百万	百
特征表示	非学习	学习
局部/整体性	局部特征捕捉好	整体性

业界解决 item2vec 的常用的算法是 Spark 中的 ALS 和 Python 中的 Word2Vec。其中 Word2Vec 在序列表征用户和商品向量，然后直接计算相似度是非常熟知的方案。在实际的新闻信息流推荐中，Word2Vec 的点击效果比 ALS 要好 30%+，主要有两个原因：

1. 用户的兴趣和行为是多样的，局部的行为往往更偏相关，往往整体的样本差异是很大的；

2. 在负样本采样中，ALS 是全局的负样本采样，Word2Vec 更倾向高频，倾向高频的采样更不容易让学习出的结果都与高频（头部）的结果相似。

我们思考一个问题，既然样本对模型的效果有显著的影响，那么我们想办法优化负样本会不会带来更大的提升呢？

针对 ctr 预测问题，我们把上一次点击、下一次展现和样本是否点击组成若干个三元组，然后我们去估计给定两个 item_id 的点击率，从而计算出两个 item 的相似。实践证明这种直接优化点击率目标函数的效果是非常好的，但是这种模型在遇到新的文章或者物品时是没法计算历史信息的，这其实是一个泛化性问题，怎么理解呢？

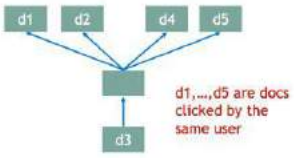
矩阵分解：演化一

- 负样本=展现未点击 ctr预估模型
- 举个最简单的例子：
  - last_click_item_id next_show_item_id1 click
  - last_click_item_id next_show_item_id2 no_click
  - ...
  - 很多很多的样本
  - $P(item_id|item_id1)=\frac{c(last_click_item_id,next_show_item_id1,click)+1}{c(last_click_item_id,next_show_item_id1,no_{click})+lamda}$

我们思考，能不能用一些泛化性的特征来表达 item_id 呢？当然可以。下面 side information 代表多个特征，熟悉 word2vec 的同学知道右边结构就是一个 skip-gram 的结构，给定一个 item_id，求其指定的窗口区间的 item_id，那么如果把 d3 替换成 itemid, keyword, category, author 等泛化特征，再放入模型中，word2vec 实质上变成一个多分类模型。同理，我们也可以在用户侧融入 side information。

矩阵分解：演化二

- Item侧融入side information
- w2v从机器学习角度来看，是多分类模型，无非就是分类规模比较大而已
- 把d3变成多个特征，毫无难度
- 特征出了itemid，keyword，category，author等统统可以纳入其中

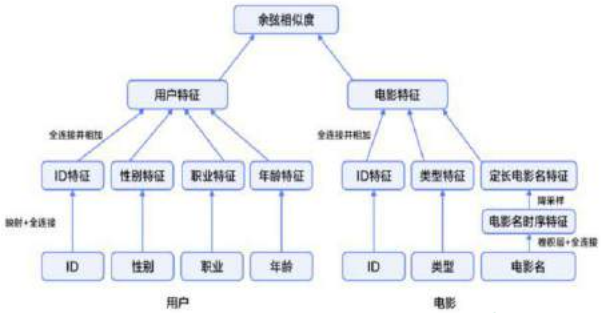


下面右侧图实际上是演化二的形式，只是在左侧用户特征部分也融入了side information，然后用户特征和电影特征两两计算相似，其中一般通过神经网络学习特征之间的交互信息。从整体上看，抽象的来理解演化三上下部分分别对应的是矩阵分解和特征融合两个过程。

上面三种都是基于矩阵分解的演化过程，但是你会发现他们特征之间的分开融合的，也就是用户特征和商品特征分别融合后再计算相似。下面介绍一种特征之间交互更紧密的模型。

矩阵分解：演化三

- 为user和item都插入特征表示的翅膀



FM（Factorization Machines），主要是为了解决数据稀疏的情况下，特征怎样组合的问题。如下其前半部分是我们熟悉的逻辑回归，后半部分就是多个 MF（n 个向量与向量相乘）。无论 MF 还是 FM 最后都是得到特征向量的表式，特征之间先 dot-product 再加和，与特征向量先加和再 dot-product 性质是一样的，只是 FM 在进行特征交叉时二元特征交叉参数不是独立的，而是相互影响的。那么这里读者可以思考一个问题，FM 既然一开始是做 ctr 预估排序模型，那为什么也能做召回？

MF的兄弟--FM

$$\hat{y}(X) := \omega_0 + \sum_{i=1}^{n} \omega_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j$$

$$< v_i, v_j > := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}$$

- FM的后半部分就是多个MF
  - 特征为两类：表达U和表达I
  - U内部特征交互和I内部特征交互对模型效果影响远不如U和I的交互
- 先dot-product，再加和 = 特征向量先加和，再dot-product

虽然“天下武功出一家”，但是每个模型都有它的不足之处，在实际项目中最后的推荐效果都是多个模型组合的效果。最有效也最容易的解决办法就是采用集成手段，从不同的角度生成多个 ctr 预估模型，然后通过 GBDT 去做适合 dense 特征交互，这样集成的方法效果是非常大的。

没有完美的模型—集成

- W&D
- FM
- Item级别信号
- 用户画像

- Score=G(W&D,FM,item,用户画像)

- GBDT比较适合dense特征之间的交互

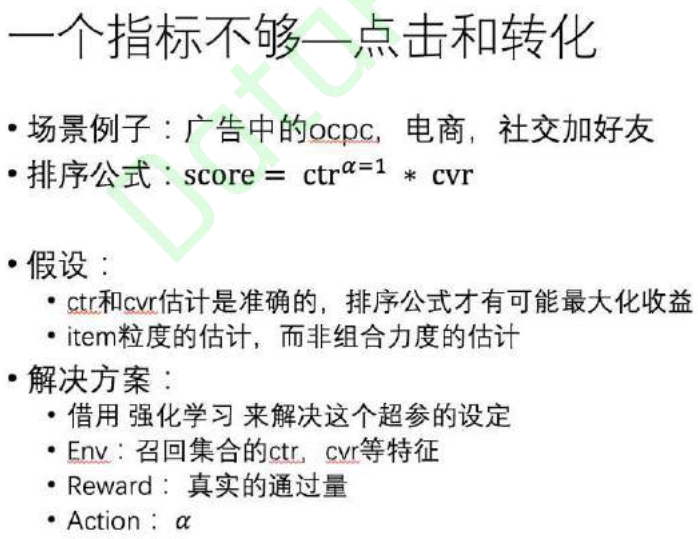
前面涉及到的都是追求的一个业务指标最大化的问题，其实往往在实际业务场景中追求的是多个指标。下面以电商为例，提升点击率不是最核心的，

提升成交转化率才是目的，所以，如何在点击和转化两个目标平衡是非常关键的。我们的解决办法是分别建立 ctr 和 cvr 模型，排序公式：

$$score = ctr^{\alpha=1} * cvr,$$

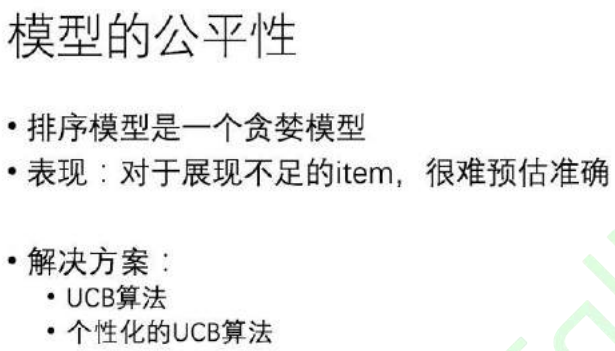
这里前提条件是两个模型预估准确率是非常高的。

另外，其中有个非常重要的超参是借助强化学习来解决的，超参可以理解你要求解的公式中相关的不确定的权重系数，这里不展开强化学习细节，业界强化学习在超参的调试中是比较成熟的解决方案。我们最后花了一周左右的时间调到最佳参数后成交率提升达到 10 几个百分点。



最后，分析一下排序模型，排序模型是一个贪婪模型，它往往是把高 PV 和高点击等前排商品推荐出来，这样能保证推荐效果的置信度。但是那些因为长尾问题而展现不足，质量比较好的内容就不容易被探索出来。业界

现在比较成熟的解决方案就是采用个性化的 UCB 算法，该类型算法能很好的探索出一些被隐藏的高质量的商品。



以上，基本上关于常用的推荐技术做了大致的介绍，最后一点时间，我们谈一下算法之外与业务非常相关的问题。搜索更多是用户主动检索信息，然后通过用户提供的信息推出相关的内容，而推荐通过收集用户更多信息，更加主动推荐相关或相似或者可能感兴趣的内容。但推荐是没法捕捉长尾的内容的，所以没法分发不了一些需求和流量，因为推荐本质上还是“大众投票”的模型。总的来说，“搜索求准，推荐求新”。

推荐技术要在业务场景中落地，其实并不是太难，难就难在如何推出用户真正感兴趣的高质量内容，并且很多工程化上的细枝末节还是需要大家在实践中多积累、多试错、多复盘。

搜索和推荐

	搜索	推荐
表达门槛	高	低
长尾分发	好	差
表达频次	低	高

好，今天跟大家分享的观点，可能都是错的，因为随着技术的进步，上一秒可能正确的做法，下一秒可能就用不到了。

这就是今天所要分享的内容，感谢大家。

附录

为大家推荐些不错的文章，可以自行百度/Google。

❖ 张俊林 《 推荐系统召回四模型之：全能的 FM 模型 》

- Google 《 Deep Neural Networks for YouTube Recommendations 》
- 阿里巴巴 《 强化学习在阿里的技术演进和业务创新 》
- IRGAN
- Lessons from the Netflix Prize Challenge

DataFunTalk

京东电商推荐系统实践

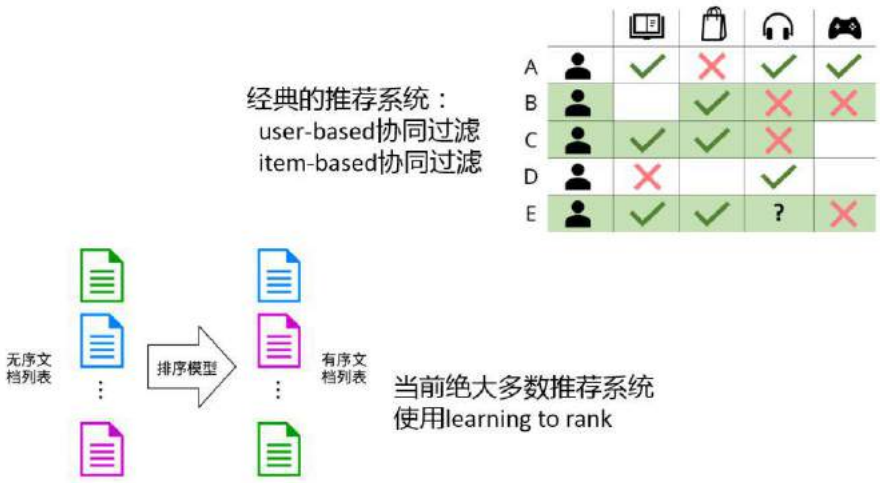
嘉宾：孟崇　编辑：Hoh Xil

今天为大家分享下京东电商推荐系统实践方面的经验，主要包括：

- 简介
- 排序模块
- 实时更新
- 召回和首轮排序
- 实验平台

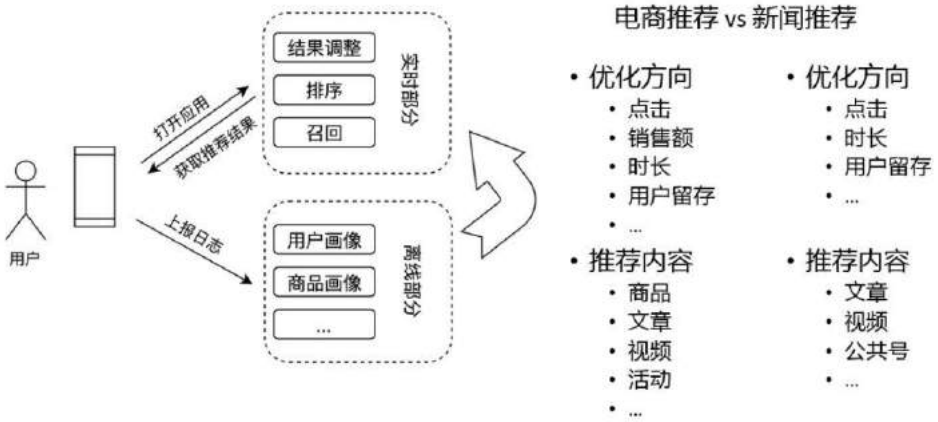
简介

DataFunTalk



说到推荐系统，最经典的就是协同过滤，上图是一个协同过滤的例子。协同过滤主要分为俩种：user-based 基于用户的协同过滤和 item-based 基于商品的协调过滤。

但是，现在绝大多数推荐系统都不会直接使用协同过滤来做推荐。目前主要用的是 learning to rank 框架。



这里，是推荐系统的框架，整个推荐系统可以分为两部分，在线部分和离线部分。

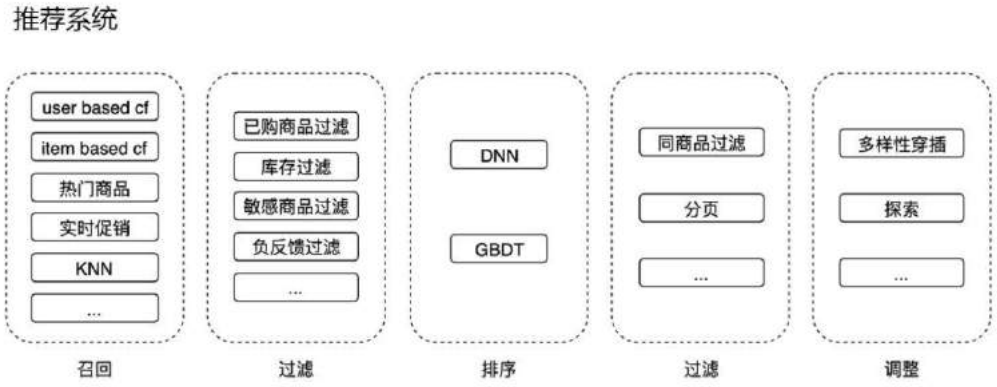
在线部分主要负责当用户访问时，如何把结果拼装好，然后返回给用户。主要模块有召回、排序和对结果的调整。

离线部分主要是对用户日志的数据分析，应用于线上。

整个推荐系统大概就是这样的一个框架。

和新闻、视频这类的内容推荐相比，电商推荐系统又有一些特殊的地方，比如：

优化方向（点击、销售额、时长、用户留存等）。另外，电商中推荐的内容也会有很多种，尤其像是活动类的内容，很多推荐都是算法和人工运营共同完成的。这就是电商推荐和新闻推荐等的区别之处。



我们展开看下在线推荐系统：

除了刚才说的召回和排序以及最终的调整之外，还有实践过程中的一些细节。

召回：这里召回会有很多种方法，如协同过滤，热门商品、实时促销等和应用场景相关的召回，还有一些基于 KNN 的召回。

过滤：召回之后，会进行过滤，主要是和应用场景相关，如已购商品过滤掉、没有库存的过滤掉，或者敏感的商品过滤掉等等这些逻辑。

排序：排序目前主要用到的是 DNN 模型，某些流量比较小的地方会用到 GBDT。

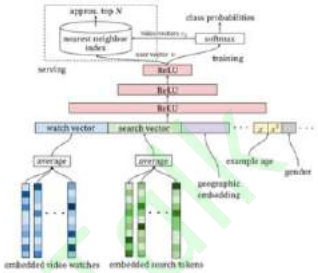
过滤：排序之后还会有些分页、同商品过滤等逻辑。

调整：最终调整过程中，主要有两部分逻辑，多样性和探索逻辑。

排序模块

1. 模型结构

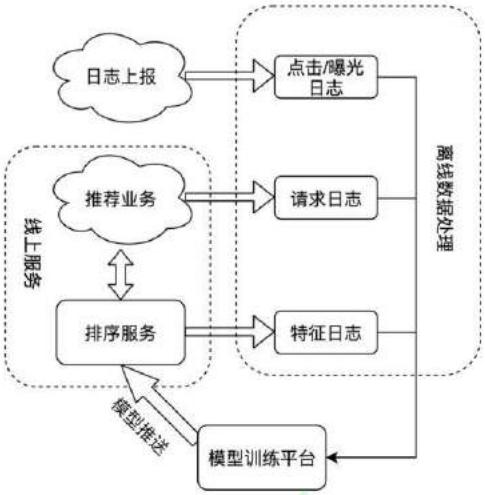
- Ranking models
  - embedding规模很大
  - 训练和上线都比较耗时



深度学习 ranking 模型结构我们不作为重点讨论，这里列举了一种最经典的模型，它们都用到了很多 id 的 Embedding，然后这些 Embedding 规模都很大，这样训练和上线都比较耗时。因此，我们做了一些优化，比如做分布式的训练，并且会有一套 Pipeline 来完成模型的上线。另外，虽然模型很复杂，并且能带来很好的效果，但是特征工程还是必不可少的，很多指标的提升还是依赖于特征工程，当然也包括一些模型调整的工作。

2. 实践

- 单独的排序服务
- 分片计算/并行计算
- 特征记录
- 分布式训练
- 模型更新

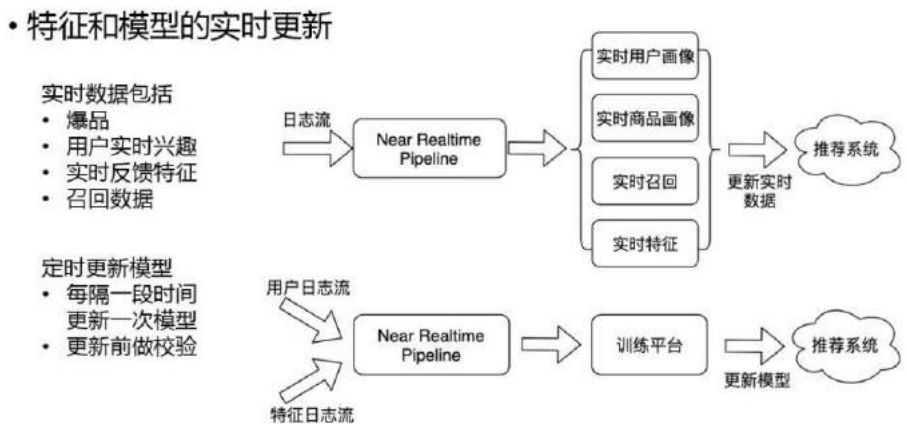


那么如何把这些模型落地呢？我们看下整个模型的上线过程：

首先最重要的部分是模型训练平台和排序服务，因为很多深度模型计算量要求很高，为了能达到比较快的效果，需要部署单独的排序服务。目前比较流行的是 TensorFlow serving，可以很快速的来上线一个深度模型，并充分利用对分片、单机并行，达到很高的计算效率。

模型线上线下一致性问题对于模型效果非常重要，我们使用特征日志来实时记录特征，保证特征的一致性。这样离线处理的时候会把实时的用户反馈，和特征日志做一个结合生成训练样本，然后更新到模型训练平台上，平台更新之后在推送到线上，这样整个排序形成了一个闭环。

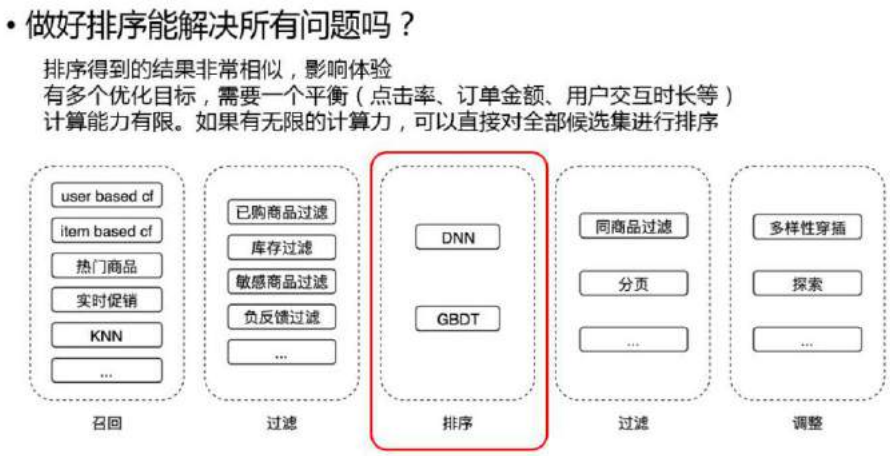
3. 实时更新



我们的特征和模型都需要做实时的更新。因为我们经常需要很快的 catch 一些实时的信号，比如需要实时的用户画像来抓住实时的用户兴趣的变化，还比如需要抓住实时的商品画像，因为经常会有一些活动或者爆品，我们需要快速的捕捉这些信号，并应用到推荐中。另外还有一些实时的召回和特征，比如一些交叉的特征，实时的点击率，实时订单等特征。

除了特征外，模型也需要实时更新，对于电商场景来说这是有一定困难的，因为订单是有延时的，延时可能是十几分钟到十几小时不等，这样实时模型更新上就会采取一些保守的策略，比如用点击率对模型做些微调，然后订单数据再通过离线来获得，这属于业务场景的限制。

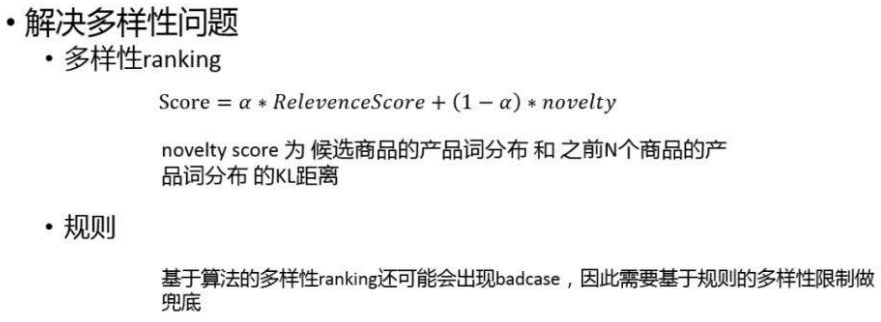
思考



排序可以算是推荐系统中比较重要的一个环节，但是只有排序肯定是不够的，事实上，有一些问题是目前的排序框架无法解决的：

- 排序得到的结果非常相似，影响体验。
- 有多个优化目标，需要一个平衡（点击率、订单金额、用户交互时长等）。
- 计算能力有限，如果有无限的计算力，可以直接对全部候选集进行排序。

1. 多样性



使用模型输出的结果一般都会非常相似，如果直接给用户看体验会很差，因此在模型之后我们需要加入多样性的逻辑。

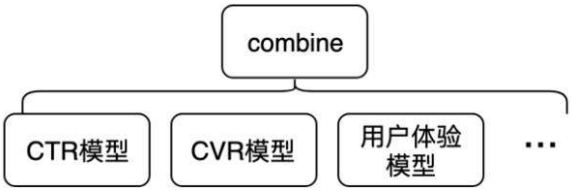
比较通用的解决办法是多样性的 ranking，这是一个贪心算法，从第一个商品开始选，当选第二个商品的时候，会重新计算下候选集中每个商品的 score，然后选择一个 score 最高的。我们的方法是看 novelty score 候选商品的产品词分布和之前 N 个商品的产品词分布的 KL 距离。这样做的思路，就是选一个和已有商品最不像的商品，来更好的保证商品推荐结果的多样性。

由于纯基于算法的多样性可能会出现 badcase，因此还需要一个规则来进行兜底，确保在极端情况下结果也能接受。

最后，我们思考一个问题，有没有更好的方法实现多样性的逻辑呢？当然有，比如是否可以考虑使用 list wise ranking。这里只是为大家分享一个比较容易的，并且效果比较好的方法。

2. 多目标

- 解决多目标优化问题
  - ctr+cvr模型
  - 每个模型优化一个目标

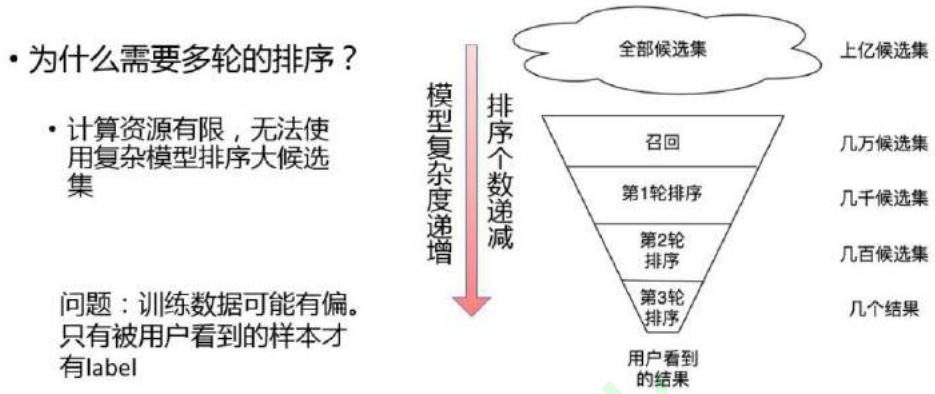


我们的优化目标有很多，比如点击、转化、时长等，问题会变得比较复杂，单一的模型训练很难覆盖到所有指标。另外，经常我们需要在各个指标之间进行权衡，因此可调试性也非常重要。

一种很有用的方式是多模型 ranking，然后用某种方式把所有模型的结果 combine。

这也体现了一个思想，在算法的实际应用中，其实需要在算法的先进性和系统可维护性、可调试性之间做一个平衡。往往 paper 里很有创意的算法落地的时候是有些困难的。

3. 多轮排序



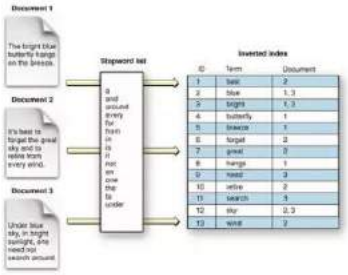
下面我们讨论一下多轮排序的问题。多轮排序是 learning to rank 实践中很重要的一个思想。使用多轮排序主要是因为计算资源的限制，无法使用复杂的模型进行大规模的候选集排序。右图描述了一个多轮排序的框架。这像是一个漏斗模型，从上往下模型的复杂度是递增的，同时候选集是逐渐减少的，就是越到后面用越复杂的模型来保证效果更好，越到前面可能只需要简单的模型来保证能拿到一些商品就可以了。

这样会存在一个问题，由于训练样本可能有偏，导致只有被用户看到的样本才有 label，但是一般不会有太大的影响。

基于索引的首轮排序

1. 索引召回

- 各种召回都可以放在倒排索引里
- 召回完后统一做第1轮排序



用于召回的key	含义
Item cf的key	Item cf召回
产品词	用户画像对产品词召回
热门类目	热门商品召回
促销产品词	促销召回
...	...

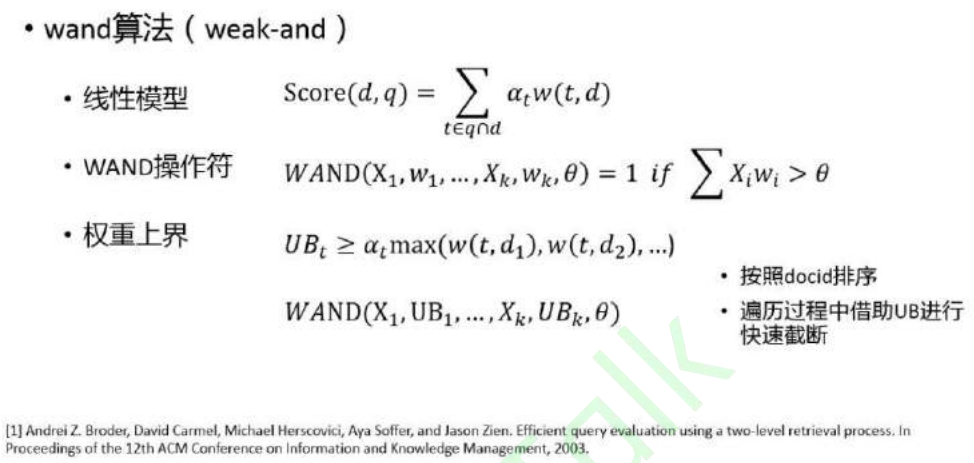
下面我们重点介绍一下第一轮排序。倒排索引很常见，是信息检索里常用的工具。它通过把 doc 的内容索引到 doc id 的方式，快速通过内容来查找 doc。我们很多召回都是通过索引实现的。这里我列举了一些基于索引的召回方式，如 item cf 的 key、产品词、热门类目、促销产品词等。

- 索引召回量依然很大，需要基于quality score进行截断
- 模型选择，LR或GBDT(特征数较少）
- 截断后增加多样性逻辑

$$\text{Score} = w_1 * f_1 + w_2 * f_2 + \cdots$$

虽然索引能够很大程度上的缩小候选集的范围，但是经常情况下，第一轮排序的 doc 数量仍然可能会很大。为了保证性能，截断逻辑是必不可少的。通过情况下可以通过 quality score 截断，保留质量好的 doc。经过线性的 LR

或者 GBDT 模型就可以有结果了。另外截断之后需要有些多样性的逻辑，因为只有在召回的时候保持多样性，最终结果才会有多样性。



[1] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In Proceedings of the 12th ACM Conference on Information and Knowledge Management, 2003.

基于 quality score 截断是一种 naive 的算法，这里我们讨论另一种业界也较常用的算法，wand。wand 其实是 weak and，它的重点是 wand 操作符。wand 操作符是一个布尔操作符，当 $X_i w_i$ 比 θ 大时，它的值是 1，否则是 0。之所以叫做 weak-and，是因为当 w 都取 1， θ 取 K 时，wand 操作符就变成了 and，当 w 取 1，θ 取 1 时，wand 操作符就变成了 or。可以看出 wand 是介于 and 和 or 之间的操作。对 $X_i w_i$ 求和的操作其实和我们线性模型很相似。通过 wand 操作符，我们可以定义一些上界，因为是倒排索引，可以给每个索引链赋予一个估计值，这样就可以拿到权重上界 UB_t，这样通过和 wand 操作符对比，就可以快速的判断 UB_t 是否满足条件，如果满足条件就可以快速的把一些 doc 扔掉，这样就可以快速的使用线性模

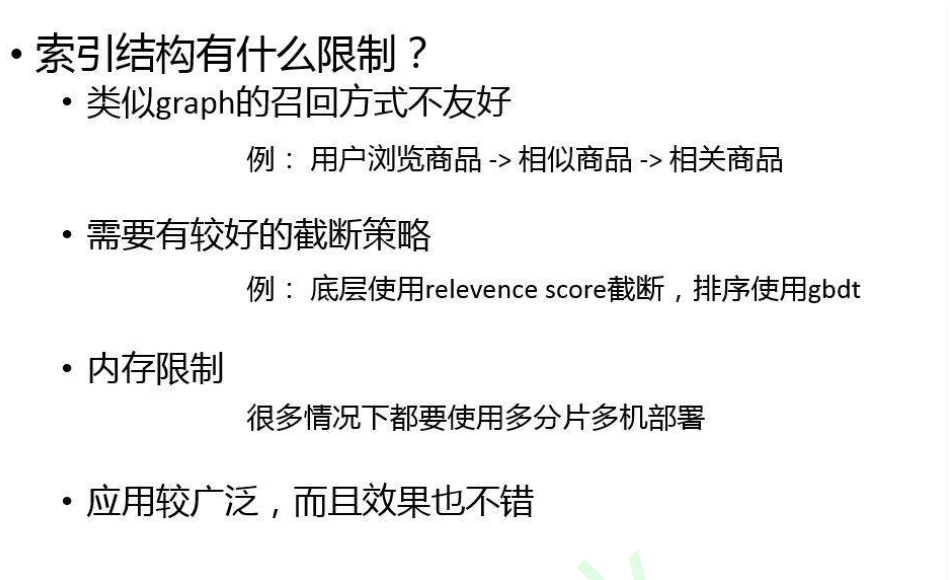
型对全户做 ranking。可以看到，基于线性模型的分数做截断，比完全基于 quality score 截断的策略要稍微好一点。

- wand算法
  - curDoc表示当前遍历到的docid
  - 所有DID≤curDoc的文档已经被检查过
  - 对于一个term t，DID<posting[t].DID的文档已经检查过

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
        DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
        UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
          one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
            to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
            one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

这里我列了 paper 中 wand 算法的伪代码。出于时间关系，我们不会过算法逻辑的细节。我认为它的主要的思路是通过快速使用 upper bound 做截断和跳转，可以略过很多明显不符合的候选 doc，从而减少计算 score 的次数。当然这种方法对于线性模型来说，有一个缺点，当我们需要多样性的时候，没办法很好的实现在模型中增加多样性的。

wand 算法目前已经应用非常广泛了，在很多开源的索引如 lucene 中，也会用到这种方法快速计算文本相关分。



刚刚我们介绍了使用倒排索引做第一轮排序，以及一个常见的排序加速算法，回过来我们思考一下倒排索引本身，它适用于什么场景，不适用于什么场景。

首先它适用于 kv 查找这种场景，并且 kv 查找也属于实际应用很多的情况。但是对于更复杂的方式，类似 graph 的召回方式不友好，比如找用户看过的商品中相似商品的相关商品，这时实现起来会比较麻烦，这是它的一些限制。再一个，我们需要有较好的截断策略，例如底层使用 relevence score 截断，排序使用 GBDT。

当然，索引还会受到机器本身的内存限制，限于机器的大小，很多时候我们需要多机分片部署索引，这样会带来一定的复杂性。虽然有些限制，但是索引是目前应用很广泛、有效的方式，包括在推荐、搜索等领域都会使用到。

2. KNN 召回

- K Nearest Neighbors

1. 如何产生embedding

2. 定义embedding之间的距离

用户embedding： $X = [x_1, x_2, \dots, x_n]$

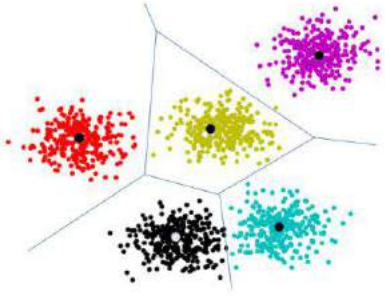
候选商品embedding： $C = [c_1, c_2, \dots, c_n]$

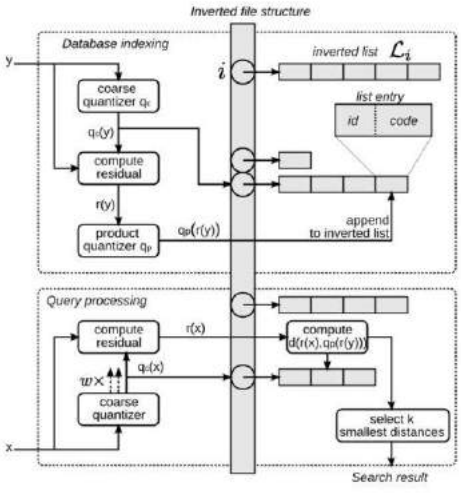
使用欧氏距离： $d = \|X - C\|_2$

3. 找到用户embedding向量的K近领

除了索引召回，KNN 也是现在较常用的一种召回方式。首先，我们把所有的候选集转换成 embedding，我们把用户兴趣也可以转换成 embedding，通过定义 embedding 之间距离计算公式，我们可以定义 KNN 召回问题，也就是在全部候选池中，找到与用户最接近的 k 个结果。

- KNN近似算法





[1] Johnson, J., Douze, M., J'egou, H.: Billion-scale similarity search with GPUs. arXiv preprint arXiv:1702.08734

定义好 KNN 召回的问题，下一步就是如何找到最近的 K 个候选集。由于整个候选集非常大，每次都使用用户的 embedding 去全量计算距离是不现实的，只能使用一种近似算法。我们今天分享其中的一种近似算法。是 facebook 开源的 KNN 计算库 faiss 使用的。其原理：

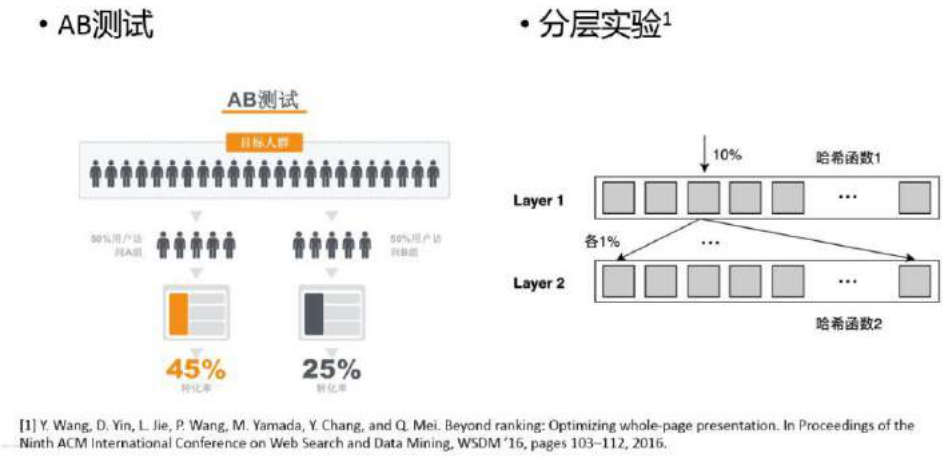
首先需要对全部候选集进行分块，每一块都会有自己的质心。paper 中使用 Lloyd 算法，将整个空间划分开。分块后，就需要对每一块构建索引，进而通过索引实现快速检索的功能。

右图是索引构建和检索的方法。

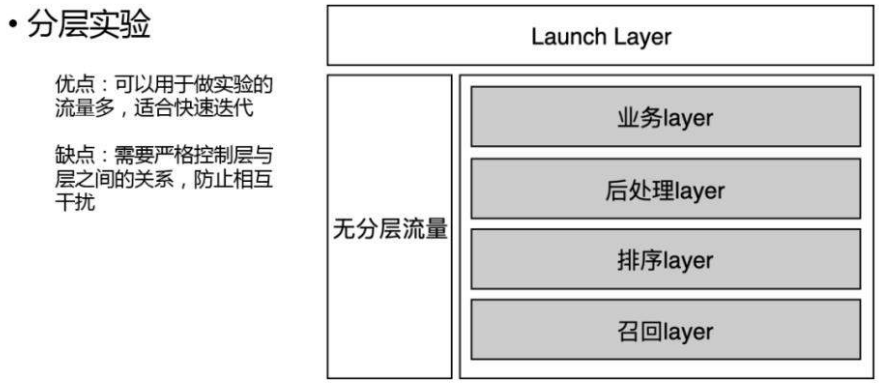
上半部分是如何构建索引(这里的优化点是使用了二级索引):首先拿到 y 候选集之后，做一个 quantizer 分类得到一个一级索引，把它放到索引表中，另外还得到残差 compute residual，可以对残差再进行一次 quantizer，得到一个二级索引，通过两级索引来加快检索的速度，同理，在真正的 quary 的时候，拿到的是用户的向量 x，先做一个 quantizer，得到 k 近邻的一级索引，然后查找 k 个一级索引，同时拿到 k 个二级索引，然后在二级索引中查找，然后这里还有很多加速的算法（这里就不展开了），通过这样一种多层的查询方式来做到加速 K 近邻的算法。

PS：关于 KNN 的一些思考，KNN 是一种有效的方式，但是不是唯一有效的方式。比如之后分享的 TDM，能够比 KNN 更加灵活。

实验平台



最后简单介绍下分层实验平台，因为大家想快速迭代特征和模型，离不开实验，经常会遇到的情况是实验流量不够用了，这时就需要对实验做分层。分层的逻辑见右图，通过在不同的 Layer 使用不同的哈希函数，保证每个 Layer 之间流量是正交的，这样就可以在不同的 Layer 上做不同的实验。



分层实验的具体做法：召回->排序->后处理->业务，另外还有一部分对齐流量，用来做全量的验证。

分层的优点，可以用于做实验的流量多，适合快速迭代；缺点，需要严格控制层与层之间的关系，防止相互干扰。本次分享就到这里，谢谢大家。

阿里妈妈深度树检索技术（TDM）及应用框架的探索实践

嘉宾：何杰　编辑：孙锴

导读：阿里妈妈是阿里巴巴集团旗下数字营销的大中台，2018 年广告营收超过 1500 亿，近乎占据中国广告市场收入的半壁江山。如何驱动这艘商业航母不断前行，阿里妈妈技术团队始终坚持技术创新驱动业务增长的战略，而 TDM 正是在这一战略指导下，由阿里妈妈精准定向广告算法团队自主研究、设计、应用从而创造巨大商业价值的创新算法典型代表。

今天的主要内容分为四个部分：

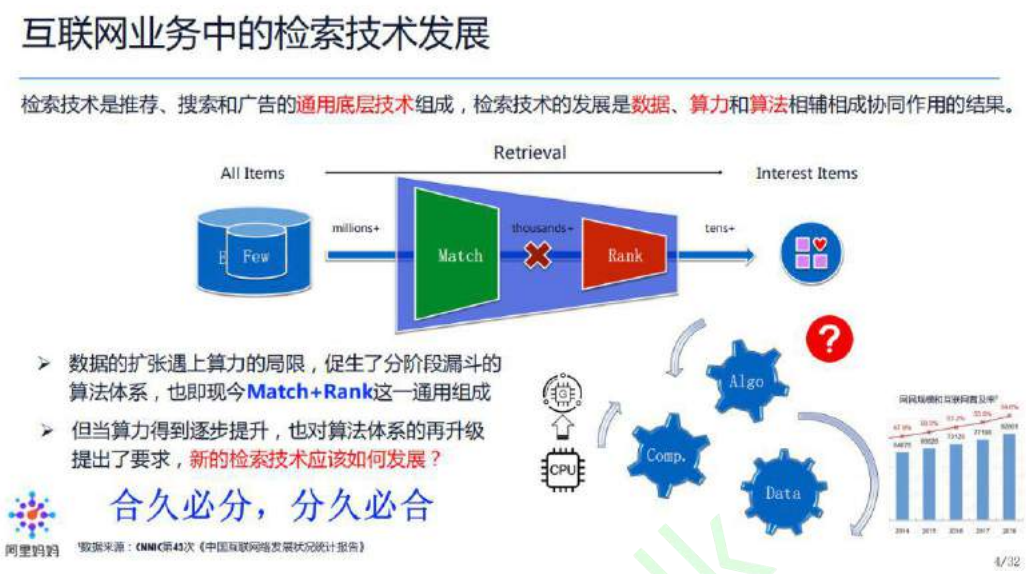
1. 从互联网推荐业务看检索技术发展现状

2. 深度树检索（TDM）的设计原理和具体实现

3. 深度树检索技术对于线上业务场景的适配应用

4. 我们对下一代检索技术未来发展的思考

从互联网推荐业务看检索技术发展现状



检索技术，是互联网推荐、搜索、广告的通用底层技术组成。

一提到检索技术，大家首先会想到搜索引擎，但是从另外一个视角来看，搜索、推荐、广告实际上是一脉相承的，他们的检索可以**统一定义为：从全量的物品里面，挑选出来用户感兴趣的物品。**不同的是，搜索中 Query 词显式表达了用户意图，而在推荐中往往利用的是用户隐式表达如历史行为、用户属性等，而广告则是在搜索/推荐上进一步施加了广告出价的影响。

我们认为检索技术的发展是数据、算力、算法相辅相成，协同作用的结果，具体表现为：

最初，数据少所以候选 Items 规模比较小，用很小的算力和简单的算法，就可以一步到位（如全量过一遍候选）计算出来用户感兴趣的 Items 是什么；

然而，数据在不断膨胀，尤其是随着近些年移动互联网的发展和移动设备的普及，使得我们的数据开始呈现爆炸式的增长，从而导致我们的候选 Items 规模也大幅增大，一步到位或者说原来的检索方式已经无法支撑这种量级的数据。

怎么办？在如此大规模数据下，算力和算法如何适应性设计呢？我们首先来看一下算力的发展，在十几年前我们的算力主要还是以 CPU 通用计算为主的算力体系，算力相对比较有限。大数据+有限算力，就要求我们对算法进行适应性设计，于是 Match+Rank 分段漏斗的算法架构被引入：从超大规模候选集中，先 Match 出小规模候选集，再通过 Rank 的方式排序出最终用户最感兴趣的 Items。逐渐地，Match+Rank 的模式成为了如今检索体系的主流架构。

数据在发展，算力体系也在发展，尤其是随着近十年来 GPU 大规模应用的兴起，形成了 CPU+GPU 异构计算的算力大融合。在大数据+大算力的现状下，我们开始思考，检索算法架构是不是可以再升级？类似于历史中的“合久必分，分久必合”：我们是不是可以突破 Match+Rank 的分段漏斗模式，将检索问题作为一个端到端整体去设计和联合优化？这就是我们今天要探讨和思考的问题，也是 TDM 的由来。

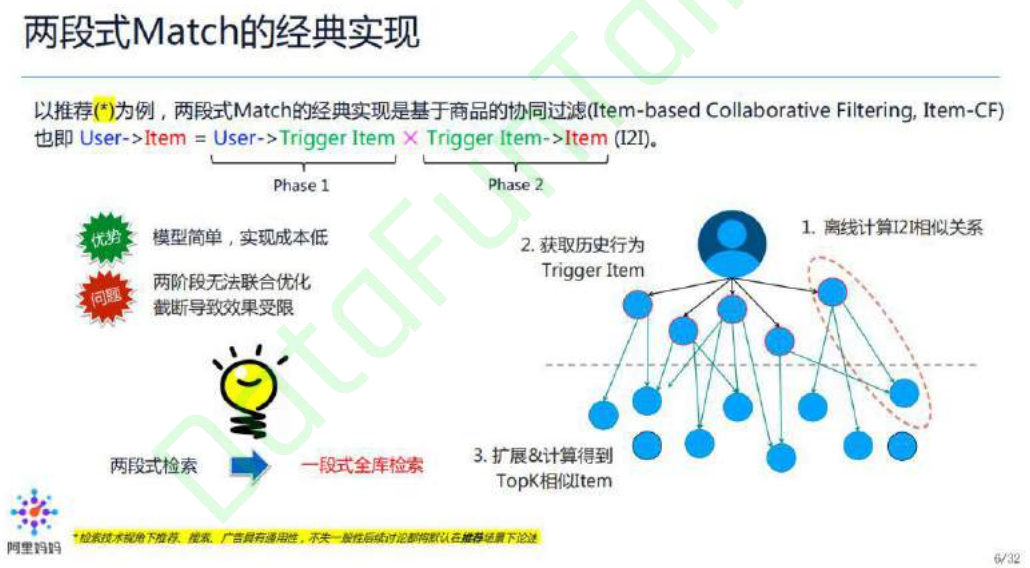
我们怎么来思考检索架构升级呢？首先需要对检索本身进行深入分析，而作为检索的第一阶段，我们先从 Match 入手。Match 的核心任务是从全量的超大规模的候选集中高效的检索相关 TopK，然后交给后链路比如 Rank 做进一步处理。比如在电商场景，我们有大规模商品库，大概是十亿量级，我们会做 Match 召回千到万级别的 TopK，送给后链路做 Rank。

类似于受算力局限下检索 Match+Rank 两阶段的设计，Match 本身也受到算力局限的影响，所以它的经典实现也是两段式的，比如：

搜索场景下，两段式的表现，就是 User 到 Query-rewrite，Query-rewrite 到 Doc；

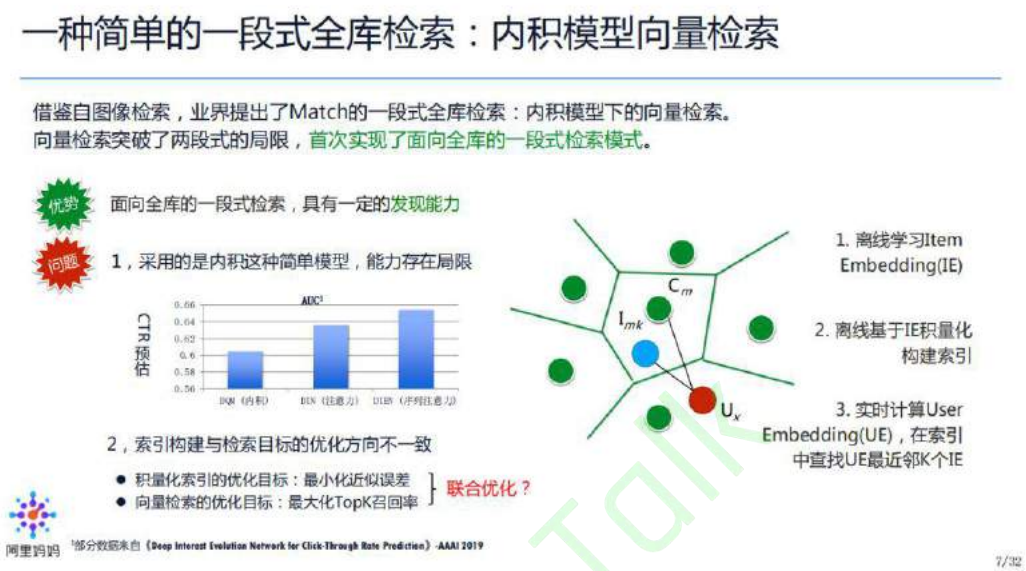
推荐场景下，两段式的表现，就是 User 到 Interest-tag（如品牌），Interest-tag 到 Item。

这里存在的问题是两段式相互隔离，比如我们对 User 到 Query-rewrite 进行优化形成各种匹配算法，然后对 Query-rewrite 到 Doc 优化形成倒排索引。这种两段式缺少联合优化，并且受限于线上性能影响，两段式的各个阶段的标签数量是有截断的，效果（如全量发现能力）上就存在局限。



Match 两段式的经典实现如图中 Item-CF 所示，这里不做过多论述。如前述说的，它的优势是模型比较简单，实现成本较低，劣势是两段式无法联合优化，各阶段的标签截断导致效果受限。由此，我们自然会想到一个问题：两段式有缺陷，那我们是否可以改为一段式？

答案是肯定的，那么一段式要怎么做？



在推荐和相关领域，简单的一段式全库检索的经典代表，其实借鉴了图像检索的算法，即：内积模型的向量检索。它通过各种方式学到了 Item 的 Embedding，然后通过 PQ 方式构建分库索引，检索时实时计算 User Embedding，做最近邻 TopK Item 检索。内积模型向量检索在 Facebook 17 年开源了 FAISS 库后，得到了广泛的应用。这种一段式全库检索，在发现能力上具有一定优势，但缺点是模型比较简单，能力有限。这里以 PCTR 预估为例，内积模式的 DQM、Attention 的 DIN、Attention+GRU 的 DIEN 做比较来看，内积模式的 AUC 是最低的，间接证明了内积模型是存在能力局限的。

另外，这种模式下索引构建和模型优化的目标存在不一致。积量化索引的优化目标是最小化近似误差，而向量检索（召回问题）的优化目标是最大化

TopK 召回率。所以由于两者优化的目标不一致，这会导致最终效果并不是最优的。

我们总结一下 Match 中检索技术的发展历程，不难发现，**检索技术是模型能力和索引效能的融合：**

基于商品的协同过滤（Item-CF），模型能力：启发式统计规则，没有学习成分；索引效能：分离两段式；问题：非学习模型，非面向全库索引；

内积模型向量检索，模型能力：内积模型；索引效能：全库一段式；问题：内积模型相对简单（问题：为什么是内积而不能是其他更复杂的模型如DNN？留给读者思考）；索引和目标之间没有联合优化。

那么面对这样的现状，下一代模型，我们要如何发展？答案是：更先进的模型、更先进（高效）的索引。

深度树检索（TDM）的设计原理和具体实现

应用先进模型，我们自然想到了深度学习模型，但它的引入会带来额外的问题：

① 深度学习的单点计算消耗 T 较大，在性能有界的情况下，我们的计算次数 N 不能过大；

② 要实现面向全库的检索，N 越大越好（最大时即为候选集 size），在性能有界的情况下，与上述 ① 产生了矛盾；

如何解决这一矛盾？还是“**检索技术=模型能力+索引效能**”，也即模型的升级需要索引结构的升级来适配，于是我们的核心任务变为：建立一个高效的索引结构来承载先进的模型能力。高效索引有很多，散列表、图等都有一些弊端，最终我们选择了树这一层次结构（如图所示）。

以十亿商品为例我们可以构建一棵 30 层的树，检索 Top1 我们只需要计算 30 次。

基于以上这些考量，我们提出了 Tree-based Deep Match（TDM）这个算法，具体我们需要进一步解决以下四个问题：

① 如何基于树实现高效检索？

② 如何做兴趣建模保证树检索有效性？

③ 如何学习兴趣模型？

④ 如何构建和优化树索引结构？

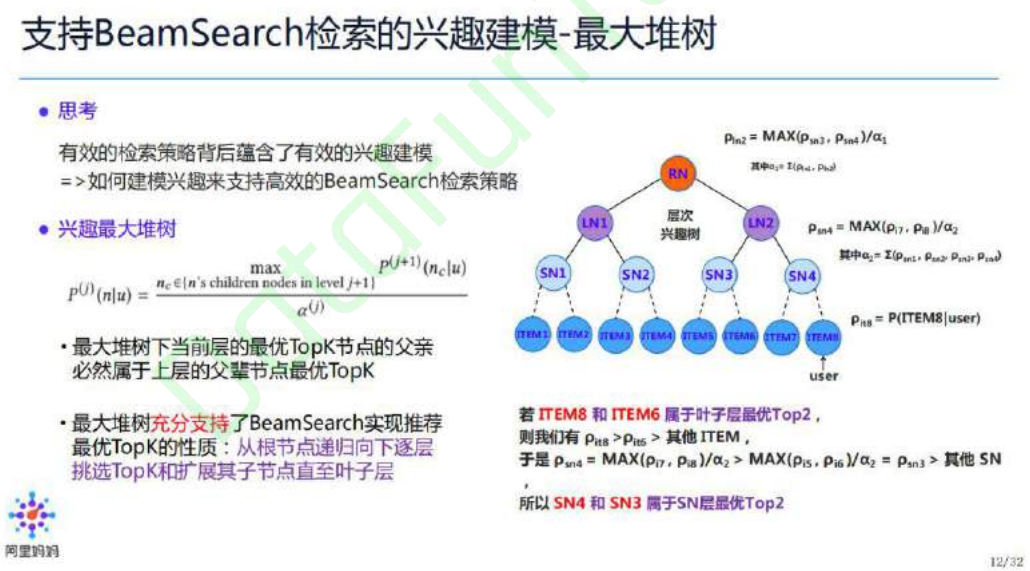
接下来，我们会以淘宝的商品推荐为例，做一个关于 TDM 设计和实现的详细介绍，来帮助大家理解。

如何做高效检索？不失一般性，假定我们的树是一棵完全二叉树，全部叶子节点代表全部商品，中间节点代表某种意义上的粗粒度聚合（例如“iPhone”是商品，作为叶子，“手机”是该商品的类别，作为父节点；但是这里需要说明，在通常的 TDM 树中，我们并不要求中间节点一定存在具体物理意义）。

树上如何做高效检索呢？我们采用了 BeamSearch 这种方式，自顶向下，做快速剪枝。

BeamSearch 检索方式如图中右侧流程所示：假设要寻找最优 Top2 个 Item，第一层有两个子节点，打分排序选出 Top2（只有 2 个，全取）；扩展出第二层四个子节点，打分排序选出 Top2；扩展出第三层四个子节点，打分排序选出 Top2，达到叶子层，最终返回 Top2 个 Items，查找复杂度为 O(2*K*log2N)，K 为返回个数，N 为叶子个数。

衍生问题：为什么该检索策略检索出来的 Top2，一定会是用户感兴趣的 Top2 呢？即 BeamSearch 的有效性要如何保证？



有效的检索背后蕴藏了有效的兴趣建模，也即，要如何建模，使得 BeamSearch 检索有效。

为此我们提出了兴趣最大堆树的概念：用户对 n 节点兴趣的偏好，是用户对 n 节点的孩子节点们的兴趣偏好 ,取 max 后 ,在 n 所在的节点层做归一化。如图中右侧树所示，SN4 的兴趣偏好：

$$P_{SN4}=normalization(Layer_{SN}, max(P_{Item7}, P_{Item8}))$$

在这样的假设下，有个很好的性质：最大堆树下，当前层最优 TopK 节点的父亲，一定属于上一层的最优 TopK。

以右图举例：如果 Item8 和 Item6 是叶子层最优 Top2，那么根据我们之前的描述，SN4 和 SN3 是 SN 层的最优 Top2。

因此，最大堆树的定义是 BeamSearch 有效的充分条件。所以我们可以从根节点递归向下，逐层挑选 TopK，然后扩展至叶子层。

我们如何学习最大堆树的模型？BeamSearch 检索本质上要求具备对每一层进行 TopK 排序的能力。我们的做法是：构建符合这样性质的样本，让样本牵引模型学习，去逼近最大堆。

具体思路：

叶子层的节点兴趣比较容易构建，因为用户对叶子层的节点是有行为产生的，有行为/没有行为对应感兴趣/不感兴趣，以此来构建序标签；中间节点，用最

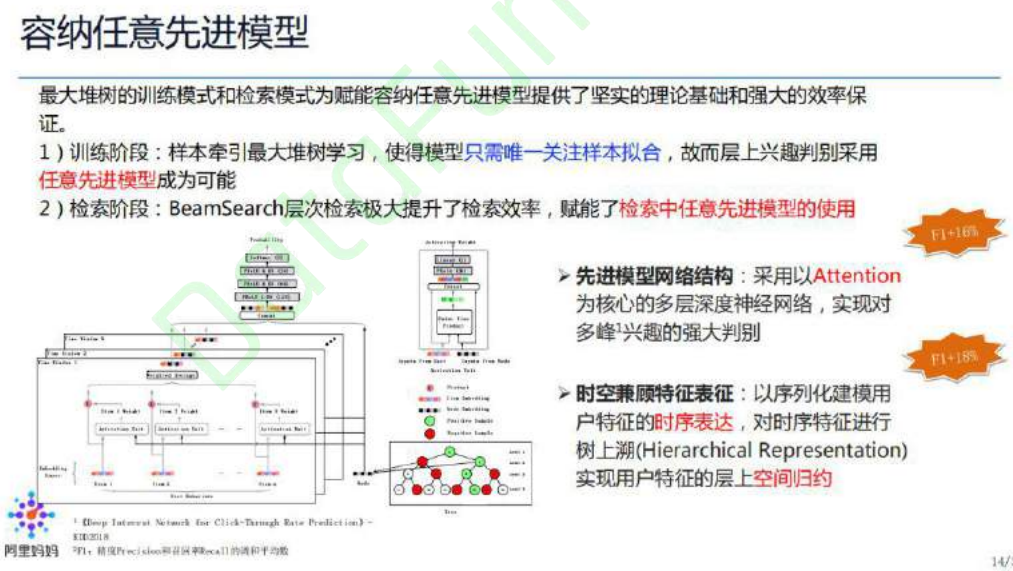
大堆递归上述的方式去推导每一层的序标签；当我们有了每一层的序标签，就可以用深度学习去拟合序标签的样本

在实际操作中，我们没有用序样本去构建序模型，而是把序估计还原成点估计，此处可以理解为把排序模型转换成分类模型。

有了分类样本之后，就可以让模型去拟合，牵引模型学习逼近最大堆性质。

采样方案：

我们认为绿色代表正样本 ,红色代表负样本 ,如果用户对 Item6 节点有访问，那我们认为用户对 Item6 节点是一个正兴趣 ,我们在当前的叶子层做随机的负采样，那么上一层怎么做呢？正样本上溯路径的祖先仍然为正样本，然后每一层再做负采样，所以这样就固定出来每一层的正负样本，这样整个树的样本都已经固定完毕。



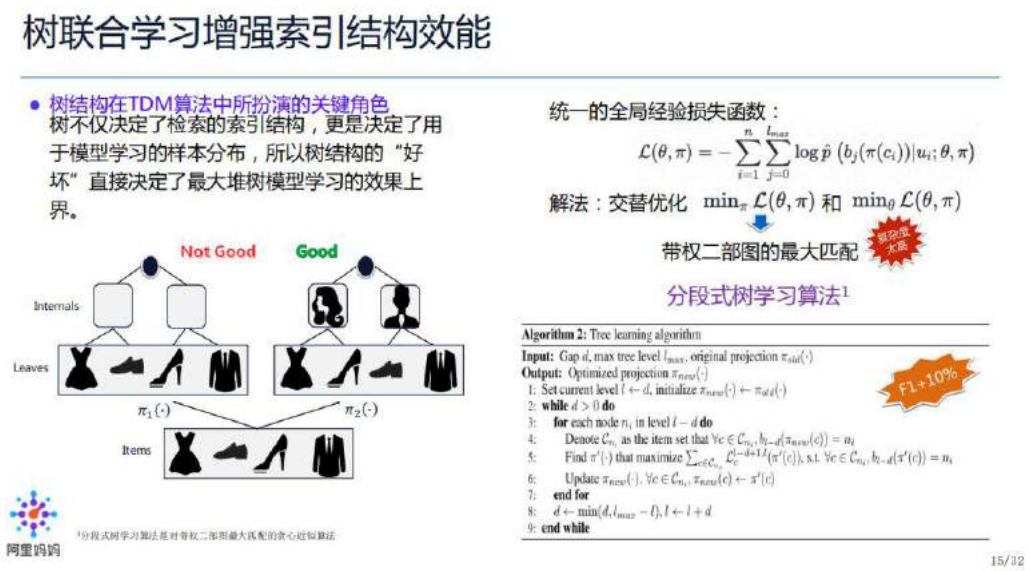
有了正负样本，我们就可以用任意复杂模型去拟合 label。可以看到这样模式的优点：

我们刚刚一直在提要用先进模型或者深度模型，这里发现最大堆树的训练模式和检索阶段的效率保证了可以基于在线检索的性能约束（上界）来决定可以用多复杂的模型达到最高的检索效果。

上图为具体的模型结构，可以看到：

先进模型网络结构：采用以 Attention 为核心的多层深度神经网络，实现对多峰兴趣的强大判别，F1(Precision 和 Recall 的调和平均数)提升了 16%。

时空兼顾特征表征：以序列化建模用户特征的时序表达，对时序特征进行树上溯（Hierarchical Representation）实现用户特征的层上空间归约，F1 又提升了 18%。



树结构在 TDM 算法中所扮演的关键角色：树不仅决定了检索的索引结构，更是决定了用于模型学习的样本分布，所以树结构的“好坏”直接决定了最大堆树模型学习的效果上界。

不失一般性，假定我们的树仍为完全二叉树，叶子节点为 n 个，那树的本质其实就是如何将 n 个商品一一投射到 n 个叶子上去，也即树的形式化定义是找个一个 π 映射实现 n 对 n 的二部完美映射。那怎么样的树是好的树呢？也即何种 π 映射是最优的呢？从左下图可以看出来，右侧的树结构更优，因为它隐含了性别分类，可以使得 TDM 模型学习得更好（上限更高）。

为了达到模型和树的联合最优化，我们建立了统一的全局经验损失函数（如右上公式），并通过交替迭代的方式进行模型参数 θ 和映射函数 π 的联合优化。其中 π 的优化是一个带权二部图的最大匹配问题，复杂度比较高（O(n^3)），为此我们建立了右下的贪心近似算法（Algorithm2）来优化 π。最终的效果是这种联合优化让 F1 提升近 10%。

PS：这里有现场同学提问

Q：前几页的最大堆树学习里有正负采样，为什么“统一的全局经验损失函数”里面没有负样本项了？

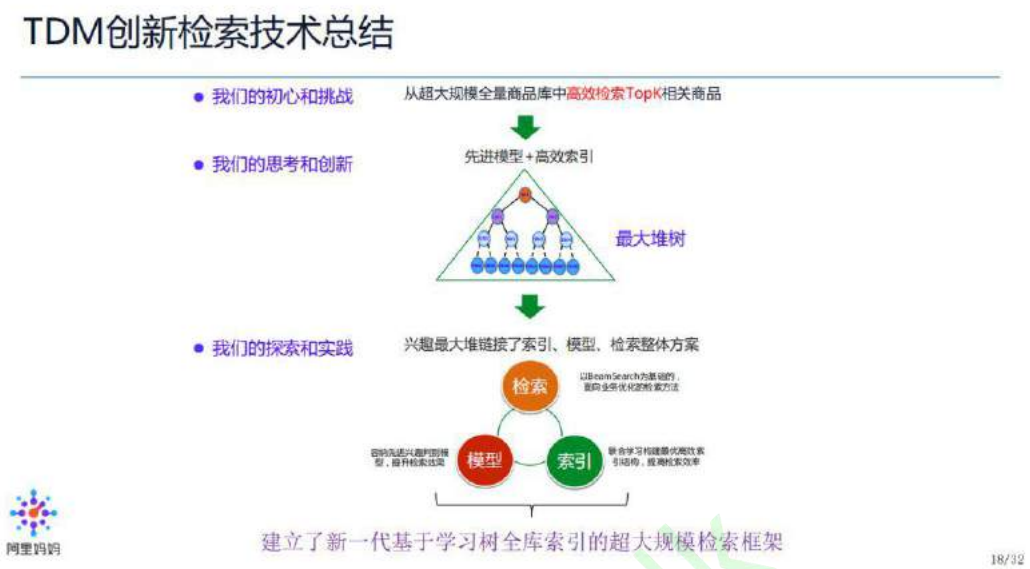
A：为了实现统一 Loss，我们这里对 TDM 模型进行了建模变换，从之前的二分类转换到了多分类，即将层上用户兴趣建模成用户对当前层上的全部树节点做多分类问题。上述统一 Loss 即为多分类建模下的标准交叉熵 Loss，其中的 p 为归一化概率。在实际实现方式上，我们采用了负采样 + NCE 的方法来近似多分类 Softmax。所以上述统一 Loss 里面确实没有了负样本项，“负样本”是以 NCE 方法中的负采样方式表现了。

DataFunTalk

TDM 在 AmazonBooks 和 TaobaoUserBehavior 两个公开数据集上进行了测试。相对于基线的 ItemCF 和 Youtube 内积模型，TDM 的提升非常显著。我们也对 TDM 各种优化做了单项拆分分析，Attention、HierarchicalRep. 和联合学习的效果如之前所说，其中三者的联合效果达到了 1+1+1>3。

回到最初的图中，我们再次发现：检索技术是模型能力和索引效能的融合。

从这个角度，大家思考一下，TDM 为什么可以解决这个融合呢？我们实现了对更先进模型，即深度学习模型的融合；其次，我们实现了更先进索引，采用了树结构来做索引，有别于常用的倒排索引，并且我们的树结构是一个与模型适配且联合学习的树。



总结一下 TDM 提出、设计和实现的脉络：

目标：解决从超大规模全量商品库中高效检索 TopK 商品的问题

思考：希望由先进模型带来性能/效果的提升；为赋能先进模型，需要高效索引；由此产生最大堆树的理论建模

探索：基于最大堆树，我们链接了 BeamSearch 检索下深度学习模型和高效树索引联合学习的方式，形成了基于学习树的全库索引的检索框架。

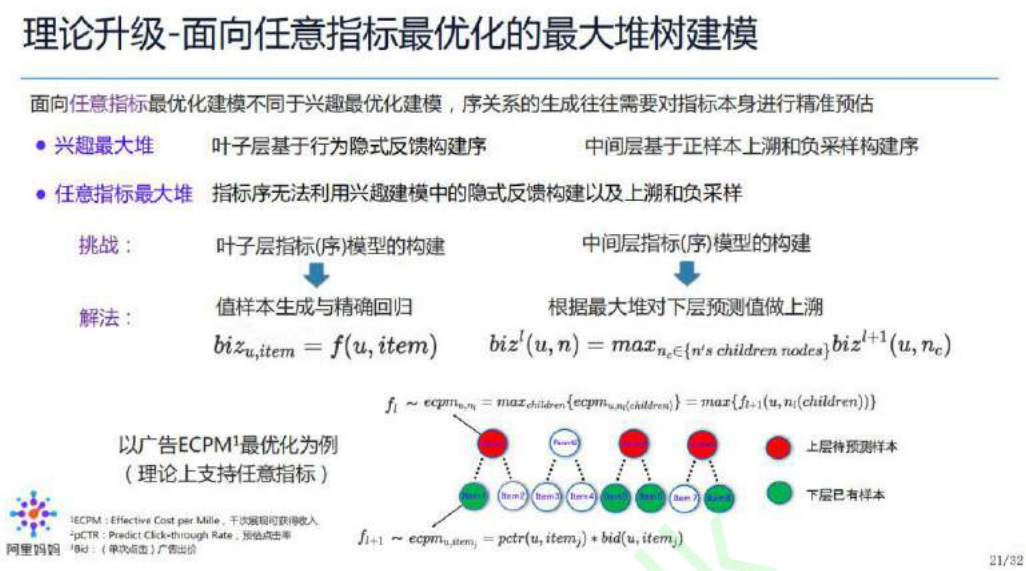
深度树检索技术对于线上业务场景的适配应用

原生 TDM 是兴趣最优的检索方案，但是在实际业务中，有些业务目标不一定是兴趣最优。例如在广告业务中，我们要考虑 ECPM 最优。那么，我们要如何改造 TDM，使得其可以满足不同的业务目标？

这里面有如下挑战：

算法要从理论上进行升级适配，使得最大堆树的建模适配业务指标；检索也需要改造，在整体链路的设计上要考虑多目标融合，使得检索体系可以支持多个业务目标

此外，克服性能瓶颈，实现检索功能的真正在线应用，也需要一段路要走。

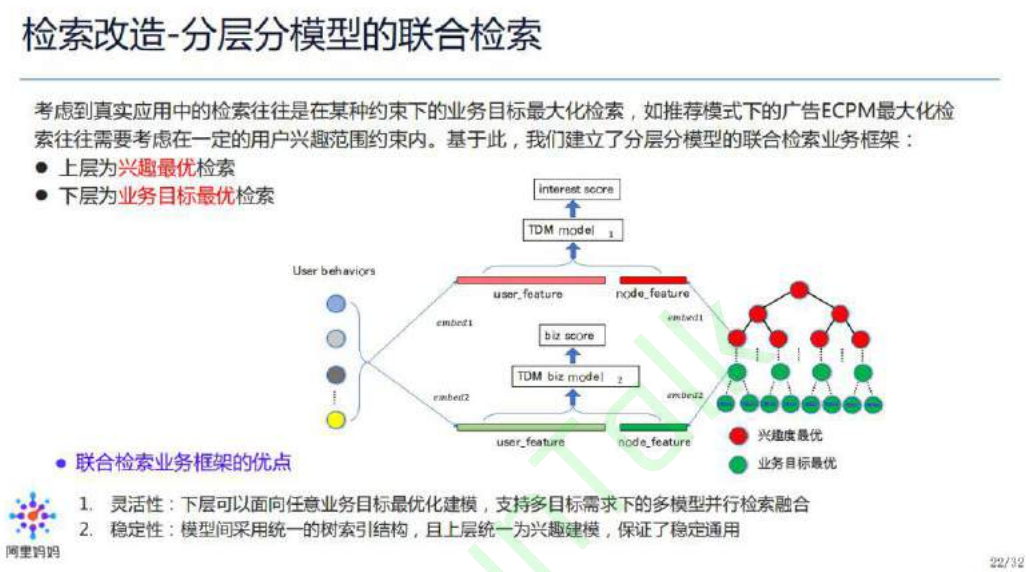


“兴趣最大堆”改造成“任意指标最大堆”的过程中，也存在一些问题，比如：“兴趣”是二值问题，即感兴趣/不感兴趣，可以抽象成分类问题；而在广告中，“ECPM”是一个连续问题，即 ECPM 是一个连续的数字，更适合抽象成回归问题。基于这个现状，我们需要思考，如何构建这样的 TDM 模型。

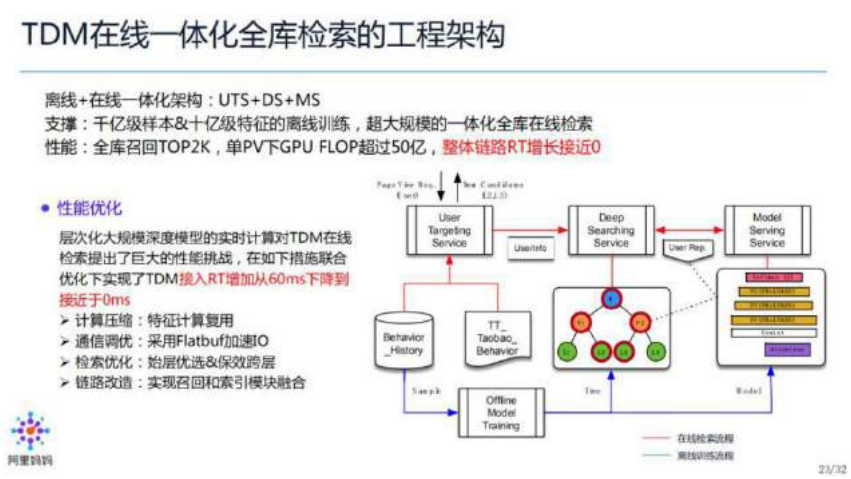
仍然以广告为例：由于叶子层是每一个具体的 Item，那么这些 Item 的 ECPM 是可以计算的（由于 ECPM=PCTR*BID*1000，我们可以通过其他的模型计算 Item 的 PCTR，BID 通过查找/统计的方式可以得到），所以我们有叶子层的所有 Item 的 ECPM 值。所以我们可以通过 max 上溯的方式，将其引入到我们的最大堆树中来，也即：

$$P_{Parent1} = \max(P_{Item1}, P_{Item2})$$

上一层的节点值为该节点的孩子节点的最大值，此处不需要归一化。并以此方式递归上溯生成整棵树，并在检索的时候递归往下完成 TopK 检索。

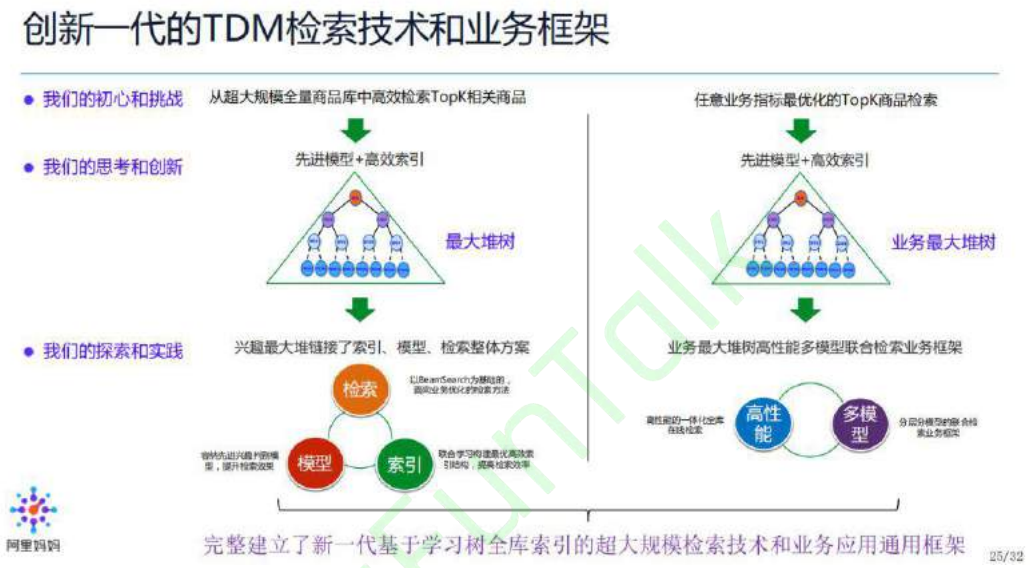


由于一个模型并不能适配所有的场景，满足所有目标（例如：广告业务考虑广告收入的最大化；电商业务考虑电商成交最大化等），因此，我们对检索模型做了另外一个改造，也即：分模型联合检索框架，上层是公共部分，对通用的功能做抽象，下层是业务部分，根据业务现状做设计，两部分实现并行调用。



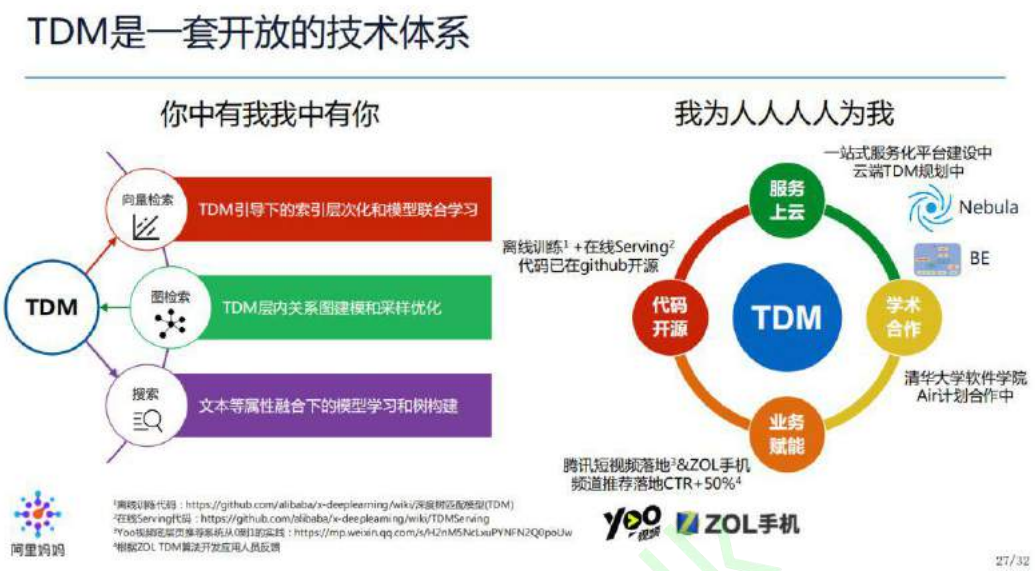
虽然 TDM 理论上可以支持任意深度模型，但实际应用中，我们仍然需要考虑在线检索 Bound 的约束。在一开始上线之初，TDM 的 RT 增量达到了 60ms，对于阿里妈妈的在线广告链路完全不可用。但通过后续的包括通信、计算、链路等优化，最终 TDM 实现了上线后 RT 增加接近于 0ms 的良好性能。

现阶段 TDM 主要接入了阿里妈妈定向广告的 Match 阶段，包括 Shop/Node/Item 已经全量上线，覆盖了阿里妈妈定向广告主要场景的大部分流量，CTR 和 RPM 的效果提升都达到了两位数。



最后，从学习树全库索引的检索，我们进一步把 TDM 扩展到了通用的业务应用框架。左边纵向为基于学习树全库索引的超大规模检索是如何设计的，右边纵向为如何应用 TDM 到线上业务，包括改造成业务最大堆树，然后面向业务的高性能优化和多层多模型的设计，最后真正应用于实际。

我们对下一代检索技术未来发展的思考



对于未来，我们希望将创新推向一个新的高度，当然这只靠我们一个团队是很难做到的。结合当前 TDM 在阿里妈妈的应用，我们希望能在下一个阶段，可以将 TDM 与现有技术做更深层的融合，比如在图检索、搜索业务等领域上面做一些优化和探索。

我们也一直坚持开放、开源的态度。TDM 的离线训练、在线预测的代码已经在 github 开源：

离线训练：

https://github.com/alibaba/x-deeplearning/wiki/ 深 度 树 匹 配 模 型 (TDM)

在线 Serving：

https://github.com/alibaba/x-deeplearning/wiki/TDMServing

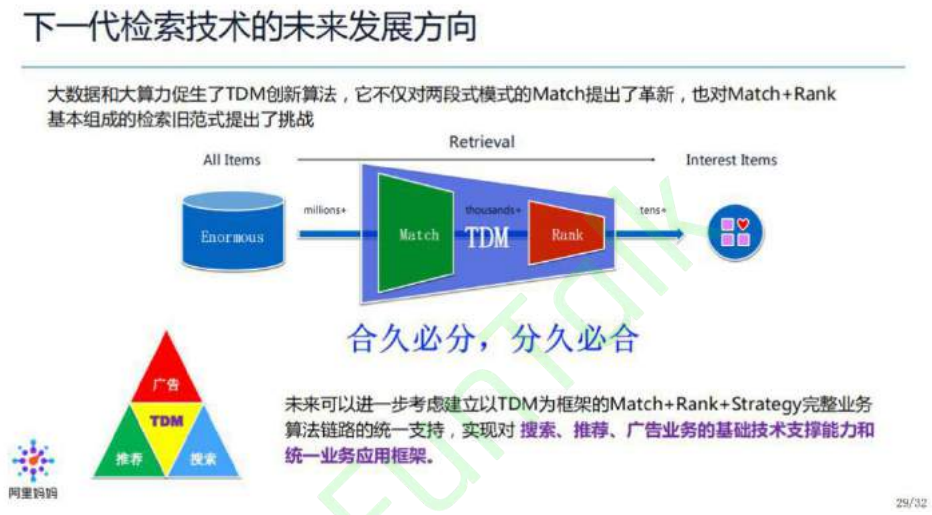
我们也正在集团内部推动 TDM 一站式服务化平台的建设，第一阶段实现 DIMO(Data In, Model Out),第二阶段实现 DISO(Data In, Service Out)，以期未来可以实现：算法工程师提供场景内行为数据，我们可以自动化构建 TDM 模型；算法工程师将 TDM 模型提交至云端，在线服务就可以自动加载运行，实现 TopK 检索。

TDM 也还在继续创新过程中，包括：

技术方面：实现千人千树、图空间增强、检索最优化；

业务方面：可解释性，基于树结构的广告白盒透出等；

生态建设：学术上正在承办 CIKM2019 比赛，工业上会持续开放、开源，加强云端建设能力。



“合久必分，分久必合”，受限于数据发展的爆炸式增长，和计算性能的限制，两段式 Match+Rank 统治了检索架构的主流，但在大数据+大算力的大环境下，我们需要对算法做出改变，为此 TDM 突破了原有的架构约束，做出了部分创新工作，实现了端到端联合优化的一体化检索框架。

当前的 TDM 不管是技术创新还是业务应用还处在发展初期，我们相信随着技术和业务的持续创新，TDM 未来有望发展成为一个面向搜索、推荐、广告业务领域的通用的底层框架。我们也非常欢迎各位可以一起来探讨、应用和改造 TDM 技术，实现更优的检索效果，取得更大的业务成果！今天的分享就到这里，谢谢大家。

快看漫画个性化推荐探索与实践

嘉宾：夏博　编辑：Hoh Xil

本次分享的主题是快看漫画个性化推荐探索与实践，主要包括：

- 业务介绍
- 技术挑战
- 技术探索
- 总结与未来规划

业务介绍

1. 关于快看漫画

DataFunTalk

快看世界创立于 2014 年，旗下快看漫画 app 是中国新生代内容社区和原创 IP 平台，截止 2019 年 7 月总用户量已经突破 2 亿，注册用户量突破 1 亿，月活突破 4000 万，绝大多数用户属于高活跃、高粘性的 95 后、00 后，快看漫画今年被 QuestMobile 等机构评为“最受 00 后欢迎的产品”。

很多人来到快看漫画，可能更多的是想看漫画，实际我们的内容不只是漫画，还有社区的 UGC 内容，从产品属性来讲虽然现在更倾向于漫画，但是我们在逐渐强化社区属性，也是未来重点的战略方向。所以，对于推荐来讲我们是长内容和短内容结合的。

2. 快看漫画推荐业务

上图为快看漫画的主要推荐业务场景，包括：首页个性推荐 tab，发现页推荐 tab，世界页推荐 tab，贴底相关推荐。画面会比之前好一些，对于推荐系统来讲，不光是技术、数据、算法，还和 UI/UE、领域知识相关。

内容形式包含：长漫画、短漫画、图文帖子、视频帖子等。

我们在做的事情就是如何为 4000 万月活用户很好的分发长内容和短内容。

技术挑战

1. 内容形式多样

短内容（短视频、新闻资讯、用户帖子等）特点：

占用用户碎片化时间，阅读时间短

兴趣点通常单一

长内容（漫画、小说等）特点：

- 占用用户大块的时间，阅读周期长
- 连续性、周期性、多章节多兴趣点

针对多样的内容形式，我们面临的技术挑战：

- 技术上如何捕捉长内容的连续性、周期性、多兴趣点等特征？
- 快看漫画既有长内容又有短内容，如何较好的融合两类内容？

2. 内容风格独特

快看漫画有大量的文本信息（帖子内容、弹幕、评论）和海量的图像信息（漫画图像、帖子图片），其带来的挑战为：

- 如何进行漫画类图像内容理解？图中古风的图片可能比较好理解，但是如何分辨校园和都市，通过图像是很难判别的。
- 独特的社区文化（比如二次元），新生代文化“暗语”（如上图帖子中的内容，对这方面不了解的人都很难理解，对于机器来说更难理解），给文本内容理解带来挑战。

技术探索：算法

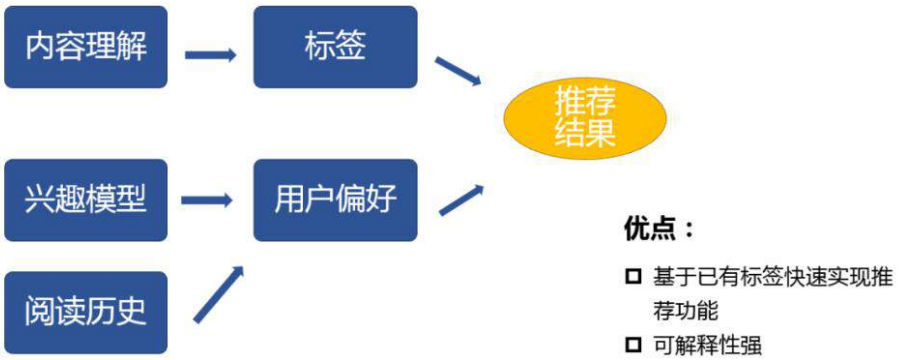
1. 推荐算法演进

如果现在界定为深度学习时代，各大公司的产品都已经上了深度学习模型，深度学习的效果是非常好的，但是它的平台搭建周期是非常长的，并且很难被解释，是一个黑盒的东西，看不到摸不到，很难干预。对于前深度学习时代，也就是传统机器学习模型来说，它的可解释性强，训练起来比较容易，并且容易部署。

2. 快看推荐算法迭代

快看推荐算法起步相对于推荐领域是比较晚的，但是相对于漫画垂直领域还是比较早的，我们在 2019 年以前更多的是基于内容的推荐，今年的上半年我们引入了协同过滤，同时 19 年到现在排序这块主要用到的是 XGBoost，未来我们会考虑深度学习。

3. 基于内容的推荐



基于内容的推荐，最大的难点在于对内容的理解，我们有比较专业的运营和内容团队，在做推荐之前已经有了一些比较基础和简单的标签，可以快速的应用起来，所以我们最早做的是基于内容的推荐。做内容推荐，我们需要有很好的内容理解，构造好物品的画像，另外，需要很好的理解用户的兴趣偏好，构建用户的用户画像，我们把两者很好的结合就可以得到推荐的结果。对于内容推荐来讲，它的可解释性也是比较强的，对于在内容方面有很深积累的公司，可以很快的构建起来。

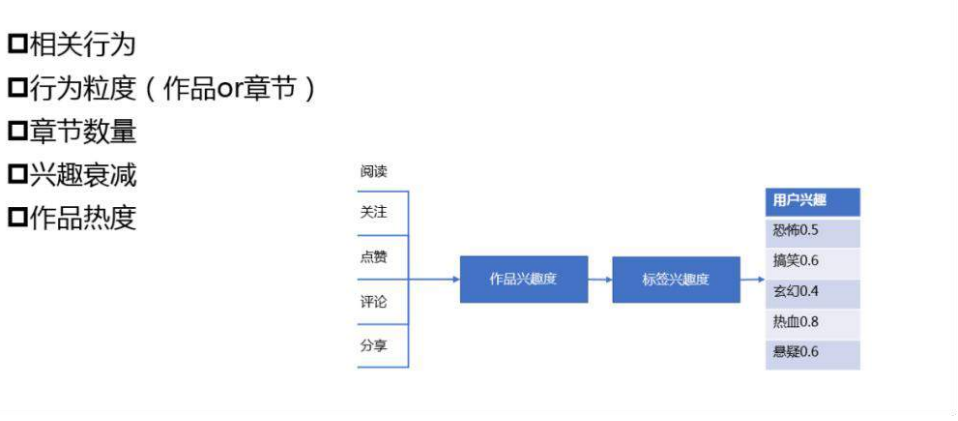
① 快看漫画标签体系

快看漫画的标签体系，分为三个维度：

- 作品基础维度：搞笑、青春、治愈等
- 用户分发维度：男性、女性、青少年等
- 内容创作维度：青春成长、兄妹、学生等

即使有专业的标签团队来打标签，建立很好的标签体系也需要很长的周期过程，因为人和人之间的感受和认知是有差距的，如何把这些标准制定好，保证每个作品打的标签是无差别的，这是一个专业性很强的问题。（上图为我们去年比较火的作品，被拍成了电影）

② 用户兴趣模型



做用户兴趣模型，需考虑：

相关行为：关注、点赞、评论、分享等

行为粒度：会精确到关注的作品或具体某个章节

章节数量：章节数量不等，有的作品很长，有的作品很短，如何判断用户对一个感兴趣，对另一个作品不感兴趣

兴趣衰减：用户的兴趣是周期性的，会存在兴趣衰减的情况

作品热度：需考虑热门作品，大家都在看的内容

③ 基于内容推荐总结

缺点：

- 非常依赖标签
- 推荐粒度较粗，召回不足
- 缺乏新颖性

基于内容推荐的总结，存在以下缺点：

- 非常依赖标签
- 推荐粒度较粗，如果用户兴趣单一的话，召回会不足
- 缺乏新颖性

但是，这是我们第一次上线基于内容推荐的模型，DAU 人均阅读次数率提升35%，效果还是很不错的。

4. 基于协同过滤

- 基于物品的协同过滤(Item-Based)
- 基于用户的协同过滤(User-Based)
- 基于模型的协同过滤(Model-Based)

之后，我们引入了协同过滤，下面为我们实现了的三种算法：

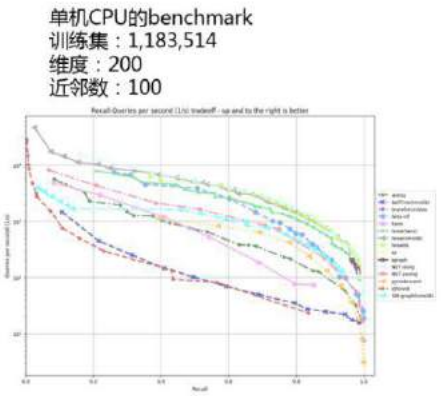
- 基于物品的协同过滤（Item-Based），对于漫画来讲，作品数量不是特别大，可以很快的离线计算完成。
- 基于用户的协同过滤（User-Based），由于我们有 4000 万的月活用户，做起来还是比较痛苦的，下面将重点介绍。
- 基于模型的协同过滤（Model-Based）

由于协同过滤都是基于矩阵来完成的，我们采用的是业界常用的 KNN 近邻算法。

① KNN 召回

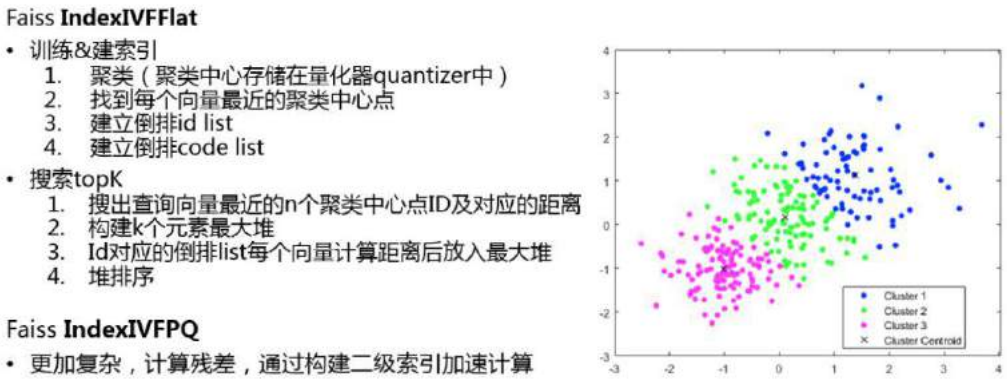
- 基于用户的协同算法用户相似度计算量巨大⊗
- ANN (Approximate Nearest Neighbor)
- nmslib vs. faiss

	Nmslib	faiss
实现语言	C++	C++
Python绑定	✓	✓
GPU支持	✗	✓
HNSW算法	✓	✓
其他算法	VP-Tree, NAPP, SW-graph	LSH IVF(IndexIVFFlat) IVFADC(IndexIVFPQ), IVFADC-R(IndexIVFPQR)



因为，基于用户的协同算法用户相似度计算量巨大，所以，针对 KNN 近邻算法，我们做了调研，对 Nmslib 和 Faiss 库做了对比：

它们都是开源的，可能 Faiss 会比较知名一点，因为是 Facebook 开源的，它们的实现语言都是 C++，都实现了 Python 绑定，但是 Faiss 会支持 GPU，都实现了目前最快的 HNSW 分层索引算法，右边为网上找的两个算法在单机 CPU 上的 benchmark，训练集大概 100+W，维度是 200，查找的是 100 个近邻。大家可以看到，最外层绿色的线就是 Nmslib 实现的 HNSW 算法，紧接着深绿色的就是 Faiss 实现的 HNSW 算法，对比 Nmslib 会慢一点，再往下一条线是 Faiss 实现的 IVF 算法，它会稍微差一些，但是它可以支持 GPU 并行计算，所以按照 GPU 去考量，那么这个明显是胜出的，所以我们综合考虑，选择了 Faiss 作为近邻计算的基础库。



这里简单介绍下 Faiss 实现的算法。

Faiss IndexIVFFlat 实现过程：

训练 & 建索引

① 聚类（找到聚类中心存储在量化器 quantizer 中）

② 找到每个向量最近的聚类中心点

③ 建立倒排 id（id 编号 ）list

④ 建立倒排 code（ 真实向量的倒排索引 ）list

搜索 topK

① 搜出查询向量最近的 n 个聚类中心点 id 及对应的距离

② 构建 k 个元素最大堆

③ Id 对应的倒排 list 每个向量计算距离后放入最大堆

④ 堆排序，最后做堆排序就可以得到 TopK

下面的 Faiss IndexIVFPQ，相当于一个升级优化版本，实现更复杂些，会计算残差，通过构建二级索引实现计算的加速。整体来说，我们实现了 User-Based CF 的实时在线召回。

② 基于协同过滤总结

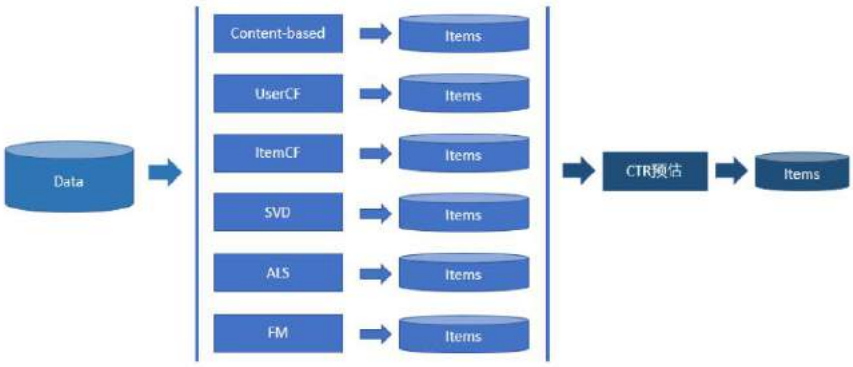
协同过滤上线后，DAU 人均阅读次数提升了 31%，同时协同过滤存在的缺点为：

倾向于推荐热门内容（ 当然可以通过一些方法对热门内容进行打压 ）

对新用户和新内容不友好

相似矩阵的计算量大（ 可以通过 ANN 的方式来解决 ）

5. 召回排序模型



我们有了基于内容的召回，基于协同过滤的召回，每个召回都有自己的排序结果，我们会考虑如何把这些结果合并起来，前期是基于规则的，后期我们采用 CTR 预估的方式，使用传统的召回+排序的结构。

① 常用 CTR 预估算法

LR

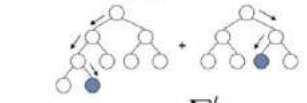
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

FM & FFM

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle V_i, V_j \rangle x_i x_j$$

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle V_{i,f_j}, V_{j,f_i} \rangle x_i x_j$$

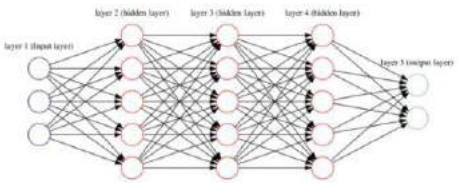
GBDT



$$F_m(x) = F_{m-1}(x) + \nu \cdot \sum_{l=1}^{L} \gamma_{lm} \mathbb{I}(\mathbf{x} \in R_{lm})$$

$$gradient_{m+1}(x) = label(x) - F_m(x)$$

DNN



常用的 CTR 预估算法有：LR、FM & FFM、GBDT（ 树模型 ）、DNN（ 深度神经网络 ）。

LR 模型

优势：

- 模型简单，善于处理离散化特征（ 包括 id 类特征 ）
- 容易实现分布式，可处理大规模特征和样本集

劣势：

- 特征需要离散化
- 特征之间在模型中是孤立的，需要做大量特征工程来做特征交叉

XGBoost 模型

优势：

- 树模型具有一定的组合特征能力
- 善于处理联系特征，可进行特征筛选，人工特征工程量少

劣势：

- 具有很强的记忆行为，不利于挖掘长尾特征
- 组合特征的能力有限

FM & FFM 模型

优势：

可以自动进行特征间的组合

通过引入特征隐向量，加速了训练的复杂度，善于处理稀疏数据

劣势：

工作量接近深度学习，效果不如深度学习

FFM 计算复杂度高

DNN 模型

优势：

- 可直接输入原始特征，减少交叉特征选择
- 效果好

劣势：

- 可解释性差
- 模型可能较大，调参复杂，需要较大的工程支持

综上，我们最终选择人工特征工程量较少的 XGBoost 方案。

② 召回排序模型总结

现状和问题：

- ❑ 特征还需要进一步挖掘
- ❑ 模型的训练效率有待提升
- ❑ 探索尝试新模型提升效果

上线召回排序模型之后，DAU 人均阅读次数提升 36.6%，目前的现状和问题：

- ❖ 特征还需要进一步挖掘
- ❖ 模型的训练效果有待提升，需要工程上的提升
- ❖ 探索尝试新模型提升效果

技术探索：系统架构

1. 架构的重要性

架构的重要性：算法是大脑，架构是骨架，如果没有好的推荐系统架构，算法很难落地。

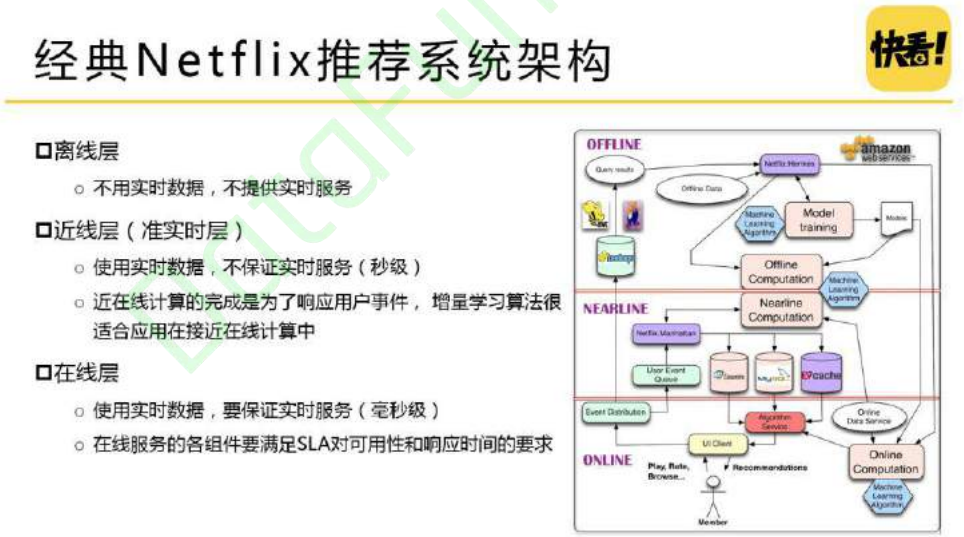
好的推荐系统需要具备的特质：

- ❖ 实时响应请求
- ❖ 及时、准确、全面的记录用户反馈

- 优雅降级，即使在服务出现问题的时候，也能推荐出个性化的结果

- 快速迭代推荐策略、算法

2. 经典 Netflix 推荐系统架构



这是 Netflix 在 2013 年公布的推荐系统架构，把推荐系统分为了三层：

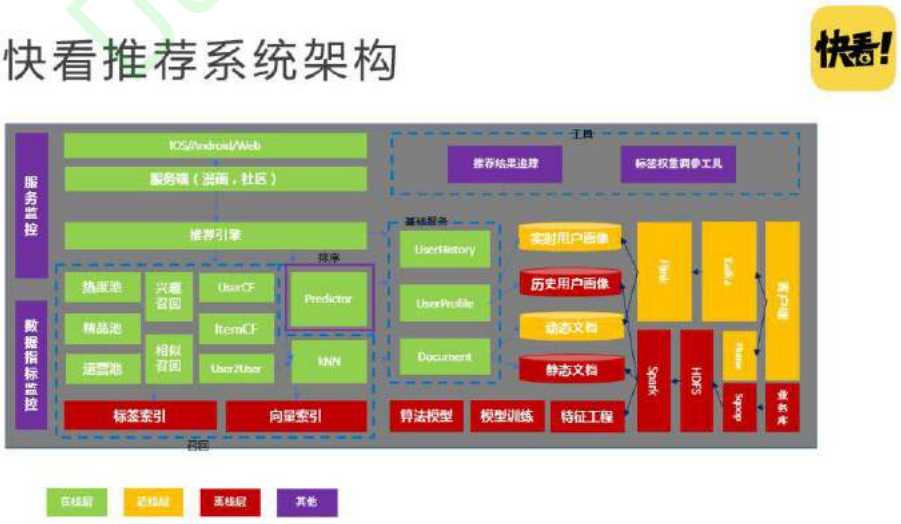
- 离线层：一个用户产生行为，通过事件分发，分发到离线层和近线层，离线部分是通过 hive 和 pag 这种通过离线的任务把数据分发到模型训练和一些离线计算上。

- 进线层（准实时层）：近线层有个组件叫 Manhattan，相当于今天常用的 Flink 和 Strom，把实时计算结果存储到 Cassandra 相当于 HBase，然后还有 EVcache 相当于 redis。

- 在线层：在线层会用离线计算的模型和近线计算的结果，得出在线的排序结果。

这就是当时 Netflix 的推荐系统架构。

3. 快看推荐系统架构



我们在做快看推荐系统架构的时候，实际上是没有参考 Netflix 的架构，但是，当我们完成之后发现，各个层也可以按照这个方式去划分：

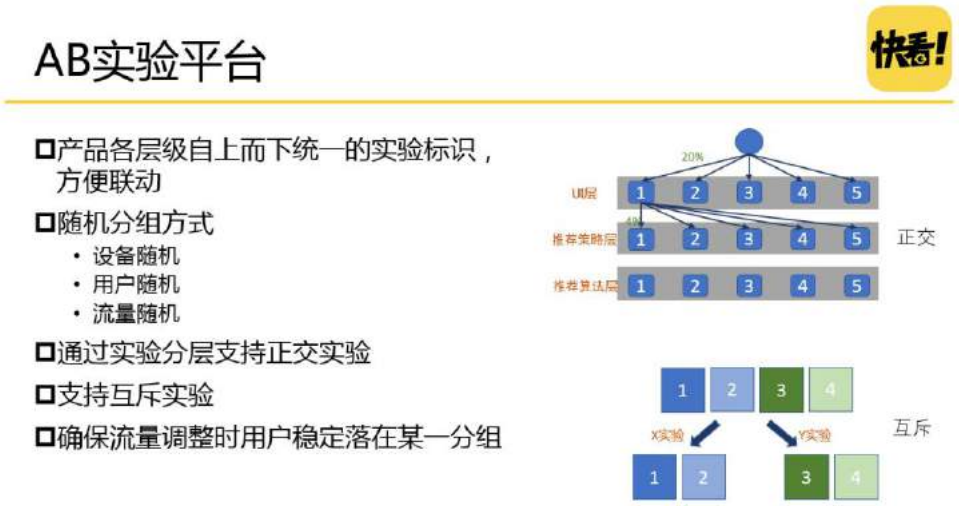
近线层（橙色，实时数据流过程）：客户端采集的日志数据，通过 Kafka、Flink 传递到实时用户画像和动态文档。

离线层（红色）：业务库数据通过 sqoop 导到 HDFS 后在 Spark 上计算，然后是离线模型，包括特征工程，模型训练，算法模型，向量索引，用户画像等等。

在线层（绿色）：包括在线的召回、排序、推荐、服务端、ios/android 等等。

工具（紫色）：标签权重模型、推荐结果追踪、数据指标监控和服务监控。

4. AB 实验平台



快看的 AB 实验平台在功能上是非常完善的，是从产品各层级自上而下统一的实验标识，方便联动；实现了设备随机、用户随机、流量随机的随机分组方式；通过实验分层支持正交实验，可以在一个层做多组实验；同时支持互斥实验，确保流量调整时用户稳定落在某一分组。

对于指标计算，进行了显著性的总结和功效的总结，并且指标可配置，在做实验的时候想关注哪些指标可以进行配置，方便查看算法实验的效果。

5. 推荐结果追踪工具

推荐往往会有一些 Bad case 暴露出来，如果没有做追踪，就很难查找那块儿出了问题，因此我们做了个性化推荐全链路的跟踪系统，保证了推荐的结果是因为什么推荐的，或者为什么没有被推荐，这样就保证了一个可解释性。如何解决的？我们会把当时的历史画像 Snapshot 和上下文，通过 HBase 记录下来。

总结与未来规划

1. 总结

本次分享主要介绍了快看和快看的推荐业务，从算法和系统两方面介绍了快看推荐技术在起步阶段的一些探索，并且介绍了大规模 k 近邻计算方法、AB 实验平台搭建等常用技术的落地方案。

2. 未来规划

内容理解是推荐业务的基石，目前这块儿还比较欠缺，未来将探索漫画领域的图像和文本内容理解技术。

传统机器学习方法探索充分之后将尝试深度学习推荐算法，以期更好的推荐效果。

浅谈 UC 国际信息流推荐

嘉宾：杰雄　　编辑：Hoh Xil

导读：本次分享的主题是浅谈 UC 国际信息流推荐。会跟大家探讨下发链路中常见的一些问题，主要包括两个场景：

列表页排序，包括：目标确定、多目标任务以及混排组合优化。

内容冷启问题，如果不做内容理解，能否做好推荐系统？

列表页推荐

这是印度语版的推荐列表页，左边跟常见 feed 推荐的产品形态是非常类似的，有不同的异构的卡片：

新闻聚合页，点开以后就是一个聚合页。

视频，点开是一个沉浸式播放的聚合页。

普通的图文，点开是一个落地页、详情页。

Memes，印度市场特有的内容 Memes。这种内容主要是一张图片（或者动图），这种内容比较特殊，可以直接在列表页消费，直接看了就曝光、阅读完成，就结束了，如果点开的话就是 Memes 的沉浸式页面。

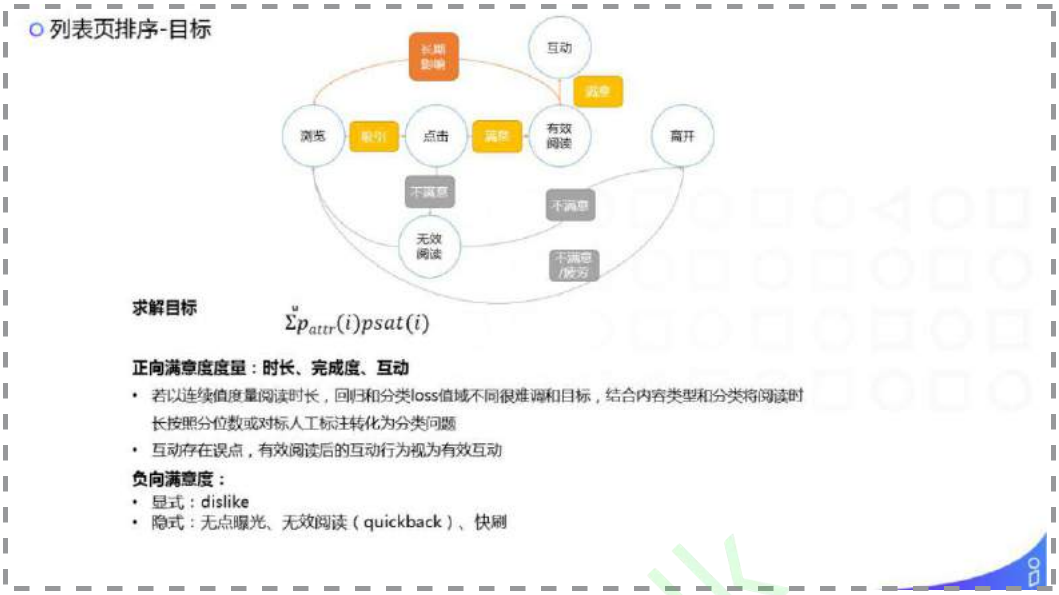
总结起来，内容消费的路径有：

一种是，列表页里直接消费的内容，如 Memes。

一种是，落地页中消费的内容。

还有就是通过聚合页再次跳到落地页消费的内容。

1. 目标确定



接下来讲下如何确定目标。对于推荐系统来说，最核心的就是如何确定目标，如果目标定不好，可能就会出现标题党的问题。在解释最终求解目标之前，先看下用户的行为路径：

图中圆圈表示的是用户的一种行为，方框表示用户发生这种行为的心里活动。

比如用户看到一篇内容之后，如果这个内容有吸引用户的点，也就是产生了吸引，会发生一次点击，在点击看到详情页的内容之后，如果对这个内容比较满意，用户可能会形成一次有效的阅读，有一次有效阅读之后，如果用户还是觉得这个内容非常好、非常满意，用户可能会有一些互动的行为，比如分享、点赞、评论等。

当然还有种可能：用户在列表页里看到一篇内容之后，用户不是很感兴趣，直接就跳走了或者快速的划过；再就是用户点开了一篇类似标题党的内容，

但是内容完全不是用户想要的，这其实是一个强烈的不满意会，一个无效的阅读，然后用户就离开了。如果把所有不满意的行为看作是一种负向的满意度，我们建模的核心目标应该是一个用户累计的所有满意的行为，使满意行为累积量最大化。

这里所有标黄色的路径其实是一个偏正向的路径，标灰色的路径是个偏负向的路径，我们的目标是使正向的路径逐渐的累积，对用户逐渐的产生一个比较正向的影响。

所以求解目标是：

$$\sum p_{attr}(i) psat(i)$$

左边为吸引的概率，右边是满意的概率，然后所有看过的内容满意度最大化。

如何衡量有效的阅读？一个传统的方法是用阅读时长来衡量是不是一个满意的阅读，但实际上用户满意的心理和时长不是一个完全线性的关系。比如有一类行为是用户阅读了 5s 或者 10s 以下快速离开（quickback），这种无效阅读，无论是 3s、5s 还是 7s、8s，效果都是用户对内容完全不满意，应该快速离开的。再有，当用户读一篇长文时，大家可能都有这样的体会，长文阅读可能会有一个瓶颈，就是大家花了很长时间在一篇文章阅读上，但是读到一定程度的时候，可能再也读不下去了，能花的时间就存在一个极限了，

所以最后满意度和时长关系是类似 sigmoid 的函数关系。因此，我们在对满意度建模时，其实是把回归问题变成了一个分类/二分类/多分类的问题。这里可能会涉及怎么做时长回归的问题（由于不同类型、分类、主题的内容以及内容信息量的不同，其阅读时长总量是会变化的），一种简单的方法是用这些维度，对内容进行统计分析求出分布，然后用分位数来截断，通过人工来排出几个档，也可以做一些人工标注来拟合这样的分类。结合 UC 国际信息流，稍微特殊的一点是列表页有 Memes 这样的图片内容。这种内容由于强调的是互动性（一般承载的是一些高分享类的内容，如早安、节日问候、搞笑的图片等），在产品设计时，会把这种交互行为做前置，在列表页就放出来，这样就可能存在误点，用户还没看到或看完这篇内容，就产生了点击，需要做一些过滤。

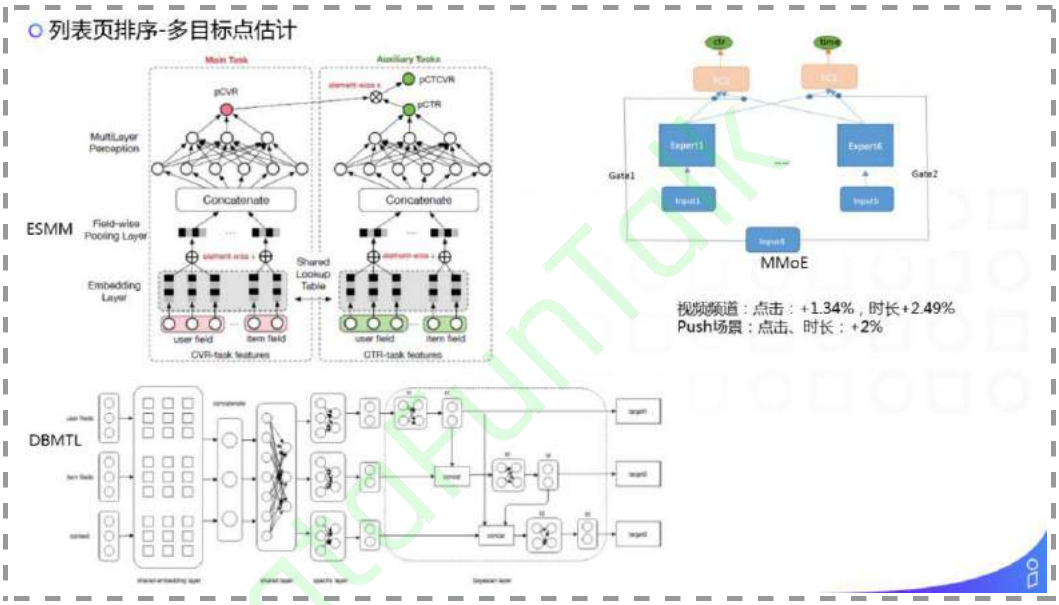
负向满意度，分为：

显式：很多产品在设计时，都会在内容边上有个 XX，也就是 dislike，比较直白的显示了负向满意度。

隐式：无点的曝光、无效阅读（quickback）、快刷等动作。

说完了总的求解目标之后，这里列举的吸引和满意，满意还可以拆解成更多的步骤，比如刚才说的有效阅读和互动行为，可以再做分解，但无论如何都是一个多目标的任务，针对这样的多目标任务该如何建模呢？

2. 多目标点估计



这里列举了一些方法，都是阿里巴巴集团内部在各个业务线上的一些沉淀：

① ESMM：

这是阿里妈妈团队在解决多目标问题的一种解决方案。ESMM 可以解决我们在对多目标问题进行求解时，比如左边是转化率的目标，右边点击率目标，往往是独立进行求解的。使用样本时，该如何表达上文说到的转移概率？常规做法用到的样本，如转化率使用的样本是所有的有点样本，由于在训练时，使用的样本是部分样本，在预测目标时，使用的是全样本，导致样本分布会存在一定的偏差。ESMM 是把样本空间放到全空间，在定义目标，计算 loss 时，计算的都是全样本空间的 loss，一个是点击率，再有一个是把 CVR 作

为一个中间节点，最后求解的 loss 目标是 CTR * CVR，然后底层网络参数共享。

② DBMTL：

DBMTL 模型是淘宝推荐团队对 ESMM 模型进行的改进。主要的改进点：左边这部分就相当于 ESMM 那张图横过来了 ,是共享参数层 ;specific layer 是走的不同目标的分支；最重要的是右边 bayesian layer，表达了概率图中目标之间的贝叶斯转移概率的因果关系。如果转移概率之间的关系，受其他的一些 feature 和因素的影响，也可以把那些 feature 加到网络中一起训练，所以 DBMTL 建模的时候还建模了几个目标之间的因果关系。

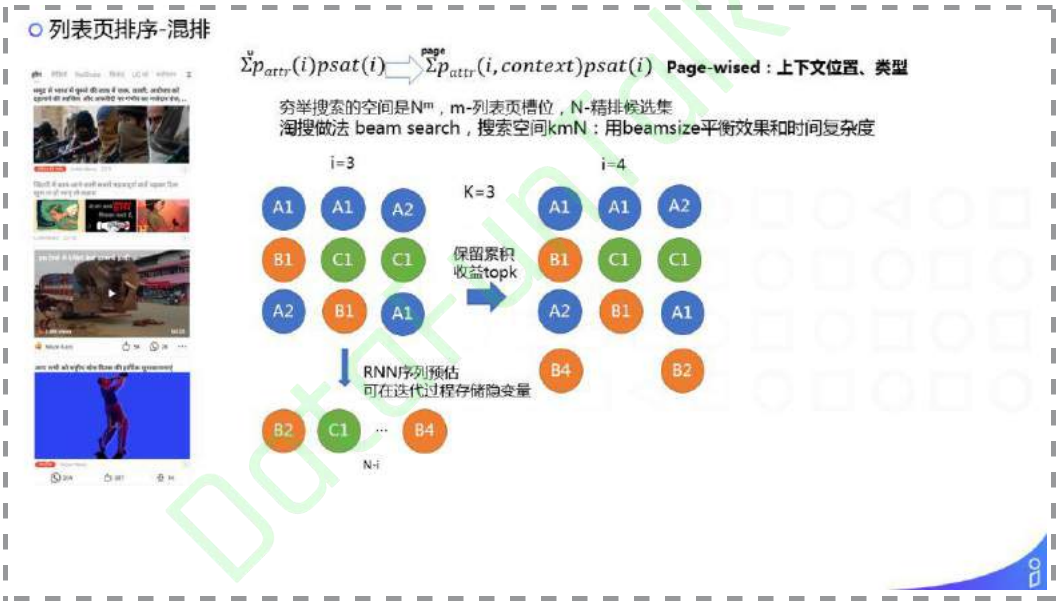
③ MMoE：

MMoE 类似一个专家系统，有多个子网络，每个子网络使用的特征和网络结构可以有差异，在最终确定多目标的时候进行票选，通过 gate 来赋予不同的权重来做票选。

最后，我们的业务在不同场景上都取得了比较正向的收益，如视频频道和 Push 场景。

多目标这儿写了一个点估计，因为主要用在精排的场景，在做每次的预估时，考虑的都还只是某一条内容的满意度效果。

3. 混排



但是在列表页场景，我们要求解的是一个组合最优的效果，也就是说对上面的问题需要做进一步的扩展。在考虑点击率时，还要考虑上下文，我上下文的信息。

然后我们的求解目标也做了一个转换：

$$\sum^{u} p_{attr}(i)psat(i) \Rightarrow \sum^{page} p_{attr}(i, context)psat(i)$$

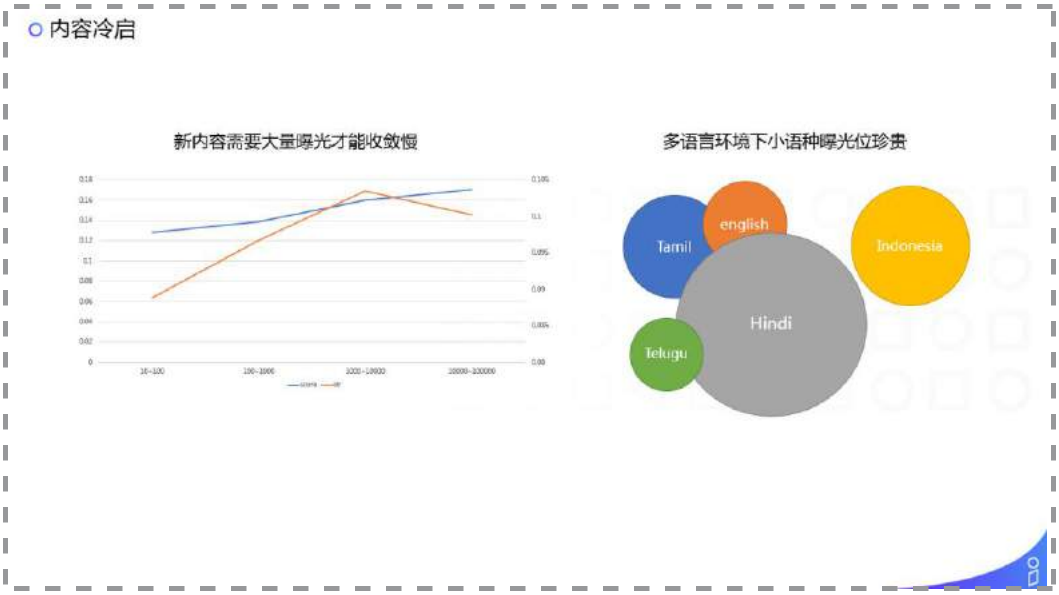
U 转到 page，做了一个独立假设，认为页与页之间是没有关联关系的，但这个假设不一定成立，只是为了把问题简化一下。这样我们的问题就变成了在组合列表页的情况下 ,如何达到组合最优。比如有 N 条精排的候选集输出，对这 N 条结果输出 M 个槽位的排列，也就是求解排列的最优解。所以搜索

空间相当于是 N 的 M 次方，一个非常大的搜索空间，在实际业务中是没法落地的，因为计算复杂度太高了。

所以淘搜这有一个简化版的方法：类似于贪心的一种算法，在每一轮只确定当前这条最优的结果，然后考虑上文，不考虑下文。举个例子，比如现在是第三轮迭代，已经确定了前三个位置的最优组合，现在是求解第四个位置应该选哪一条，预测第四条那个位置最优的一个选项。另外在做贪心搜索时，搜索空间非常受限，受选择顺序的限制。那么 beam search 有一个参数，就是宽度，每次可以把候选集保留 top3，也就是最优组合的 top3 作为候选，再进行下一次的探索，有一定的探索回溯的能力，这样 beamsize 探索的最优空间的大小，可以用来 balance 性能和效果。

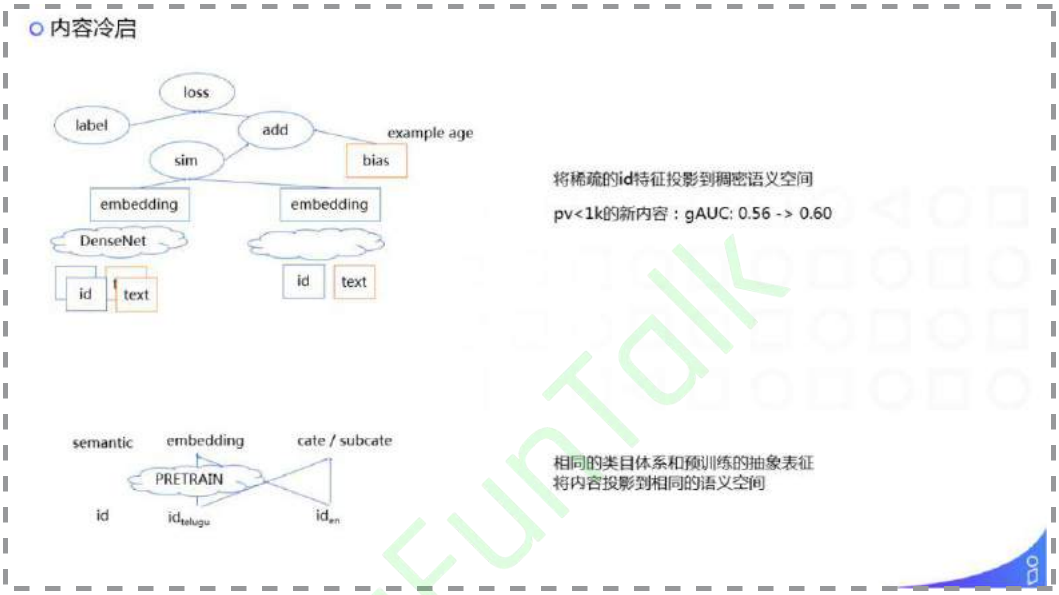
还有一点是对上文的建模，怎么表达上文的排列信息，也就是上文内容之间的位置相对关系的信息。这里有一个比较巧妙的方法：用 RNN 的网络结构来表达他们之间的相互关系，然后在计算的过程中（因为是一个贪婪式的计算，每一轮只计算一步），只计算 RNN 的一个 STEP 可以了，所以在时间复杂度上也是可以接受的，这样存储一个 RNN 中间的隐变量就可以了，这就是混排的做法。

内容冷启



内容冷启问题，有朋友问，如果一个推荐系统完全不做内容理解，是不是也是可行的？这里从其中一个角度说下。

左图是我们现在只用 ID 作为 feature，也就是说核心 feature 是内容的 ID，相当于没有内容理解，这是一个新内容收敛的效果，下边横轴是下发的 PV，可以看到点击率的收敛，要基本上要到千次左右的下发才能达到一个接近收敛的程度，而且起步阶段和后面其实 gap 还是非常大的，用 ID 做推荐就会遇到这样冷启的问题，特别是我们的业务场景又涉及到小语种。因此，对于场景，无论是 item 还是流量都做了很多的细分，所以冷启的问题会尤其的严重一些。



解法其实还是内容理解，我们把 ID 的特征映射到文本域。这里有一个 YouTube 15 年做的一个工作，怎么去掉时间的 bias，就是 Example age，其含义是在采用这条样本时，这个时间点距离这条内容发布的时间点之间的 time diff 时间差是什么样的。加上 bias 的 feature 之后，对时间敏感的内容在下发时就会考虑到下发时间差，相当于提高了时效性。

我们在召回侧 DM match，把 ID 特征加上文本特征之后，冷启（1000 条以下的曝光）内容 AUC 有了不错的提升。

另外，在多语言的体系下，如果能把文本域的表征对齐投影到同一个空间，对冷启应该能起到更好的效果。

总结

简单总结下，结合大家常见的一些问题，本次分享主要介绍了排序中如何确定目标，如何做多目标的点估计以及混排的组合优化，还简单介绍了内容冷启的一些解决思路，主要是特征泛化和语义对齐。本次分享就到这里，谢谢大家。

DataFunTalk

UC 信息流视频标签识别技术

嘉宾：雪萌　　编辑：Hoh Xil

导读：本次分享的主题为 UC 信息流视频标签识别技术，主要介绍标签识别整体架构以及基于多模态信息的视频标签识别方法，实现让机器理解海量的视频中的关键信息。

包括以下几部分：

- 标签使用场景
- 标签识别系统架构
- 标签识别算法
- 未来工作

标签使用场景

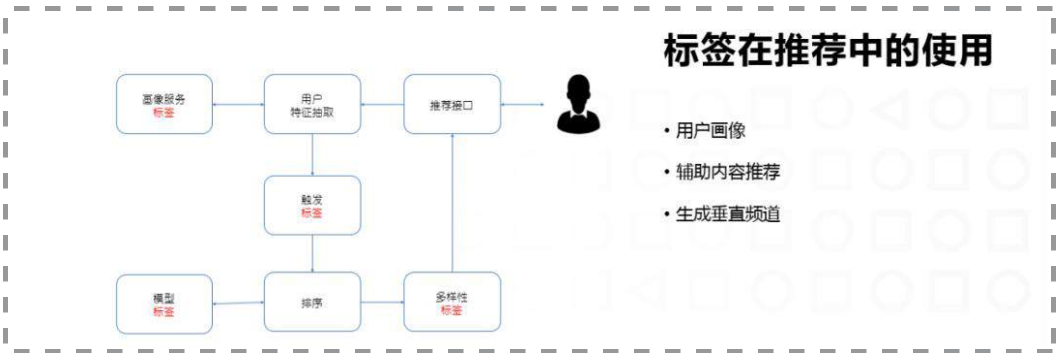
首先介绍下为什么使用标签以及标签的使用场景：

1. 什么是视频标签

DataFunTalk

UC 信息流推荐比较多的内容都是新闻资讯类的，每天新的内容都非常多，如果没有内容理解结构化的信息，是比较难做内容冷启动的。视频标签可以理解成描述一个视频的几个关键词，当我们看到视频标签的时候就可以大概知道这个视频的内容。这几张图片是我们信息流小视频的几张截图，大家可以看下可以给这几个小视频打上什么样的标签，比如第一个视频会打上时尚、穿搭、小姐姐等标签，第二个视频会打上美食、烧烤等标签。当我们提取这样的标签之后，该如何使用呢？

2. 标签在推荐中的使用



标签在推荐中的使用大概有三个场景：

① **构建用户画像**：当用户点击前面时尚穿搭的视频，我们可以大概推测用户可能是对时尚比较感兴趣的女性用户，然后我们会通过对用户一系列的历史行为更精准的刻画用户的画像。

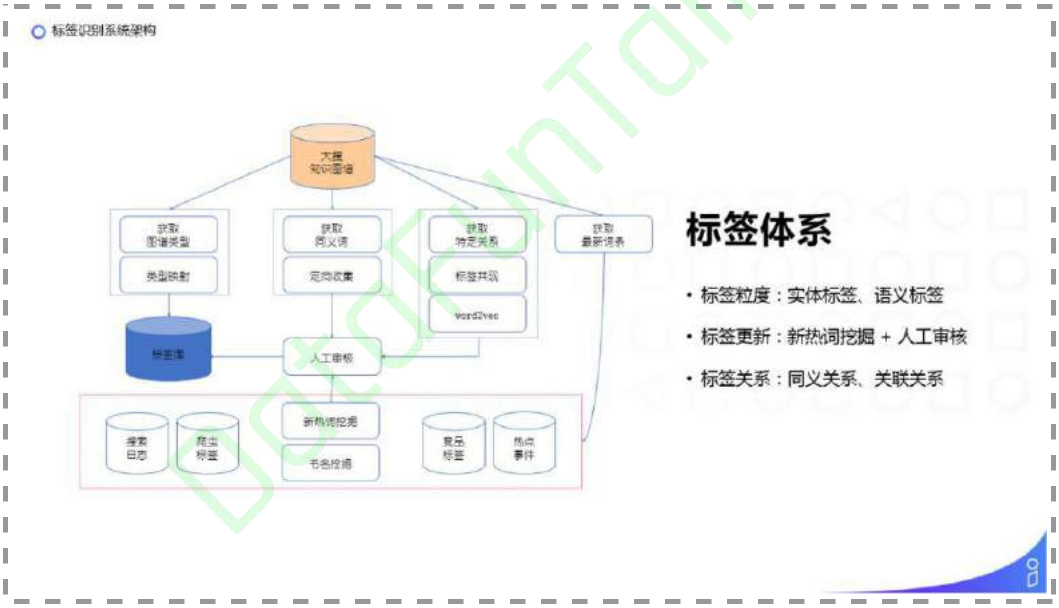
② **辅助内容推荐**：当用户触发一次刷新时，可以参考左边的图片，会调用推荐的接口拉取用户画像中用户的特征表示，然后从用户特征中匹配对应的标签作为 Key 来拉取候选的视频或资讯，放入一系列的排序模型中排序（这样的排序模型也会使用标签作为特征进行训练），再对排序之后的候选列表使用标签来控制多样性，避免推荐单一内容使用户产生烦感（我们会使用标签训练一些相似性的模型来判断两个资讯的相似度做一些过滤或者直接限制每一刷同样标签出现的次数），最后对经过排序和过滤之后的候选列表通过阈值截断，把最终的结果返回给用户。

③ **生成垂直频道**：比如有娱乐、搞笑、美女等垂直频道，可以方便用户直接获取感兴趣的内容。

标签识别系统架构

接下来介绍下标签识别的系统架构：

1. 标签体系



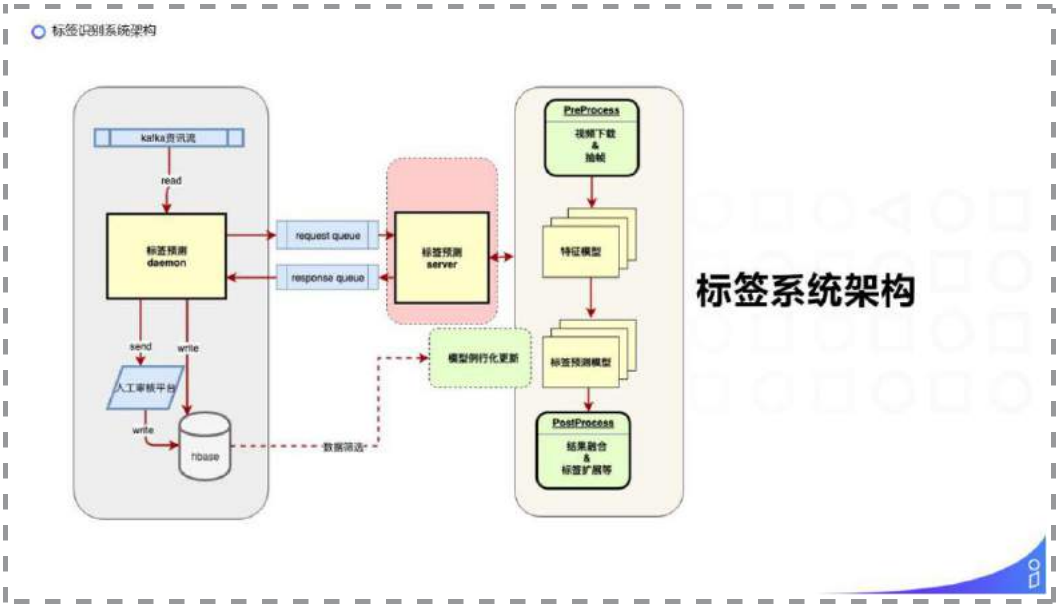
我们需要知道什么样的词适合当做标签来使用，因此，需要维护一套标签体系：

① **标签粒度**：在信息流的场景下，标签分为实体标签和语义标签，以一部影视剧为例，具体的某一部电影就可以是一个实体标签，某一类电影，比如喜剧片或者动作片，就是一个语义标签。

② **标签更新**：由于不断的有新的词出现，需要不断的更新标签库，标签来源（图中红框部分）：搜索日志、爬虫标签、竞品标签、热点事件等。我们会从来源中挖掘出新的候选标签，然后从大搜的知识图谱中拉取标签对应的属性，根据标签属性和标签库已经存在的标签挖掘它的同义词以及关联关系，再经过人工审核把标签和标签对应的属性入到标签库，供后续的使用。

③ **标签关系**：同义关系、关联关系（上一步已经介绍，不再复述）

2. 标签系统架构



下面介绍下标签的整体架构：

我们会用标签预测的 daemon 不断的消费 kafka 的资讯流，kafka 的资讯流包含新入库和属性发生变化的资讯。标签预测的 daemon 会去筛选新入

库的资讯或者包含有模型依赖属性发生变化的资讯去请求标签预测的服务，通过把请求写入 kafka request queue。标签的预测服务会按照右边的流程做预测。首先把视频下载下来，进行抽帧或者文本的预处理，然后把预处理之后的数据拿到特征模块中抽取图像、视频、文本等特征，然后把抽取的特征输入到一系列的标签预测的模型中，把得到的每个模型预测出的结果放入后处理的模块中做融合、去重或者扩展，得到最终返回的标签预测结果，同样把结果写到 kafka response queue 中，然后返回到标签预测的 daemon，标签预测的结果直接写入 HBase 或者推到审核平台，经过人工审核之后再写入 HBase。由于有一些领域的标签变化是比较快的，比如影视剧或者综艺的标签，对于这样的标签预测模型，我们会定期重新拉取数据做例行化的训练。

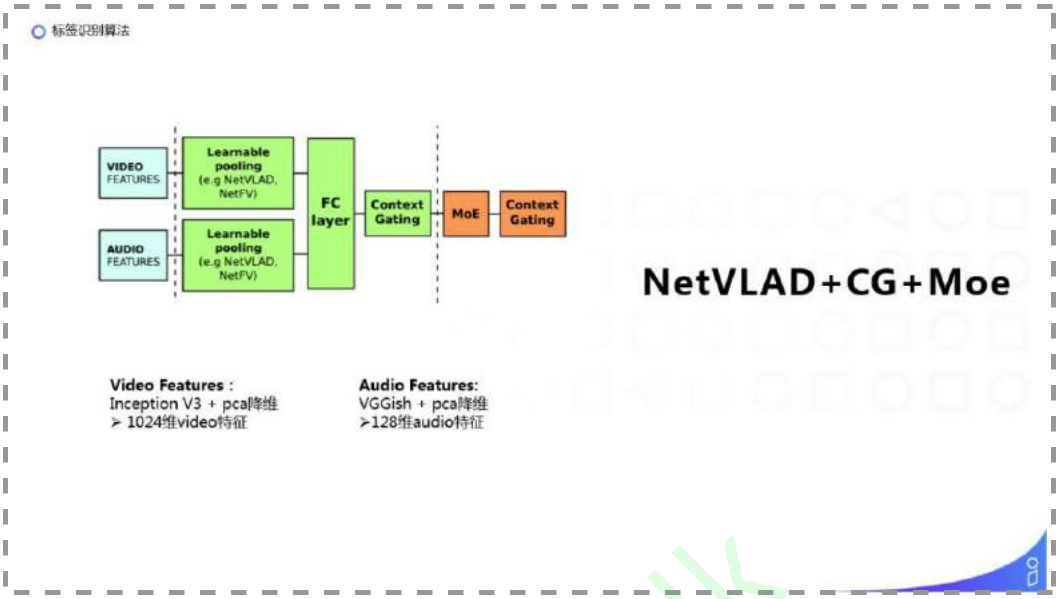
标签识别算法

接下来主要介绍下标签识别的算法：

我们会使用 NextVlad 进行全局视频标签识别，在整体的标签识别上得到不错的效果，但是对一些长尾的和专业领域的标签处理的还不够好，对于一些专业领域的标签会再用更适合的模型做处理。比如使用 3D 识别行为标签和使用 mtcbb+InsightFace 识别人物标签。

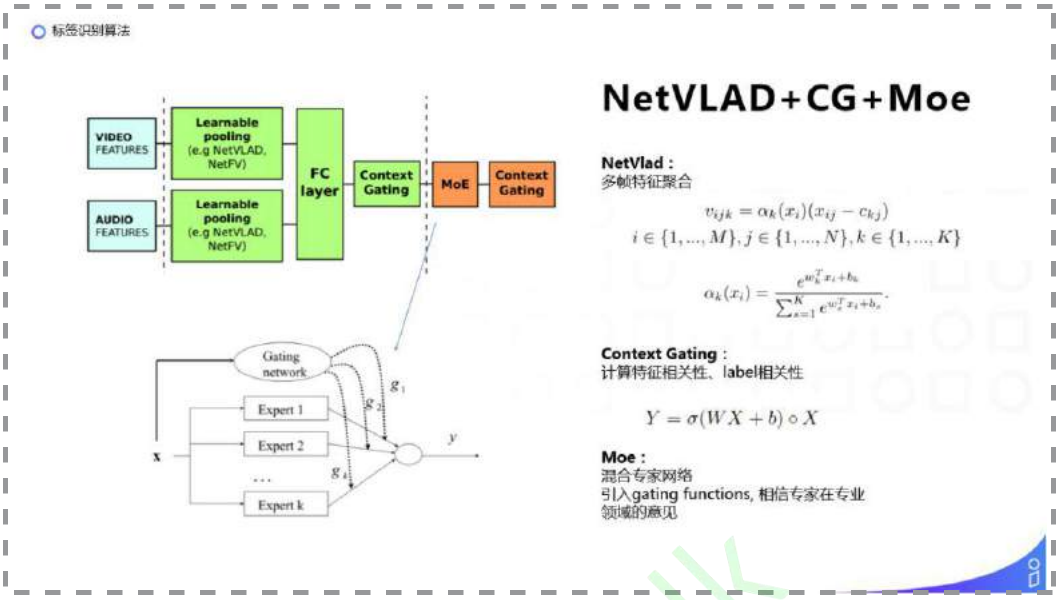
1. NextVlad

最初方案：



NetVlad 是 2017 年 YouTube 8M 视频理解比赛（主要给 YouTube 的视频打标）中提出来的一个网络结构，主要分为三个部分：左边是特征的输入，中间是视频的特征提取，右边是模型的分类阶段。

① **视频特征** 通过 Inception V3+pca 降维得到的一个 1024 维的视频特征；音频特征：经过音频的预处理，使用 VGGish+pca 降维得到的 128 维音频特征，会每秒进行一次采样，相当于每秒会得到一个 128 维的音频特征，VGGish 模型训练使用的是 Audio set 数据集，类似于在图片领域的 ImageNet 这样的模型，主要是做的是声音的分类，包括动物的声音、机械的声音或者音乐的风格等。



② **特征提取**：中间的特征提取层使用了双流的 NetVlad 模块，分别提取视频和音频的特征，之后拼接一个全连接层，然后通过 Context Gating 的机制来重新计算特征的相关性。NetVlad 的主要作用是（由于视频的长短可能是不一致的，输入的特征维度是不一样的。）通过把局部的特征聚合为整体特征的方式，得到一个固定长度的向量表示。

我们看下右边的公式：

NetVlad：

多帧特征聚合：

$$v_{ijk} = \alpha_k(x_i)(x_{ij} - c_{kj})$$
$$i \in \{1, ..., M\}, j \in \{1, ..., N\}, k \in \{1, ..., K\}$$

i 表示第 M 帧，j 表示第 N 维特征，k 表示聚类的个数 K。$\alpha_k(x_i)$ 类似 Softmax 函数计算得到的是第 i 帧属于第 k 个聚类的概率，再乘上 j 维特征与 k 个聚类第 j 维的距离得到每一帧的 N x K 维的向量，通过对所有的 M 帧的 N 维向量进行累加，得到 N x K 维的视频级别向量：

$$\alpha_k(x_i) = \frac{e^{w_k^T x_i + b_k}}{\sum_{s=1}^{K} e^{w_s^T x_i + b_s}}$$

Context Gating 作用：

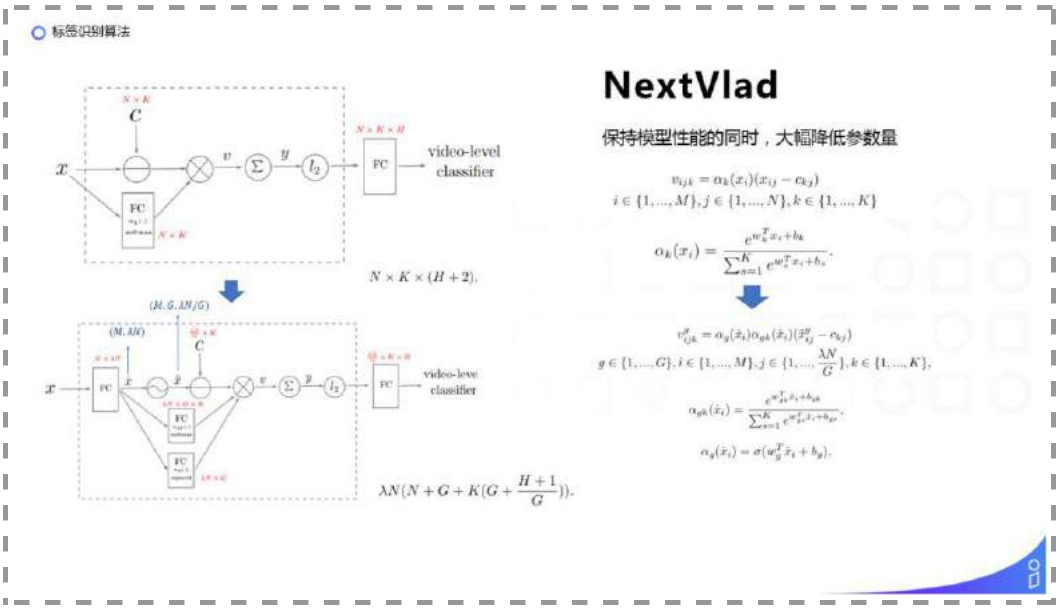
一方面，计算特征的相关性，使得比较重要的权重变的更大，不相关的权重变得比较小；

另一方面，引入一些非线性，使其效果更好。

Moe：

③ **分类阶段**：使用的是混合专家网络（如左下图），把输入 X 分别放入 K 个专家网络，每个专家的单元可以看做一个简单的分类器，再用 gating 网络训练每个分类器的权重，通过线性的加和得到最终预测的效果。这样做的目的是，我们认为每一个专家在他自己的专业领域里会做的比较好，最终的结果是相信了每个专家在他专业领域中的意见，对其进行整合，得到更好的效果。

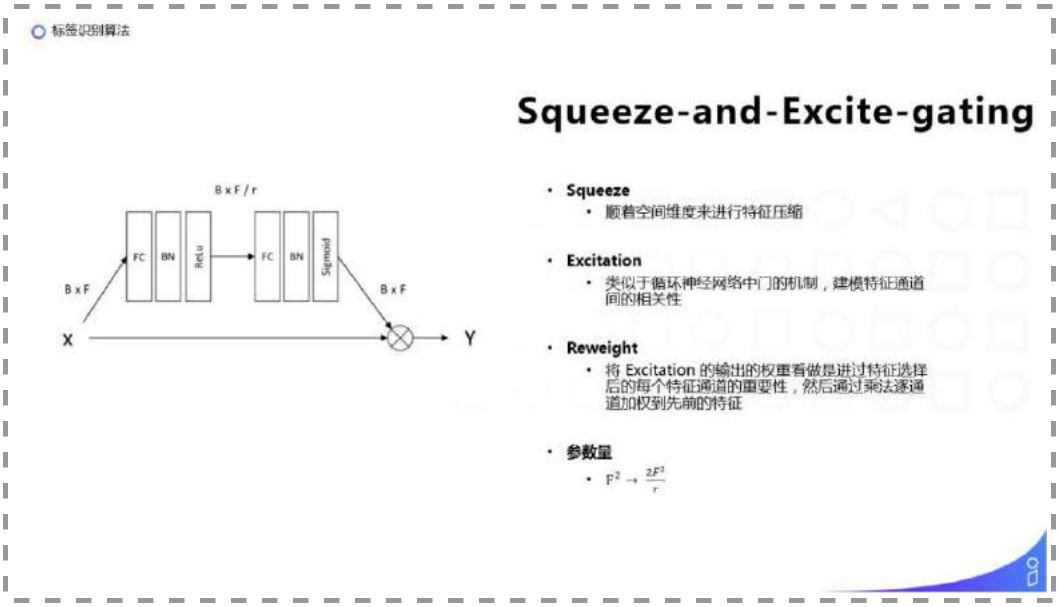
NextVlad：



我们使用的 NextVlad 模型是在 NetVlad 的基础上进行改进的。NextVlad 是 2018 年 YouTube 8M 比赛中提出来的网络结构，在保持模型性能的同时，大幅降低了参数的数量。上面是原始的 NetVlad 单元的网络结构，下面是改进之后 NextVlad 的网络结构，标红的部分表示需要训练的参数量。NextVlad 首先是把输入 x 的 N 维特征扩张到 λN 维特征，得到 $\dot{x}_i$，再对 $\dot{x}_i$ 做 reshape，得到（M，G，λN/G）的 x 波，这样在我们训练分类的时候，降到了 λN/G x K 的维度， 然后使用 Softmax 计算 $\dot{x}_i$ 属于第 K 个聚类的概率，使用 sigmoid 的门函数计算 group 权重，再通过一个连乘得到 i x G 个的聚类维度乘以 K 的向量，然后通过累加得到了一个 λN/G x K 的 y 维向量，这样在进入到全连接层，全连接层的维度就从 N x K x H 降到了 λN/G x K x H 的维度，这样做的话就使我们在方框内的特征

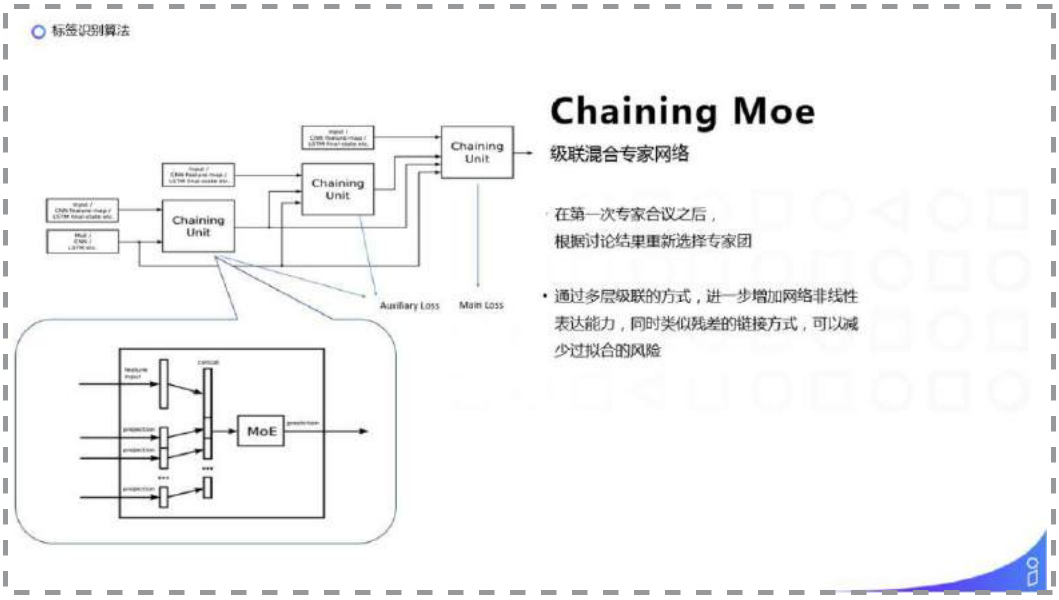
encoding 阶段整体的特征参数是增加的，可能更好的做特征表达，而在特征全连接层的特征量是下降的，达到了整体的参数量有所下降，而模型性能仍然能保持的效果。

Squeeze-and-Excite-gating：



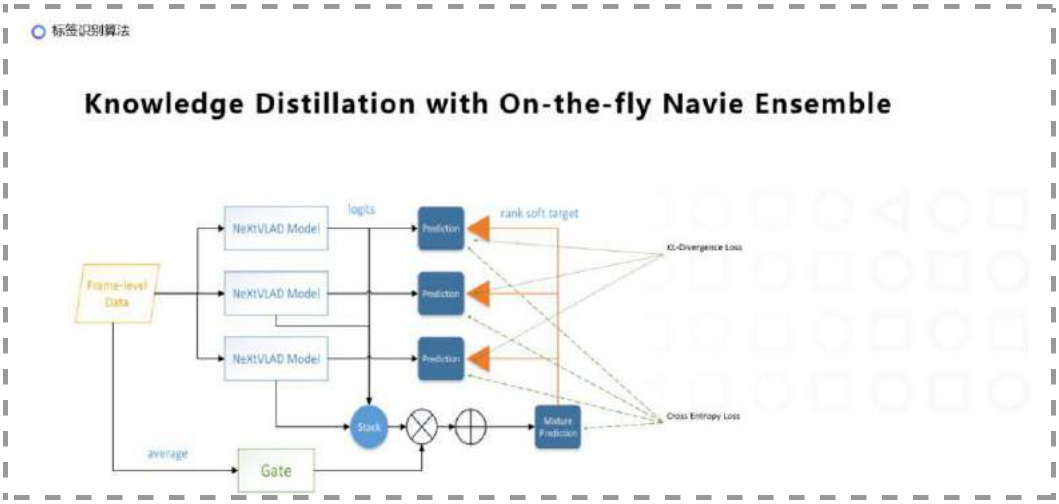
提出 NextVlad 的那篇 paper 同时还改进了 Context Gating 结构，通过把 squeeze 阶段的输 入 B x F，即 batch size 乘以 F 维的向量特征，通过全连接压缩到 F/r 维，然后再加入一个全连接层，把 F/r 维重新放大到 F 维，再加上一个 sigmoid 得到一个 gate 权重，通过 x 乘上 gate 的权重得到最终的输出 Y，这样一方面减少了参数量，另一方面通过两层的全连接层，增加了它的非线性。

Chaining Moe：



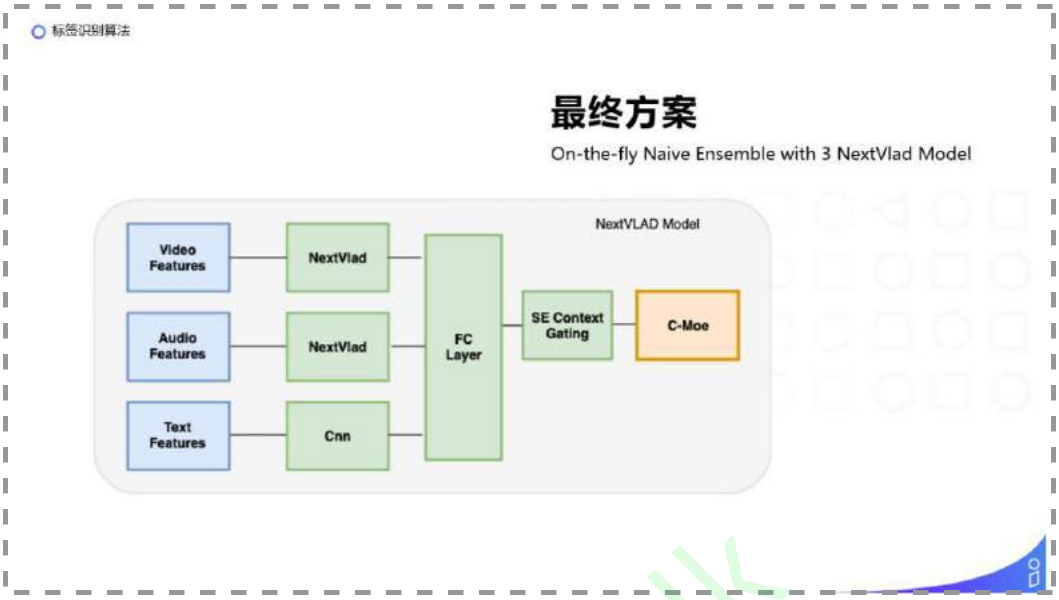
在分类阶段，使用 Chaining Moe 替代原始的 Moe，其网络结构如上图，每个 Chaining Unit 是把原始输入拼接上前面 Chaining Unit 的输出结果，再输入到 Moe 中。这样做相当于每次我们都选取了一批专家，每次在专家讨论结果之后，再重新选择一个专家团。一方面进一步提升了非线性表达能力，另一方面可以减少过拟合。

Knowledge Distillation with On-the-fly Navie Ensemble：



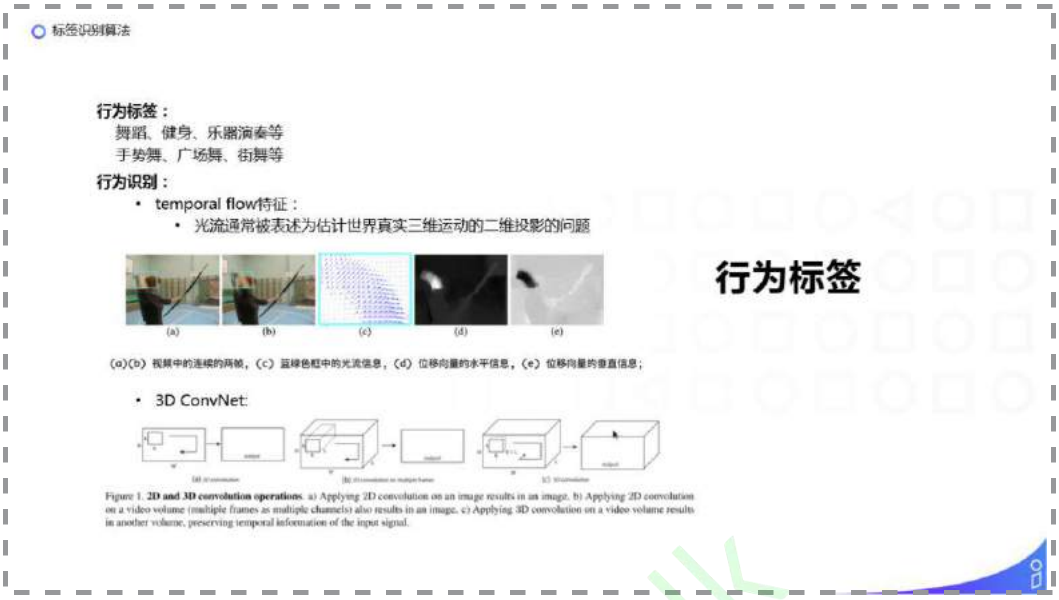
NextVlad 的作者还提出了使用知识蒸馏的方式来做 On-the-fly 的 Ensemble，Mixture Prediction 相当于一个老师模型，上面的 3 个 Prediction 相当于 3 个学生模型，老师模型是通过原始输入求平均之后，训练一个门函数得到每一个学生模型的权重，然后再进行线性加和，得到最终老师模型的输出。老师模型训练的 loss 是原始的视频分类的交叉熵损失函数。每个学生模型是在原始的交叉熵损失函数的基础上加上了老师预测模型的结果，得到的一个 KL 散度 loss 来作为每个学生模型的损失函数。这样做是为了让简单的模型通过学习复杂老师模型的预测结果，使用比较少的参数量也能训练得到比较好的效果。

最终方案：



我们最终的线上方案使用了知识蒸馏，用了 3 个 NextVlad 的子模型，每个子模型的结构如上图：对视频和音频的特征分别使用 NextVlad 进行提取，然后对于文本的特征使用 CNN 提取，再把提取到的多模态特征拼接起来输入到一个全连接层，然后输入到 SE Context Gating 中，最后输入到 C-Moe 中训练分类。

2. 行为标签



我们可以看到前面的 NextVlad 模型没有用老视频的前后帧的顺序，一个时空的特征，很多行为是跟时间发生的先后顺序是比较相关的，比如跳起和落下，根据出现的顺序，会是两个完全不同的动作。我们在行为识别上引入了光流特征（可以理解为把 3D 的运动投影到 2D 的图片上），图中的 (a) (b) 是视频中的连续的两帧，(c) 是计算出来的光流信息，为了使用光流信息，我们会把它储存为一个水平的图片和一个垂直的图片，分别包含水平和垂直维度的向量，这样得到的两张图片可以用传统的图片模型进行训练。

3D ConvNet：

另外一个方案，使用图片的时空特征，通过 3D 卷积（在 2D 卷积的基础上，feature 又增加了一个维度，除了水平和垂直外又增加一个时空维度，它的窗口除了在水平和垂直上进行滑动，也会在时空上滑动）进行训练。

Architecture	UCF-101			HMDB-51			Kinetics		
	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow
(a) LSTM	81.0	–	–	36.0	–	–	63.3	–	–
(b) 3D-ConvNet	51.6	–	–	24.3	–	–	56.1	–	–
(c) Two-Stream	83.6	85.6	91.2	43.2	56.3	58.3	62.2	52.4	65.6
(d) 3D-Fused	83.2	85.8	89.3	49.2	55.5	56.8	–	–	67.2
(e) Two-Stream I3D	**84.5**	**90.6**	**93.4**	**49.8**	**61.9**	**66.4**	**71.1**	**63.4**	**74.2**

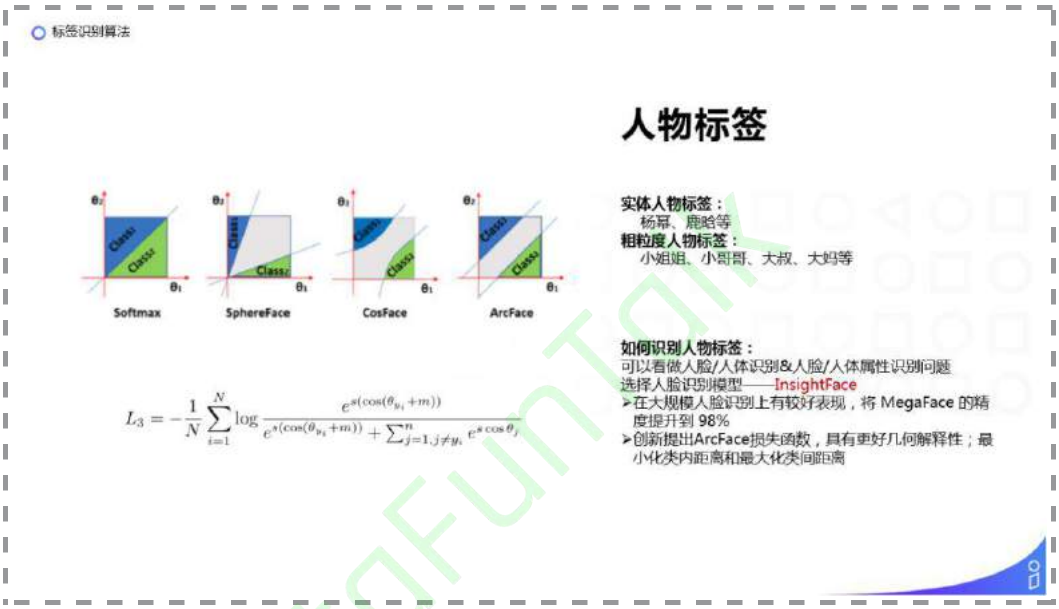
Table 2. Architecture comparison: (left) training and testing on split 1 of UCF-101; (middle) training and testing on split 1 of HMDB-51; (right) training and testing on Kinetics. All models are based on ImageNet pre-trained Inception-v1, except 3D-ConvNet, a C3D-like [ ] model which has a custom architecture and was trained here from scratch. Note that the Two-Stream architecture numbers on individual RGB and Flow streams can be interpreted as a simple baseline which applies a ConvNet independently on 25 uniformly sampled frames then averages the predictions.

上图为目前比较主流的行为识别模型，有用到 3D 卷积网络，另外一个比较主流的是把每张图片对应的光流信息拼接起来，然后输入到卷积网络，再合并起来预测动作，我们使用的是 (e) I3D 模型，把每张图片对应的光流信息依次排列分别输入到 3D 卷积网络中，然后把 3D 卷积网络训练的特征拼接在一起预测行为。I3D 模型的主要贡献为：

- 提出了 Kinetic 大规模数据集及预训练模型，可以在视频数据比较少的情况下，训练得到一个不错效果；
- 通过把 2D ConvNet 扩充成 3D 的 ConvNet，可以在参数初始化的时候用原来在 ImageNet 上预训练的参数做初始化；
- 支持 rgb 和 optical flow 分开预测，再合并分数。

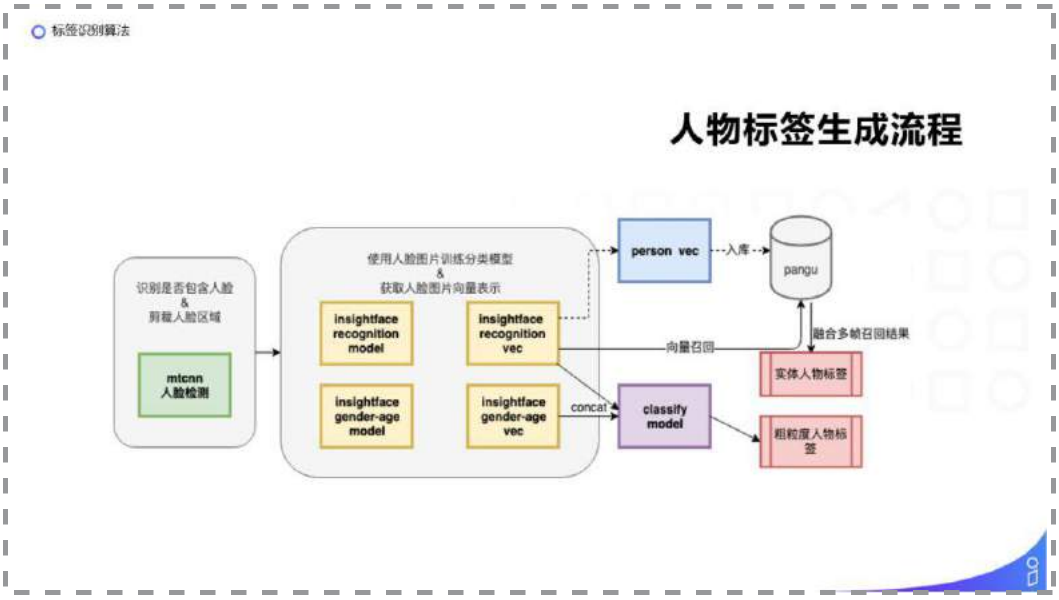
图中还对比了几种行为识别方法以及最终得到的效果，可以看到 I3D 模型在几个公开的数据集上都有比较好的表现。

3. 人物标签



人物标签比较不同的地方，根据人物标签不同的粒度，需要采取不同的方法进行识别。人物标签的识别，可以理解成人脸或者人体的识别问题，我们在现有的模型中，选择了 InsightFace 模型：

- 在大规模人脸识别上有较好表现，将 MegaFace 的精度提升到 98%；
- 创新提出 ArcFace 损失函数，在 Softmax 以及之前在人脸识别领域对 loss 做优化的 SphereFace 和 CosFace 基础上，进一步优化了分类平面，具有更好的几何解释性，使类间的距离是更大的，类内距离是更小的。



人物标签生成的流程：

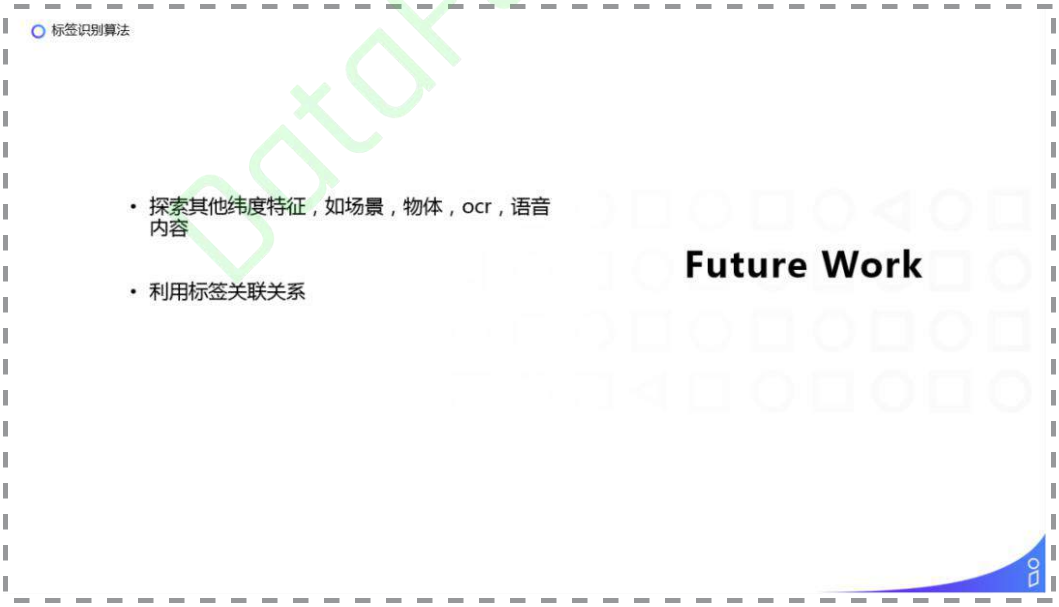
① 首先会用 mtcnn 检测视频帧中是否存在人脸，并且通过 mtcnn 把人脸区域剪裁出来，然后把检测出来的人脸输入到 Insightface 训练的人脸识别模型和 Insightface 训练的年龄性别模型中分别提取全连接层的向量。

② 对于实体人物标签我们会对需要识别的具体某一个人使用一批他的人脸图片得到一批他的人脸向量，通过计算得到这个人平均脸的向量，再把这个向量入库到 pangu 中。当有一个新的视频过来的时候，我们对单帧图片识别人脸并计算得到人脸向量，召回 pangu 中最相似的人脸向量，如果召回的人脸向量相似度满足阈值，我们认为图片中出现了对应的人物。

③ 我们对视频会采取多帧，对多帧分别预测，再通过融合来得到最终这个视频里出现的人物。

④ 对于年龄和性别这样的粗粒度的标签，我们会把通过 InsightFace 训练的人脸向量和年龄性别的向量拼接起来，然后把他们输入到一个分类模型中分类，得到他具体是某一个类型的人物标签。

未来的工作

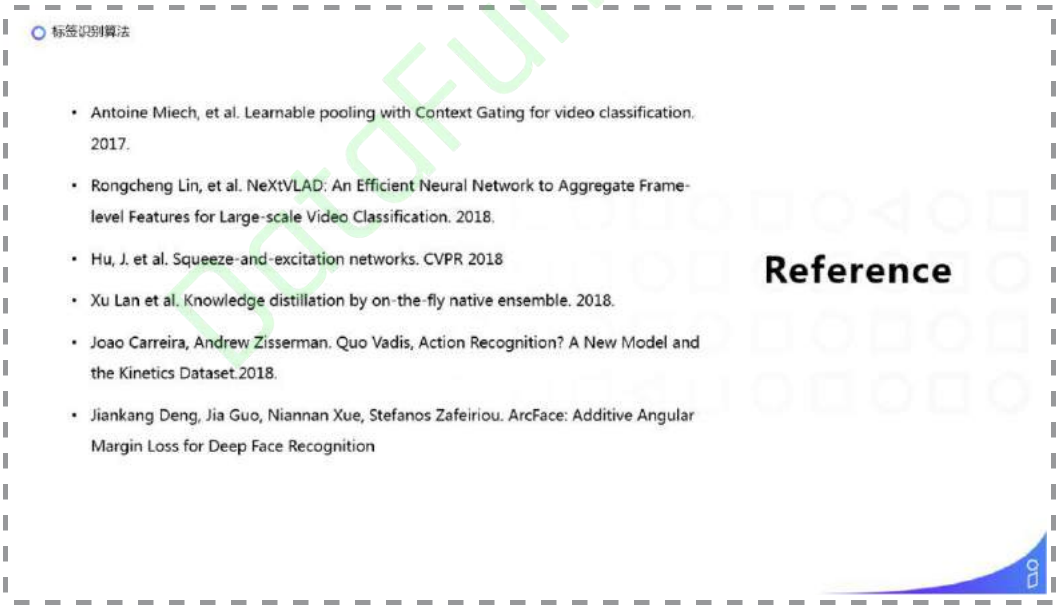


我们目前还是有很多特征没有使用到，比如视频的一些场景特征和物体检测的特征，包括有些视频它的核心内容体现在视频中的 ocr 或语音上，这些是未来我们打算去尝试的特征。

利用更多的标签的关联关系，优化我们目前的打标结果。

参考资料

这是我们工作中，参考的一些论文，大家感兴趣的话，可以去看一下，今天的分享就到这里，谢谢大家。



UC 信息流推荐模型在多目标和模型优化方面的进展

嘉宾：语露　编辑：Hoh Xil

导读：短视频已经成为信息流行业的风口，成为拉动规模增长的主要驱动力。短视频天然具有信息能量高、用户粘性大、内容丰富等优点，也有视频帧内容难以分析提取、结构化的缺点。如何提高短视频的分发效率和推荐精准度，做到千人千面的个性化推荐，是一个推荐系统的核心能力。基于深度学习的推荐模型，是业界前沿的研究课题；在短视频推荐的业务中，如何利用深度学习算法有效的提升消费点击和消费时长，是推荐模型的核心命题。此次演讲主要讲解视频推荐模型在多目标和模型优化方面的进展。包括以下几个模块：

- 业务和系统
- 基于 Graph Embedding 的多目标
- 基于 WnD 的 Boosting 算法
- 未来规划

业务和系统

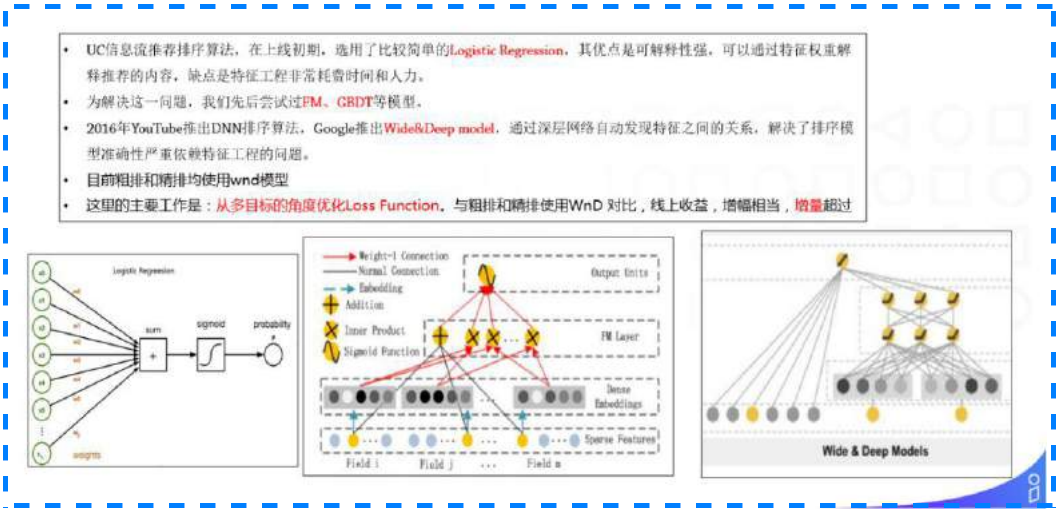
1. 短视频业务背景、系统结构

我们的业务主要是做视频推荐，嵌入到 UC 浏览器中做国内信息流：

- UC 从工具型产品转型至内容分发平台，信息流业务成为 UC 的第二大引擎；
- 短视频已经成为信息流行业的风口，成为拉动规模增长的主要驱动力；
- 短视频播放时长已经超过图文，极大的增强了信息流的用户粘性；
- 算法的持续优化与迭代提高了短视频流量分发的效率和准确度。

右图是视频推荐的界面，其背后的视频推荐系统分为三个结构，第一个是召回模块，第二个是粗排的模型，第三个是精排的模型。从召回 -> 粗排 -> 精排，Item 的数目从多到少，推荐的准确性从低到高。

2. 技术演进史

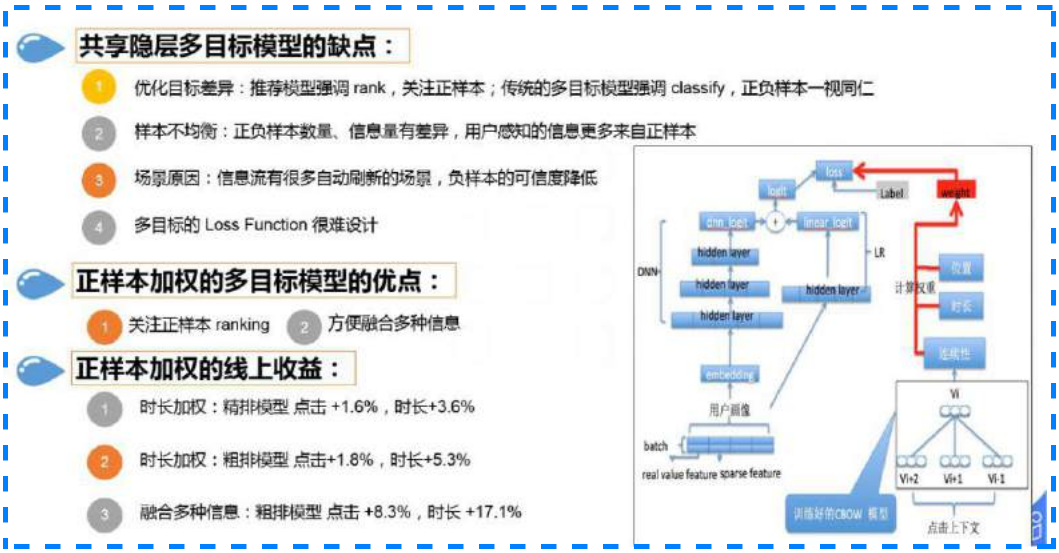


我们在视频推荐上，尝试过多个推荐方案：

- 最早用的是 LR（ 如图，左下角 ），LR 的缺点主要是在特征工程上会耗费很大的人力；
- 后来我们又尝试了 GBDT 和 FM（ 如图，中间部分 ），但是这些模型在泛化性上相对较弱；
- 最后我们采用了 Wide & Deep model（ 如图，右下角 ）。

这里主要介绍的工作是：从多目标的角度优化 Loss Function。与粗排和精排使用的 WnD 对比，线上收益增幅相当，增量超过。

3. 多目标在视频推荐模型中的应用



常规的多目标是共享隐层的 soft 多目标模型 ,我们使用的是基于正样本加权的模型。

共享隐层多目标模型的缺点：

① 优化目标差异：推荐模型强调 rank，关注正样本排序；传统的多目标模型强调 classify 分类，正负样本一视同仁；

② 样本不均衡：在信息流中正负样本数量、信息量存在差异，用户感知的信息更多来自正样本；

③ 场景原因：信息流有很多自动刷新的场景，负样本的可信度降低

④ 多目标的 Loss Function 很难设计，市面上的多目标方案大多是一个主目标和一个辅助的目标

我们使用的正样本加权的多目标模型的优点：

① 关注正样本 ranking

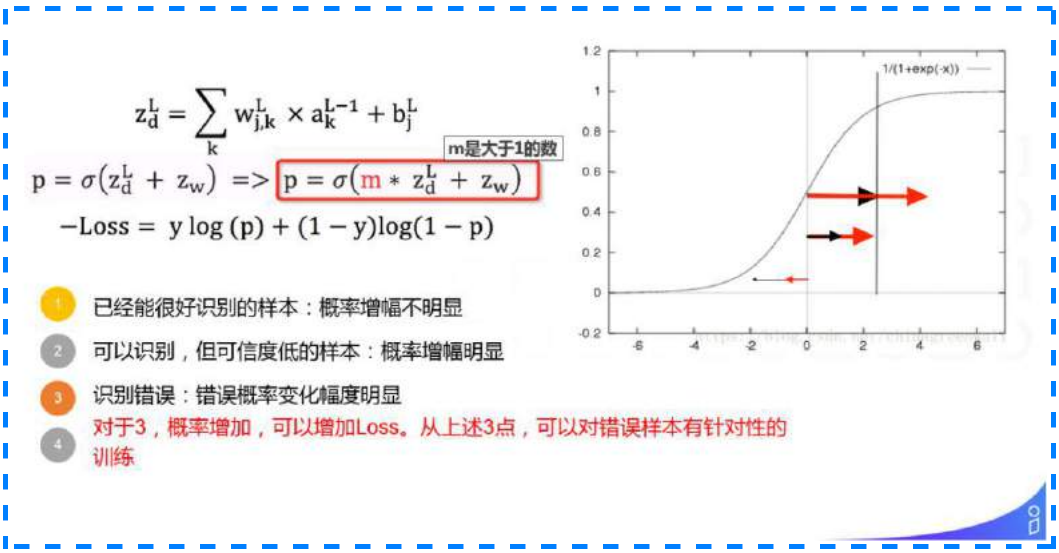
② 方便融合多种信息

我们的方案最终在线上取得了不错的效果。

基于 Graph Embedding 多目标融合的 WnD 模型优化

1. 优化 WnD 模型 logit 配比

信号正向传播



首先是 WnD 模型的一个小的优化，我们知道 WnD 分为 LR 侧和 DNN 侧，LR 侧使用的优化器是 FTRL，它的优化速度非常快，在数千万 DAU 的

场景下，只需采用一次流式训练，在这样的场景下 DNN 对样本的训练很难迭代多次，这时 WnD 模型更倾向于使用 LR 侧的 logit。基于这样的原因，我们想能不能增加 DNN 侧的 logit，提高系统的准确度，大体就是这样的 motivation。

具体实施的时候，见图中红框部分，d 指 DNN，w 指 LR，z_d^L 指 DNN 最后一个隐层的输入，z_w 指 LR 最后一个隐层的输入，我们在 DNN 侧乘以了一个系数 m，这就是大体的一个优化方向：

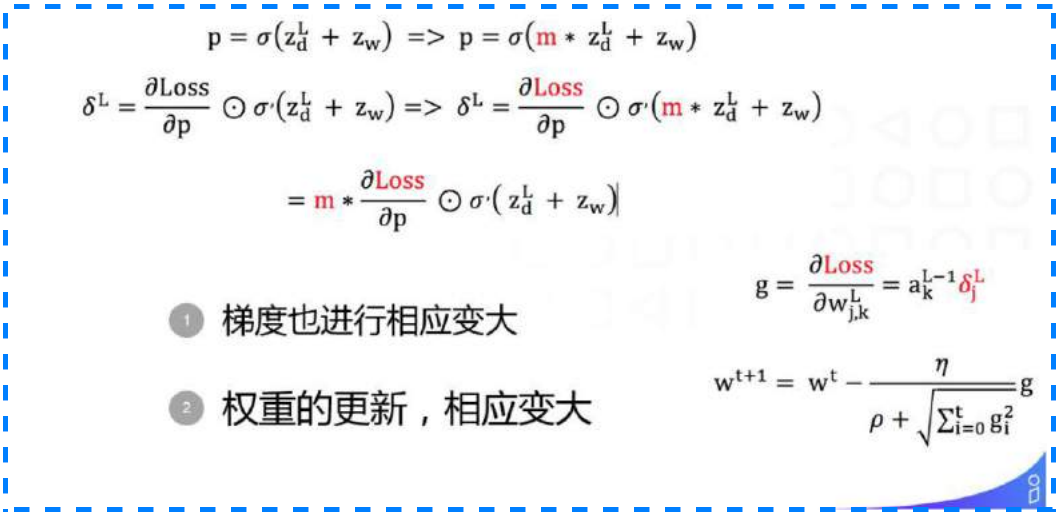
① 对于已经能很好识别的样本：概率增幅不明显（如右图最上部分的箭头，已经进入饱和区，所以增加不明显）

② 对于可以识别，但可信度低的样本：概率增幅明显（如右图中间部分的箭头）

③ 识别错误：错误概率变化幅度明显（如右图最下部分的箭头）

④ 对于 ③，概率增加，可以增加 Loss，从上述 3 点，可以对错误样本有针对性的训练

梯度反向传播、Adagrad 优化器



这是一个反向传播公式的推导，上面讲过增加 DNN 侧的 logit 会增加 Loss，图中标红的表示 Loss 会增大，同时，m 可以提出来作用到 Adagrad 优化器时：

$$g = \frac{\partial Loss}{\partial w_{j,k}^L} = a_k^{L-1}\delta_j^L$$

也是相应增加的。

2. 基于时长加权的多目标融合的 WnD 模型优化

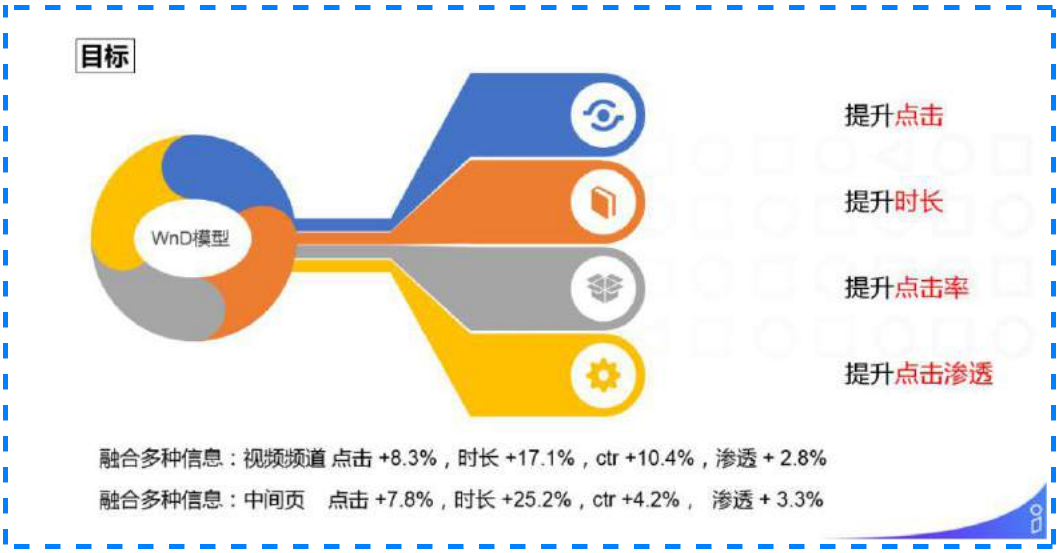
这部分的 motivation 是我们在做模型训练时，做的还是二分类的一个分类模型，这时点击观看的时长没有用到。因此，我们利用用户观看时长，来提升推荐效果。

这里设计了一个 weigh 计算方式：对于某个视频用 u_play_len 这个视频的历史观看总时长 / avg_item_history_view_len 这个视频历史的平均观看时长，用 max 做了一个规范化，求得 weigh，且 weigh > 1。图中统计了右边视频随着时长增加，用户观看数量的一个变化情况。有了 weigh 我们可以对 WnD loss function reweight：

① 如果某用户观看 Item 时长低于历史平均时长，权重为 1，那么原来的既得收益（ctr）不会变化。

② 如果某用户观看 Item 时长高于历史平均时长，权重为 1.x，那么此类视频加权充分训练，会被 Ranking 比较高的分数，增加时长收益。

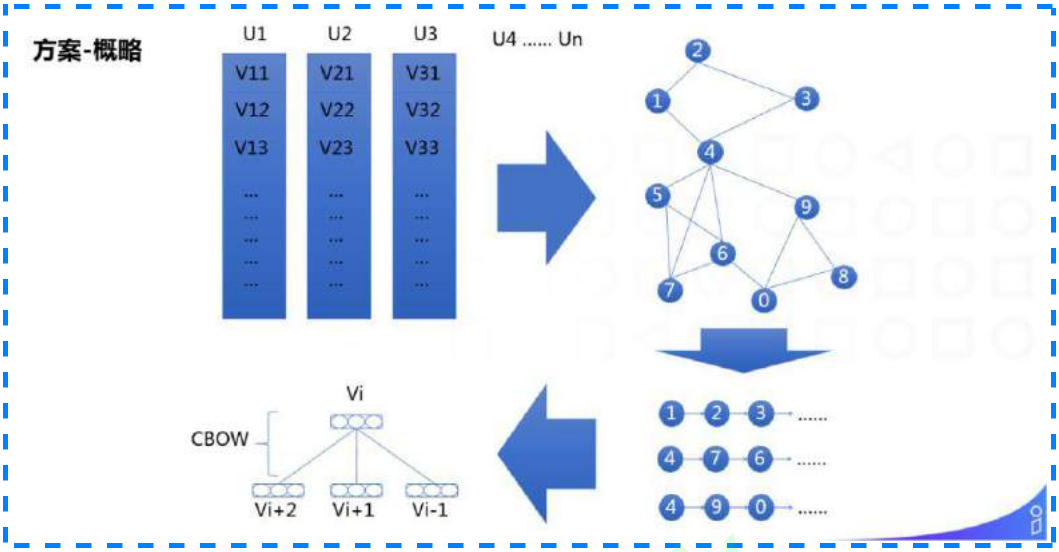
3. 基于 Graph Embedding 多目标融合的 WnD 模型优化



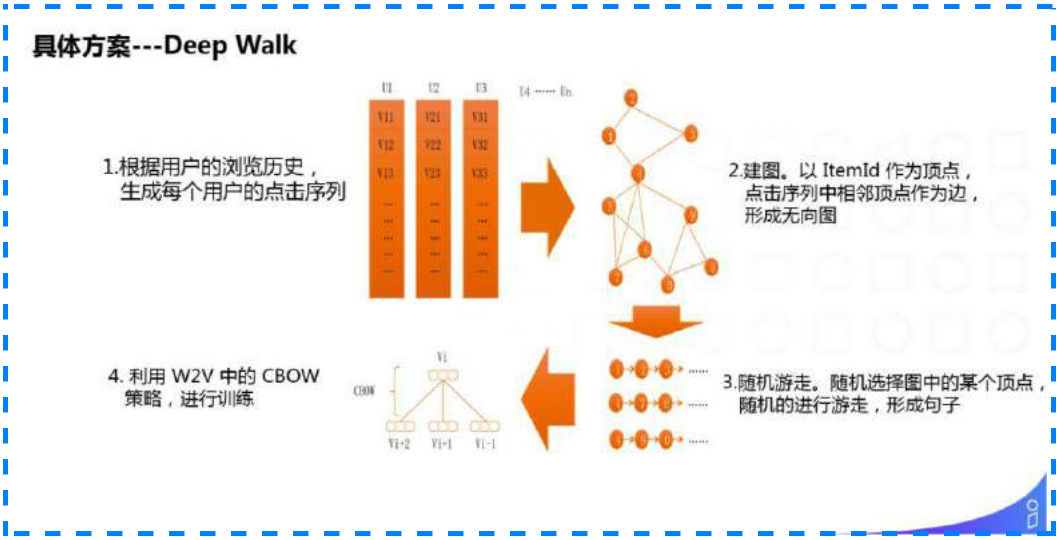
基于 Graph Embedding 多目标融合的方案中，多目标包括：提升点击、提升时长、提升点击率、提升点击渗透。

首先如何取到一个 Item Embedding？我们采用的是 Deep Walk 算法，在拓扑网络中，通过随机游走和 w2v 算法，学习网络中顶点的低维向量表示，向量中蕴含着顶点之间的相关性等其他特征。真正训练的时候采用的是 CBOW 训练策略，其 Loss Function 如下：

$$\mathcal{L} = \sum_{w \in C} \log\ p(w|context(w))$$



我们的方案是，从用户日志中挖掘出每个 User 的点击序列，根据点击序列建立无向图，然后在每个节点随机游走形成句子，最后经过 w2v 的训练策略，得到每个 Item 的 Embedding 表示。



具体方案：

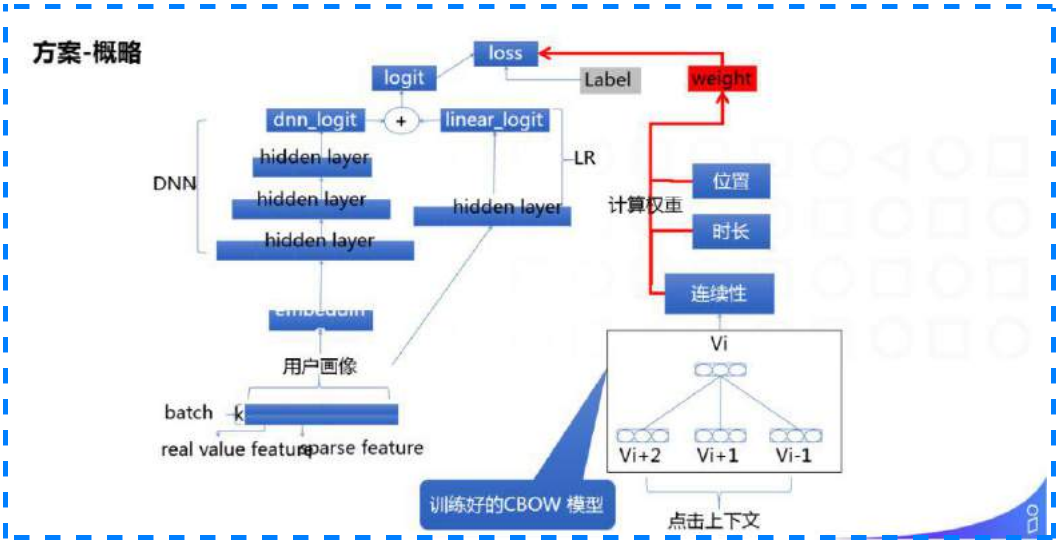
① 根据用户浏览历史，生成每个用户的点击序列；

② 建图，以 ItemId 作为顶点，点击序列中相邻顶点作为边，形成无向图；

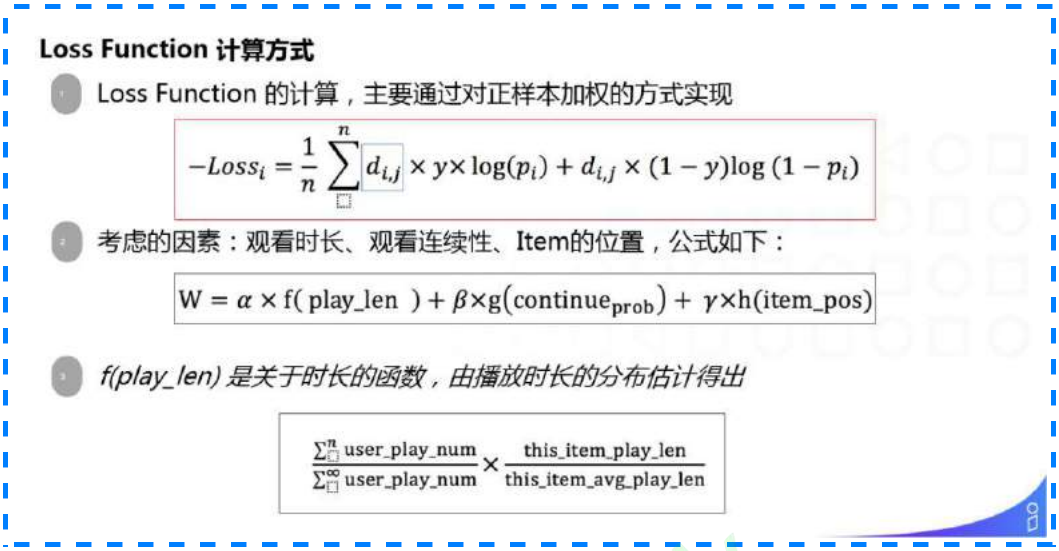
③ 随机游走，随机选择图中的某个顶点，随机的进行游走，形成句子；

④ 利用 w2v 中的 CBOW 策略，进行训练。

DataFunTalk



有了 Item Embedding 之后，做 Loss Function 的 reweight，reweight 时加入了位置、时长、连续性（ 通过 Graph Embedding 计算得出 ）等信息。



Loss Function reweight 方式是在正样本前面乘一个系数 $d_{i,j}$，然后负样本保持不变，如果拿掉系数，是一个标准的交叉熵公式，$d_{i,j}$ 考虑了观看时长、

DataFunTalk

观看连续性、Item 的位置等综合信息来得到的，它们之间用的是 α、β、γ 线性加权得到的：

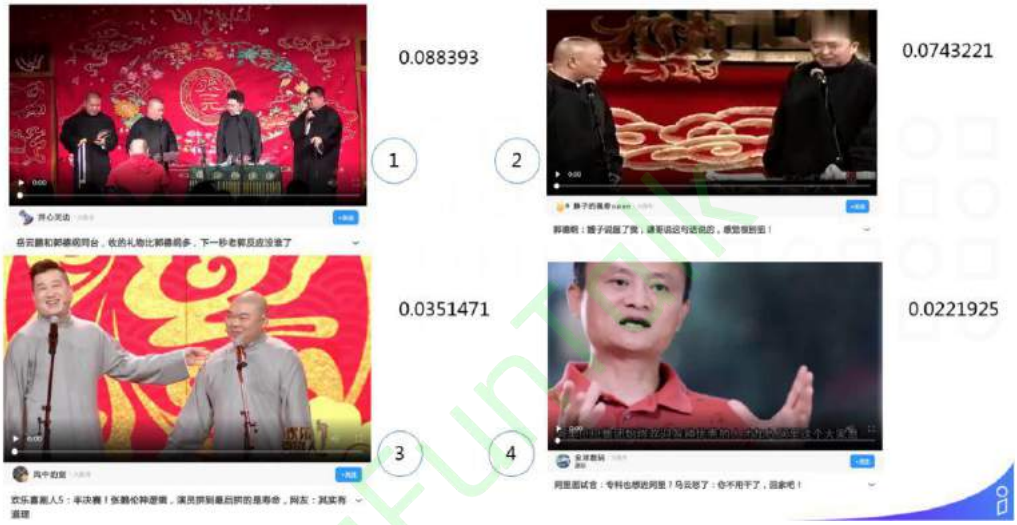
$$W = \alpha \times f(\text{play_len}) + \beta \times g(\text{continue}_{\text{prob}}) + \gamma \times h(\text{item_pos})$$

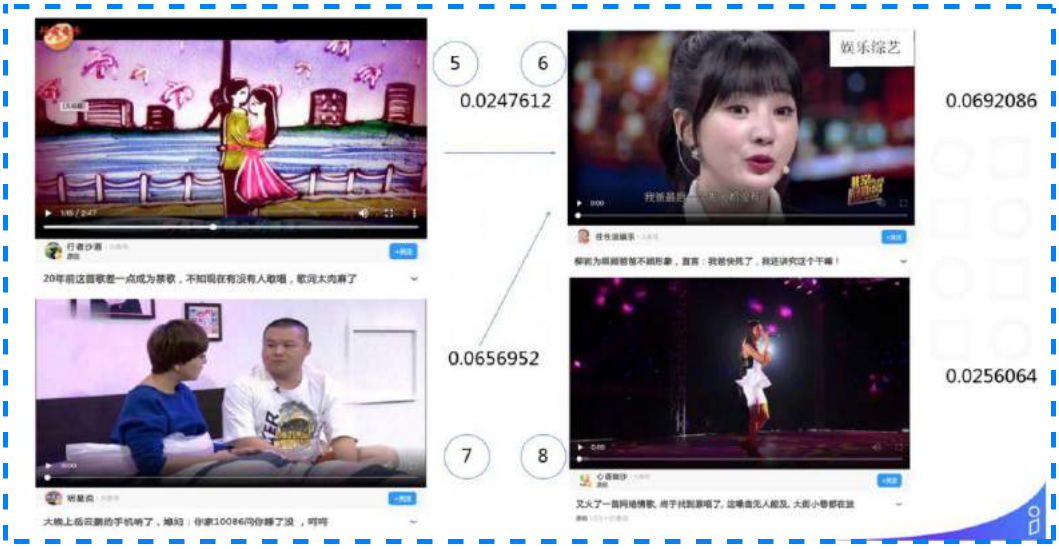
f（play_len）是关于时长的函数，由播放时长的分布估计得出：

$$\frac{\sum_{\square}^{n} \text{user_play_num}}{\sum_{\square}^{\infty} \text{user_play_num}} \times \frac{\text{this_item_play_len}}{\text{this_item_avg_play_len}}$$

左下角为用户播放数的累加，它的上面是观看到某一时长 n 的用户播放数的累加，右边侧延续了时长加权公式的模板。

4. 连续性





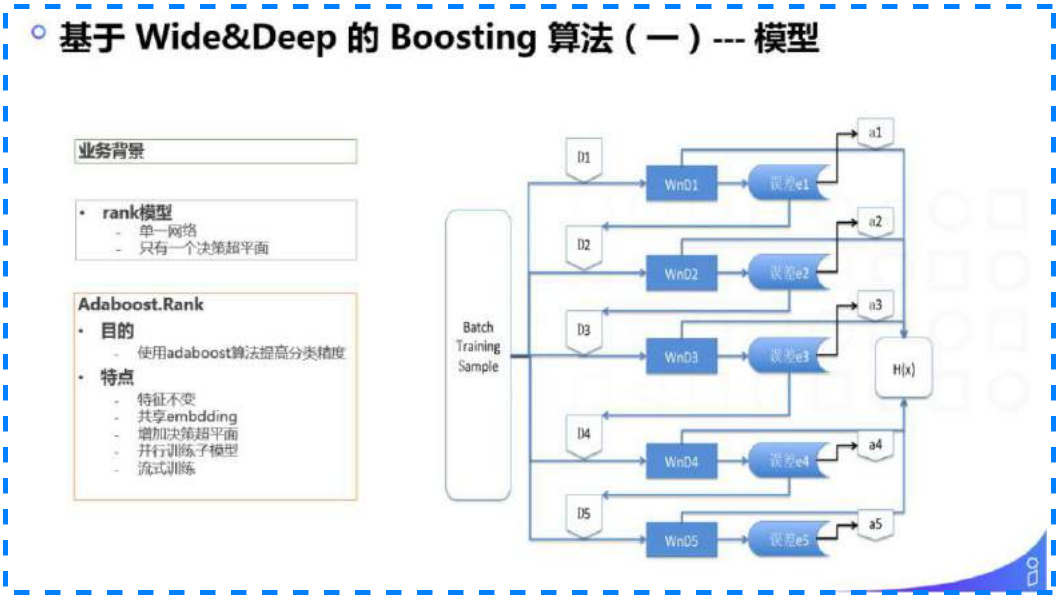
接下来讲一个连续性的实际案例：

①②③④⑤⑥⑦⑧这是用户的点击顺序，对应的分数是基于连续性 w2v 算法计算出来的分数，我们认为分数较高的视频很容易带领用户继续看下去，把它称作一个连续性的信息，我们对此类信息进行加权，希望能够带动用户整体的消费。

基于 WnD 的 Boosting 算法

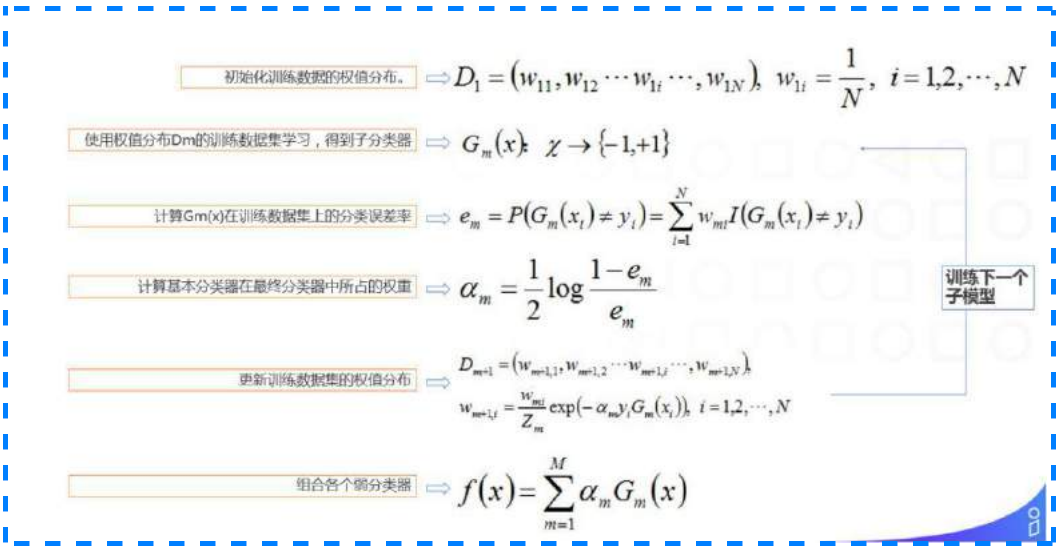
1. 模型

DataFunTalk



由于 Wide&Deep 建模方案是一个二分类，理论上只有一个分类超平面，很容造成样本错分。我们理想的方案是，虽然是二分类，但是我们可以有多个决策超平面，所以我们用了 Boosting 算法，综合 5 个 Wide&Deep，进行二分类。其应用场景是训练的实时流，每次训练都是一个 Batch（相当于在线训练），中间融合了 5 个 Wide&Deep 模型（5 个模型的特征是不变的，共享 Embedding 隐层，采用的是并行计算），最终生成一个强分类器。

Adaboost：



这是 Adaboost 算法的标准模式：

① 初始化训练数据的权值分布

② 使用权值分布 Dm 的训练数据集学习，得到子分类器 Gm(x)

③ 计算 Gm(x) 在训练数据集上的分类误差率

④ 利用分类误差率计算基本分类器在最终分类器中所占的权重

⑤ 更新每个样本的权重，继续训练下一个模型，如果有 5 个子模型就训练 5 个子模型

⑥ 最终综合每个子模型的权重和预测值，得到一个强分类器

2. 算法

定义 i 为子模型的下标，εi 表示第 i 个WnD 模型误差， Pi 是第 i 个 WnD 模型的输出概率， Di 是第 i 个WnD模型样本的权重 Di={d1, d2,d3,...dn}，Di 集合样本的个数与当前训练 Batch 的样本个数相同。第 i 个Wnd 模型的误差:

$$\varepsilon_i = 1.0 - AUC(Label,\ P_i,\ D_i)$$

定义 t 为训练 Batch 的时序下标。第 i 个Wnd 模型的权重：

$$a_{t,i} = \frac{1}{2}a_{t-1,i} + \frac{1}{2}\log\left(\frac{1-\varepsilon_{t,i}}{\varepsilon_{t,i}}\right)$$

具体的算法有几个不同点：

第一个是误差项的计算方式，原先标准 Adaboost 算法误差项计算用的是加权指标函数：

$$e_m = \sum_{i=1}^{N} w_{m,i} I(G_m(x_i) \neq y_i)$$

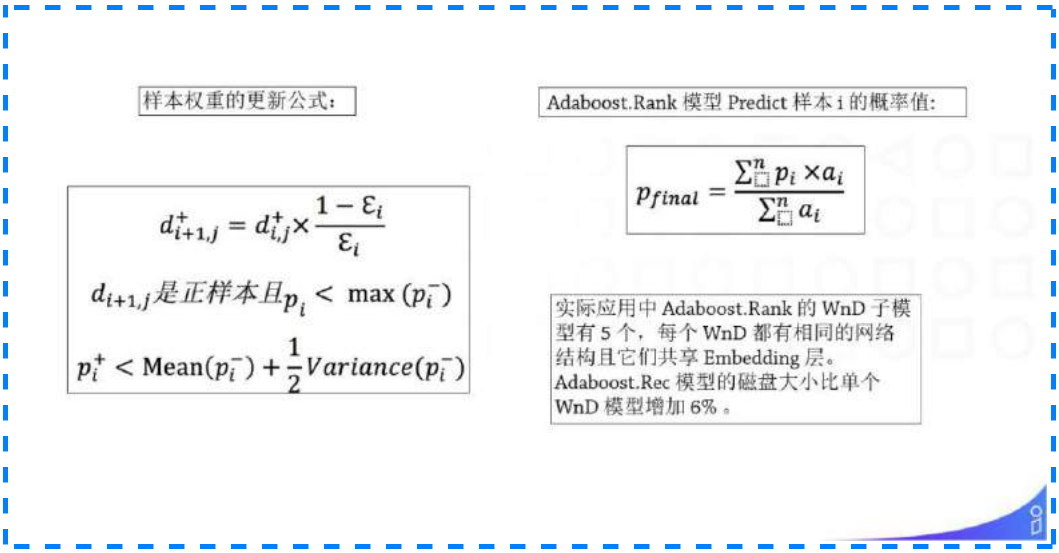
我们的方案采用的是 AUC 加权的误差项：

$$\varepsilon_i = 1.0 - AUC(Label,\ P_i,\ D_i)$$

每个子模型的权重，由于是在线训练，所以采用的是迭代的方式：

$$a_{t,i} = \frac{1}{2}a_{t-1,i} + \frac{1}{2}\log\left(\frac{1-\varepsilon_{t,i}}{\varepsilon_{t,i}}\right)$$

每次用上一个 Batch 1/2 权重加上此次误差项的计算。



第二个是样本权重的更新，样本权重更新时只对正样本进行更新，而标准的 Adaboost 是对正负样本都进行更新的：

$$d_{i+1,j}^{+} = d_{i,j}^{+} \times \frac{1-\varepsilon_i}{\varepsilon_i}$$

$d_{i,j}^{+}$ 指的是这个样本上一个子模型的权重，ε_i 是误差项。

$$d_{i+1,j}\text{是正样本且}p_i < \max(p_i^-)$$
$$p_i^+ < \mathrm{Mean}(p_i^-) + \frac{1}{2}Variance(p_i^-)$$

这是样本的更新方式。

最终强分类器的预测和 Adaboost 标准模型差不多：

$$p_{final} = \frac{\sum_{\square}^{n} p_i \times a_i}{\sum_{\square}^{n} a_i}$$

也是线性加权然后除以总的权重。

实际应用中 Adaboost.Rank 的 WnD 子模型有 5 个，每个 WnD 都有相同的网络结构且它们共享 Embedding 层，Adaboost.Rec 模型的磁盘大小只比单个 WnD 模型增加了 6%。

3. 在短视频推荐中的应用

方案的实验效果和 WnD 模型对比如上，提升了模型推荐的精准性。

未来规划

未来的规划：

- 现在的推荐都是基于 point wise 的推荐模型，每次都是预测单个 Item 的分数，未来希望可以一次给用户推荐一刷的结果。

- 因为每天都有新用户进来，所以我们希望建立一个留存新用户模型，来留住新用户。

- UC 信息流目前的主要消费群体为男性用户，在推荐的偏好上模型和策略都偏向男性用户，我们希望可以留住更多的女性用户。

这就是未来的一个方向，谢谢大家。

DataFunTalk

深度学习技术在美图个性化推荐的应用实践

作者：陈文强、白杨、黄海勇　编辑：王吉东

导读：美图秀秀社交化的推进过程中，沉淀了海量的优质内容和丰富的用户行为。推荐算法连接内容消费者和生产者，在促进平台的繁荣方面有着非常大的价值。本次分享探讨美图在内容社区推荐场景下应用深度学习技术提升点击率、关注转化率和人均时长等多目标的实践经验。

1. 美图社区个性化推荐场景概况与挑战

2. embedding 技术在召回阶段的应用实践

基于 Item2vec 的 item embedding

YouTubeNet 和双塔 DNN 在个性化深度召回模型应用实践

3. 美图排序模型的研发落地

NFwFM 模型研发迭代历程和经验

多任务学习（Multi-task NFwFM）在多目标预估场景的探索与实践

美图社区个性化推荐场景与挑战

1. 业务场景

美图社区个性化推荐场景大大小小有十多个，其中流量比较大的场景是美图秀秀 app 的社区内容推荐 tab（ 图 1），这个场景以双列瀑布流的形态给用户推荐他最感兴趣的内容。

当用户点击感兴趣的图片后会进入图 1-2 的相似推荐 feeds 流场景。在这个场景下，用户消费的图片和视频 ,都是和用户刚刚点击进来图片是具有多种相似性的，如视觉、文本、topic 等。而如果用户是从双列瀑布流里点击视频，则会进入到图 1-3 的视频 feeds 流场景。这个场景主打让用户有沉浸式的消费体验。以上是美图社区内容推荐的主要业务场景。

图1. 美图秀秀社区内容推荐tab　图2. 美图社区相似推荐feeds流　图3. 美图社区视频feeds流

2. 工作目标

个性化推荐的首要目标是理解内容。从内容本身的视觉、文本以及特定场景下用户的行为来理解社区里可用于推荐的内容。接下来是理解用户，通过用户的基础画像（ 年龄，性别等 ）、设备画像（OS，机型等 ），以及用户的历史行为来挖掘其兴趣偏好。

再理解了社区的内容和用户之后，才是通过大规模的机器学习算法进行精准推荐，千人千面地连接用户与内容，从而持续提升用户体验，促进社区繁荣。

3. 挑战

在实际生产实践过程当中，主要遇到挑战如下：

海量

美图内容社区有月活超过 1 亿的用户，每天有 100 多万的候选图片和视频。在这种数据规模下，只在小数据规模下验证有效的复杂算法模型无法直接应用到工业界线上系统。

实时

算法需要在端到端小于 300ms 的时延里，每天处理超过 3 亿的个性化排序请求。这要求召回和排序算法不能过于复杂，要能够进行高效的计算。

长尾

在实际场景中，用户分布以及 item 行为分布都是长尾的：

- 用户分布的长尾性：新用户占比超过 27%；
- 曝光分布的长尾性：关注<< 点击<<曝光。

在这样长尾数据上进行预估要求我们的模型具备稳定的泛化能力。

美图深度学习技术栈——召回端

在上述的应用场景和技术挑战下，美图是如何将深度学习应用到个性化推荐中的召回端和排序端的呢？我将会在下面为大家一一介绍。

召回端的 Item embedding 技术和召回模型，用于从百万级别的候选集里挑选 TOP 500 个用户最感兴趣的候选集。相对于召回端，排序端的深度排序模型能够融合多种召回来源并进行统一排序，排序模型能包容规模更大的细粒度特征，相对召回模型，排序模型能够实现更加精准的推荐。

美图目前部署在线上系统的召回技术主要包括 Item2vec，YouTubeNet，以及双塔 DNN。

1. Item2vec

Item2vec，是一种通过用户行为来理解内容的方式。

传统的理解内容方式是基于用户行为构造 item 侧的统计类特征，例如 item 的点击率，收藏率等。这些特征是非常有效的，但是对内容的理解维度比较单一。

另一种方式是从图片的本身的视觉来提取比如图片质量、清晰度、图片物体等等。还可以通过内容本身的文本特征，比如关键词，实体词等等，来帮助理解内容。这些维度的特征在内容冷启动中是很有效的。但是他们无法表达内容的某些潜在特性，比如某个内容是否给用户呈现出清新有趣的感觉。这种潜在的特征借助用户的行为来理解比较合适。Item2vec 正是这样一种技术，它基于短时间内被浏览的 item 具有内在相似性的假设来学习 item 的 embedding。

在图 4 中可以看到 item2vec 在美图社区图片上的部分效果。可以看到和查询图在训练数据中高频共现和中频共现的 item，在背景和主体人物上和查询词是高度相似的。而低频共现的部分和我们的查询图片有些差异，不过主体内容总体上还是比较相似的。

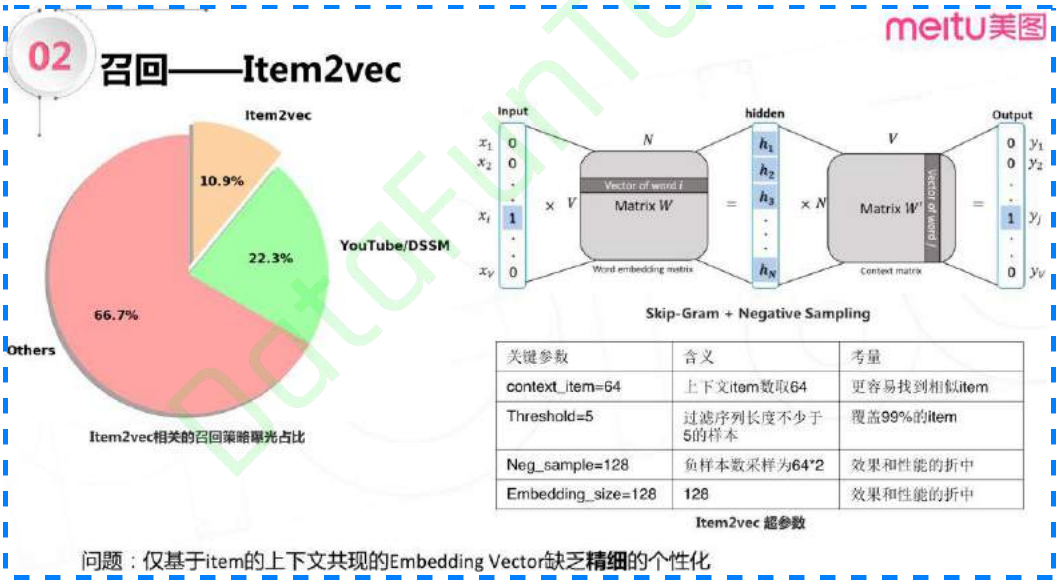
总体而言，item2vec 是一种学习 item embedding 的成熟方案。

Item2vec 学习出来的 item 向量是美图多种向量检索式召回策略的底层数据，包括实时兴趣，短期兴趣等等。比如当用户点击了某个 item，系统会实时地通过向量内积运算查询相似的 item 并插入到召回源头部，用于下一轮的排序。

使用 item2vec 学习出来的 item 向量作为底层数据的召回策略，在美图应用非常广泛，它们在整体曝光中占了 10%以上。

实际应用时，我们是基于 skip-gram+negative sampling 来做 item2vec 的。

它是一个只有一个隐层的深度学习模型。输入端是用户的点击序列，输出端是与输入端的 target item 邻近的 64 个 item。64 相对于 NLP 里取的 5-6 个是比较大的，这是因为用户的点击序列不像自然语言那样具有严格的局部空间句法结构。在比较大上下文窗口中，更容易找到和目标 item 相似的上下文 item，模型更容易学习。

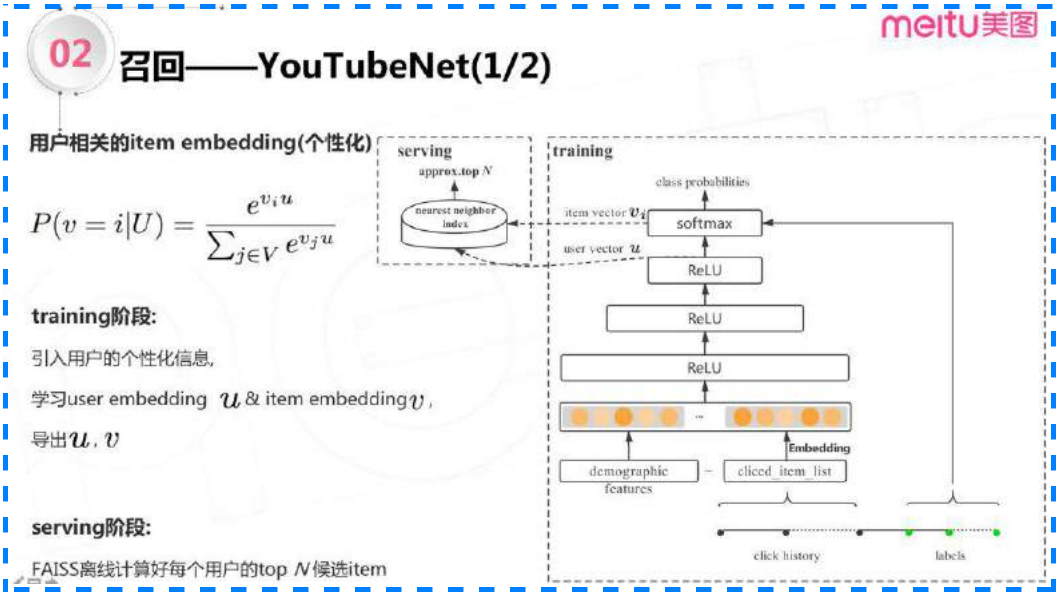


负采样的个数是正样本数的两倍，这是一个效果和性能折中，在我们的应用中正负样本数 1:2，能够在天级别模型更新情况下，取得比较好效果。隐层的 embedding size 取 128 可以得到比较好的效果，这个取值同样是效果和性能的折中。另外，我们过滤了点击序列长度小于 5 的样本，这样过滤之后，我们的点击序列能覆盖 99%候选图片和视频。

Item2vec 是学习 item embedding 向量的一种非常好的方案，它也覆盖了美图多个推荐业务。但是它不直接考虑用户的个性化行为，只考虑了训练样本中 item 与 item 之间的局部共现关系。如果要利用上丰富的用户侧特征，实现个性化的话，那么我们需要借鉴其他方案，而 YouTubeNet 正是这样一种业界成熟的方案。

2. YouTubeNet

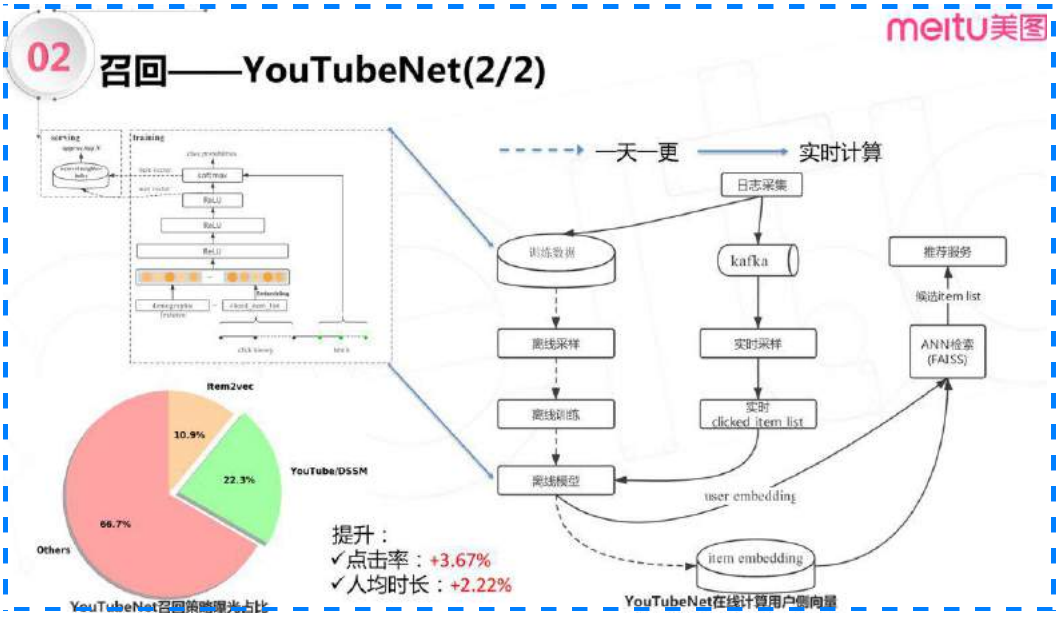
YouTubeNet 是 Google 于 2016 年提出的。与 item2vec 不同，YouTubeNet 在学习 item 向量的时候考虑了用户向量。从模型的优化目标上可以看出，是在给定用户向量的情况下，从候选池中筛选出该用户最感兴趣的 item 列表。



从上图右侧模型架构图可以看到，模型训练用的 label 是用户最近点击过的 item 列表，特征包括两部分，一部分是用户更早之前点击过的列表（clicked_item_list），另一部分是用户的 demographic 统计特征，如年龄、性别等。引入上述用户的个性化信息之后，模型通过学习 user embedding 和 item embedding，并离线导出用户向量和 item 向量。线上使用时利用 FAISS 工具离线计算好每个用户的 top N 候选 item 集，提供给排序算法使用。不过这种离线存储候选集的方式，不能实时应对用户不断变化的兴趣，要捕获这种变化，需要实时采集用户不断变化的点击 item 数据，实时计算用户侧向量。

实时计算用户侧向量的工作，一共分了两部分：第一部分是离线部分，为下图右侧的虚线部分，这里模型一天一更新。离线部分基本流程和上一段所述相同，模型训练完之后导出 item 向量并在 FAISS 中构建好索引。第二部分是实时部分，这一部分借助 kafka，实时采集用户点击行为数据并构建 clicked_item_list 特征，接着请求离线训练好的模型，计算出用户侧向量，最后从 FAISS 中查询的候选集，输出给排序服务。

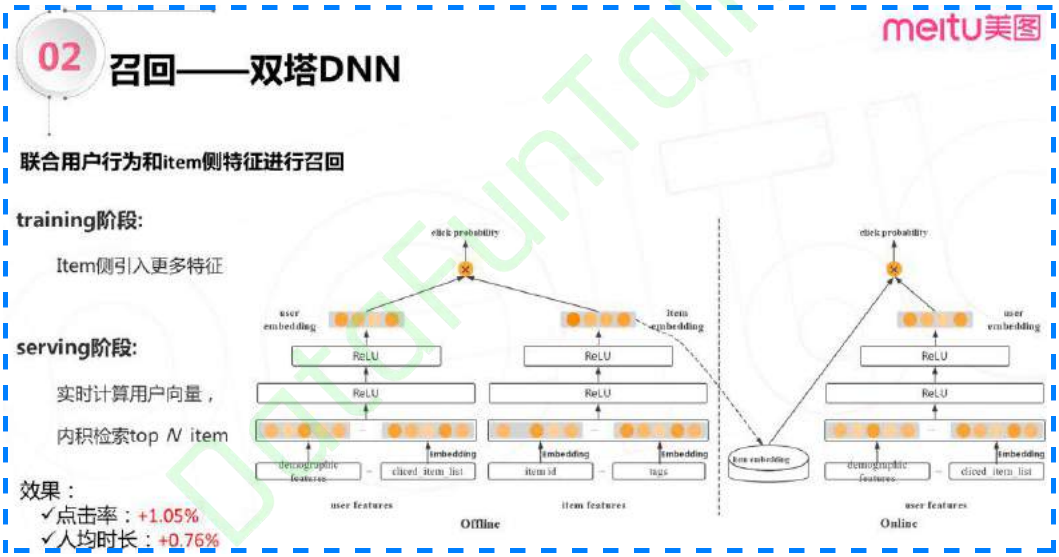
使用 YouTubeNet 模型实现实时计算用户侧向量之后，曝光占比 22%的 YouTubeNet 给整体带来了点击率 3.67%的提升，人均时长提升 2.22%。



3. 双塔 DNN

双塔 DNN 模型，联合用户行为和 item 侧特征进行召回。双塔 DNN 模型构建用户侧 embedding 的方式和 YouTubeNet 是一样的：先给用户的点击行为序列，年龄性别等稀疏的特征做 embedding，再经过几个简单的全连接层，得到用户侧向量。对于 item 侧特征，双塔 DNN 引入另一个子网络来学习，学习方式和用户侧特征是一样的。

离线训练完了之后和 YouTubeNet 还是一样，把 item 向量提前导出并加载到 FAISS。在线上环境使用的时候，实时计算用户侧向量，来快速捕获用户兴趣。引入 item 侧特征，使得线上用户点击率提升 1.05%，人均时长提升 0.76%。



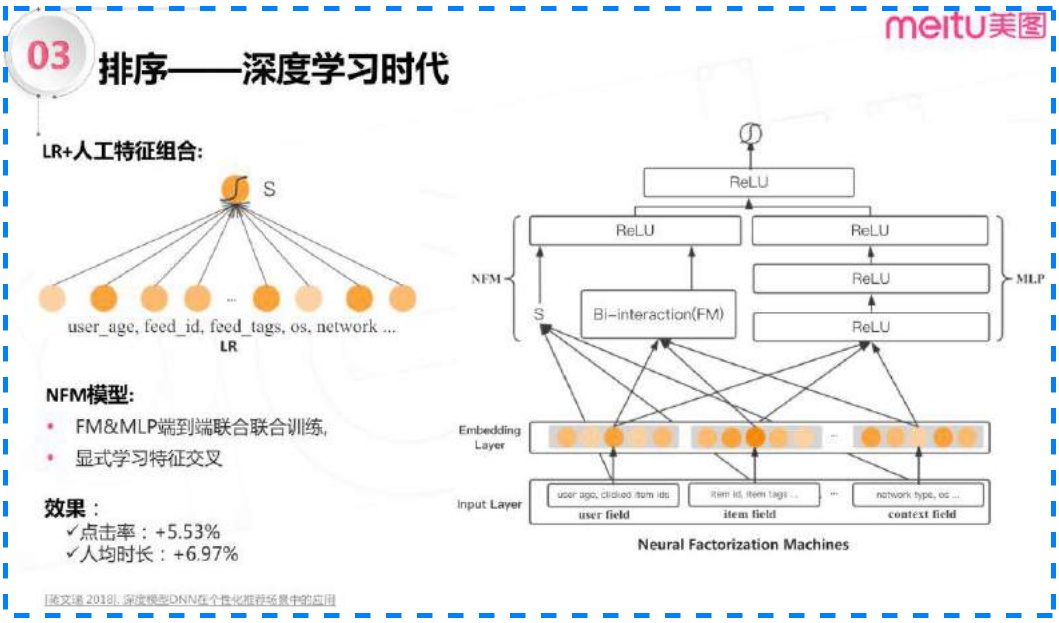
4. 总结

Item2vec 学习出来的 item 向量具有很好的相似性，作为底层数据，服务了多个召回策略，在美图具有非常广泛的应用。包括实时兴趣 ,短期兴趣等等，覆盖了 10%以上的曝光占比。YouTubeNet 和双塔 DNN 则分别引入用户侧和 item 侧特征，有监督地学习用户最感兴趣的 item 候选集，在美图个性化推荐召回层，累计点击率提升了 4.72%，时长提升了 2.98%。

美图深度学习技术栈——排序端

1. 重新审视 NFM 模型

美图的第一代模型主打 LR 为主+人工特征组合。随着业务发展，大大小小的推荐场景越来越多，做特征的人力越来越紧张。恰逢深度学习在工业级推荐系统有大规模应用落地实践，因而逐渐将算法模型转向深度学习。



2018 年年初，美图用 NFM 模型首次打败 LR 并取得稳定效果，NFM 模型创造性的将 FM 和深度模型端到端的联合训练，在底层就进行显示的特征交叉 ,NFM 论文的实验和美图数据集上的实验都表明了模型能够收敛更快也更加稳定。在美图的实践中，引入右侧多层感知机学习隐式的高阶特征交叉之后，效果进一步提升，好于原始的 NFM 模型。改进后的 NFM 模型在我们的推荐流场景中取得了 5.5%的点击率，以及将近 7%的时长提升。

在 NFM 模型取得稳定的正向效果之后，美图推荐团队继续探索了业界更多的模型。不过都未能落地，主要有以下两个原因。

第一个是像 Wide&Deep， DeepFM， DCN 等从模型的复杂度上看没有比 NFM 拥有更强的预估能力，计算效率也没有明显优势。离线评估和线上实验上都没有得到正向效果。

第二种情况是 ,xDeepFM 和 NFFM 离线指标提升了 ,但是计算复杂度很高。此外 NFFM 模型参数量大，内存是个瓶颈。导致它俩无法大规模落地。

模型名称	出处	年份	实践效果(AUC)	优点	不足
Wide&Deep	Google	2016.06	-0.024	**首次公开&应用**深度学习到推荐系统	**LR组件**需人工组合特征，效率低
DeepFM	Huawei Noah	2017.03	-0.003	**FM取代LR组件**,不需人工组合	**仅限二阶特征**和隐式高阶特征
NFM	**NUS**	**2017.08**	**0.0(baseline)**	**引入特征的乘性关系到深度学习网络**	仅限于二阶特征和隐式高阶特征
DCN	Google	2017.08	-0.002	特征组合阶数可控，**不限于二阶特征**	低阶交叉无明显优势，高阶交叉时因多项式乘法，**数值不稳定**
xDeepFM	Microsoft Research	2018.05	+0.009	引入特征field信息构造复杂特征组合	计算复杂度高，**落地困难**
NFFM	NJU	2019.04	+0.009	引入特征field信息，构建二阶特征交叉	**模型参数量大，落地困难**

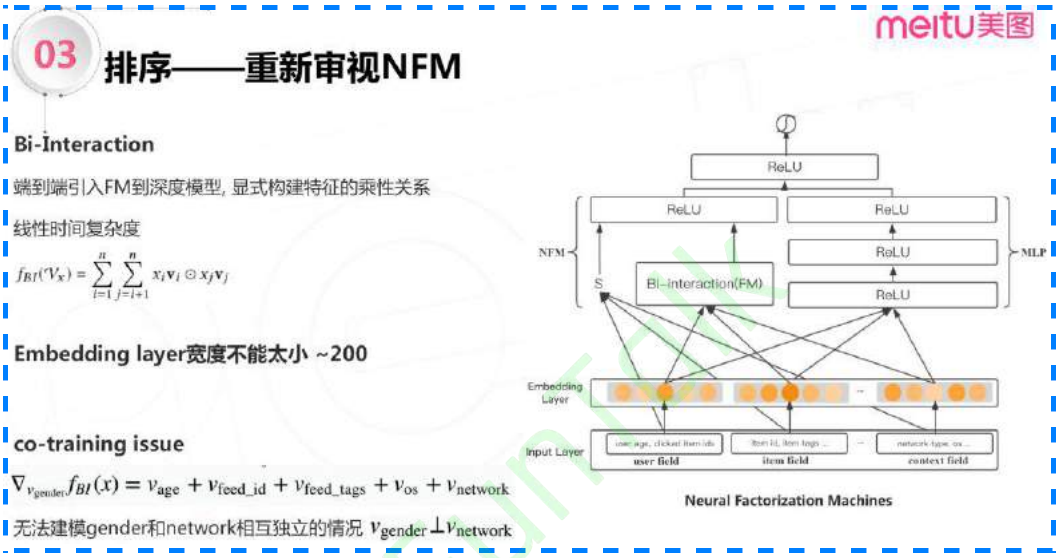
经过一年多的探索，在业界的众多模型中我们没有找到合适美图推荐场景的排序算法。另外，在我们引入行为序列特征之后，NFM 的计算复杂度已经不能很好的支持线上流量。在这样的背景下，美图算法团队决定自行设计算法。

NFM 模型的优点在于，通过 Bi-Interaction，将 FM 模型端到端引入到深度模型，显式构建特征的乘性关系，加强模型预估能力，同时没有增加时间复杂度。但是在实际生产实践中，存在 2 个不足：

(1) NFM 需要足够的 Embedding layer 宽度来学习特征。在实际场景下，其宽度取 200 左右，效果最好。但是随着百万级别用户行为序列特征的加入，

NFM 模型的计算量越来越大，越来越不能满足线上小于 300ms 的时延要求。

(2) 另一个不足是，NFM 模型本身存在 co-training 的问题，即：一个特征的学习，会不可避免地受到其他特征的影响。例如，用户的性别特征，与用户的网络环境特征是不相关的；但是 NFM 模型无法构建这种情况。



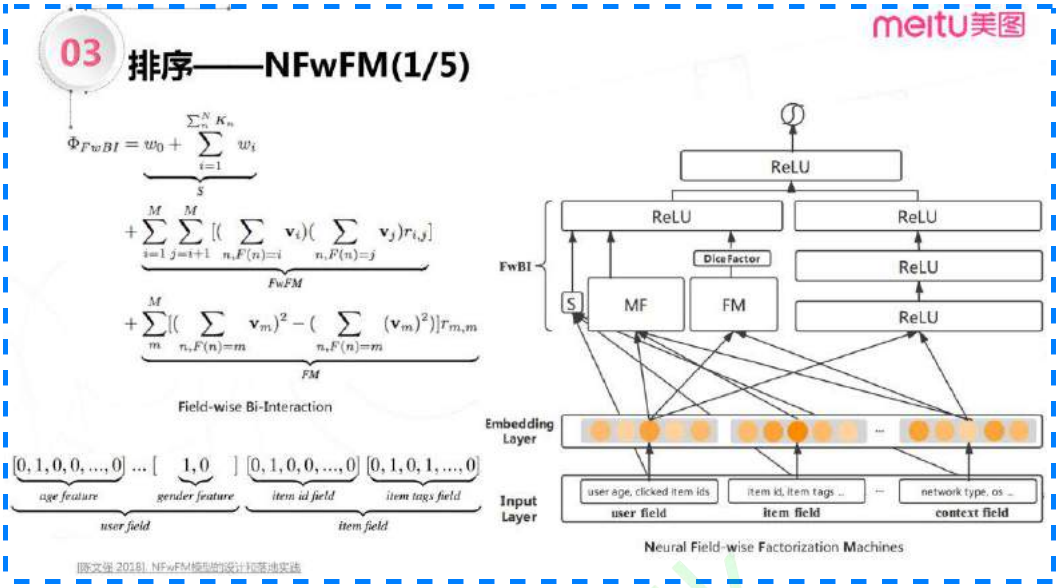
基于以上不足，我们先来看下业界相关经验：

不管是在传统浅层模型时代还是在深度学习时代，引入特征的 field 信息之后，模型几乎是总能带来提升的。比如 FFM 仍然一直活跃在 Kaggle 等 CTR 预估比赛中，稳定的取得比不能建模 field 信息的 FM 更好的效果。而深度学习时代，业界很多公司比如 2018 年，微软离线验证 xDeepFM 引入特征的 field 信息之后，相对不能建模 field 信息的 DCN 同样取得了很明显的提升，即便在现在，xDeepFM 仍然是很优秀的模型。但是他们或者计算量太高或者参数量太大，导致无法大规模应用到线上系统。基于上面对 NFM 模型的优点的实验和分析，美图算法团队开始尝试 NFwFM 模型。

2. NFwFM 模型

2.1 模型整体架构

NFwFM 模型是在 FwFM 模型的基础上演化出来的：通过 Field-wise Bi-Interaction 组件，将 FwFM 引入到深度模型里面。



上图是 NFwFM 模型的整体架构，首先，把特征按照逻辑分为 3 个大模块：用户侧特征（包括年龄、性别等），item 侧特征（包括 item id，item 标签等）以及上下文侧特征。

接下来将 FwFM 分解成了 3 个子模块：第一个模块是线性加和模块（上图中 S 表示），不区别 field 学习的特征；第二个模块是矩阵分解模块（上图中的 MF 部分），用来学习 field 粒度下的特征组合，比如 user field 和 item field 的二阶交叉；第三个模块是 FM 模块，用来学习 field 内部 feature 粒度的特征组合。

2.2 FwFM 和 FFM 相比

(1) FwFM 模型尺寸相对 FFM 少 M-1 倍。在美图实际应用中，特征量大约可减少 30 倍。

(2) FwFM 模型引入了 Field 相关的权重 $r_{i,j}$ ，解决了 FM 存在的不相关特征在学习过程中互相干扰的问题。

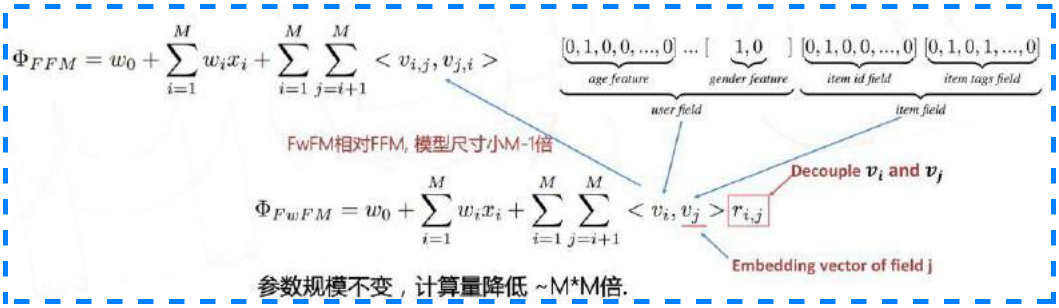
(3) 离线评估显示，FwFM 模型的预测性能（例如 AUC 指标）和 FFM 基本一致，而参数规模大大降低。

FwFM 模型由于要建模 field 信息，导致它无法像 FM 那样具备良好的线性时间复杂度。因此，需要将 FwFM 做矩阵分解（上面架构图中的 MF 模块）。

2.3 MF 模块

如下图所示，模型分别从用户侧和 item 侧提取特征向量 v_i 和 v_j，在这两个向量上进行矩阵分解，用来学习 field 粒度的特征组合。实际应用中，需要分别对用户侧、item 侧、context 侧进行两两矩阵分解，因此共有 3 个矩阵分解子模块。

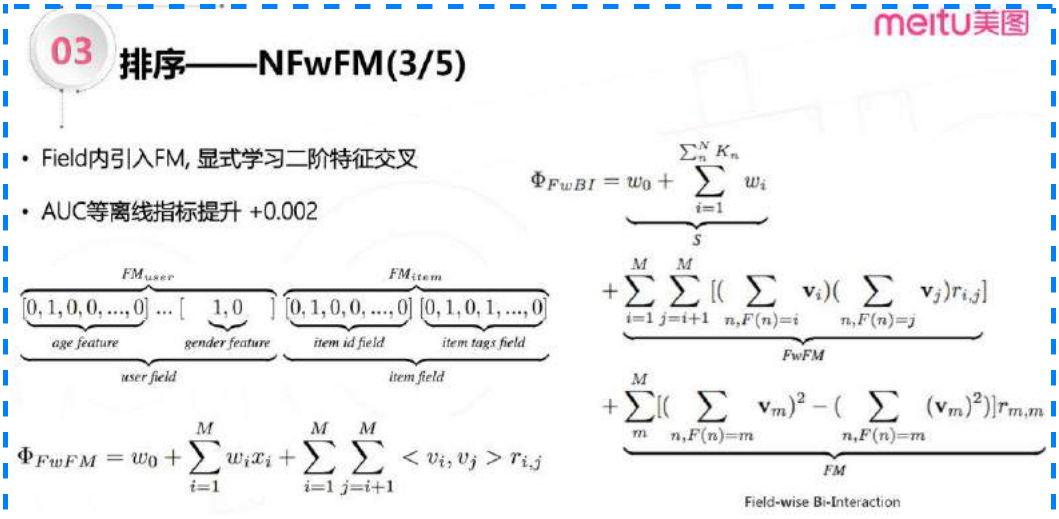
通过离线评估显示，MF 分解前后的 FwFM 模型，其 AUC 等指标持平，但是相同参数规模下计算量降低 M*M 倍，计算效率大大提升。



2.4 FM 模块

但是，MF 分解也会存在不足，例如：对于用户侧存在的丰富多样的特征，没有办法使用矩阵分解进行两两二阶交叉。

因此，在 field 内部引入 FM，用来显式学习用户侧 feature 粒度的所有特征的二阶交叉组合。具体来讲，给 user field 引入一个 FM 模型，对用户的年龄、性别等特征的二阶交叉，同样的算法也用于 item field 等。这样，FwFM 模型就演化成了下图这样一个 Field-wise Bi-Interaction 组件。引入 FM 模型后，模型的 AUC 指标提升了约 0.002。



2.5 解决特征间干扰问题

但是这样并没有解决最一开始提到的问题：FM 模型在学习过程中，特征存在互相干扰的情况。

回顾一下前文所述的特征间干扰问题，即 FM 的 co-training 问题：

$$\nabla_{v_{\text{gender}}} f_{FM}(x) = v_{\text{age}} + v_{\text{item_id}} + v_{\text{item_tags}} + v_{\text{os}} + v_{\text{network}}$$

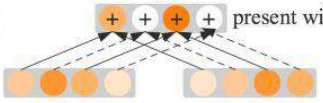
FM无法建模 $v_{\text{gender}} \perp v_{\text{network}}$

模型在对每一个特征进行学习的时候，都需要和其他特征进行交叉。例如，用户性别特征和网络环境特征应该是不相关的，但是模型在学习性别特征的时候不可避免地受到网络环境的影响。

为解决这一问题，借鉴 dropout 思路：模型训练完成 Bi-Interaction 后，按照伯努利分布（期望为 β）随机丢弃部分二阶交叉项，以解决部分 co-training 问题。

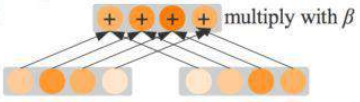
$$\Phi_{FM_\text{train}} = w_0 + \sum_{i=1}^{N} w_i + \sum_{i=1}^{N}\sum_{j=i+1}^{N} p\boldsymbol{v}_i \odot \boldsymbol{v}_j$$

where $\mathbf{p} \in \mathcal{R}^{K_e}$ and $p[i] \sim Bernoulli(\beta)$



(a) train phase

$$\Phi_{FM_{\text{inference}}} = w_0 + \sum_{i=1}^{N} w_i + \sum_{i=1}^{N}\sum_{j=i+1}^{N} \beta\boldsymbol{v}_i \odot \boldsymbol{v}_j$$



(b) inference phase

具体来讲，先从伯努利分布中采样出由{0,1}组成的向量，再用该向量和 FM 模型计算得到的表示二阶特征交叉组合的向量进行相乘，这样可以随机丢弃部分二阶交叉项。在预估的时候是将 FM 做了 Bi-Interaction 后得到的向量，乘以伯努利分布的期望 β，用来对齐计算过程中丢失的数据的大小。引入如上思路之后，AUC 提升约 0.001。

2.6 总结

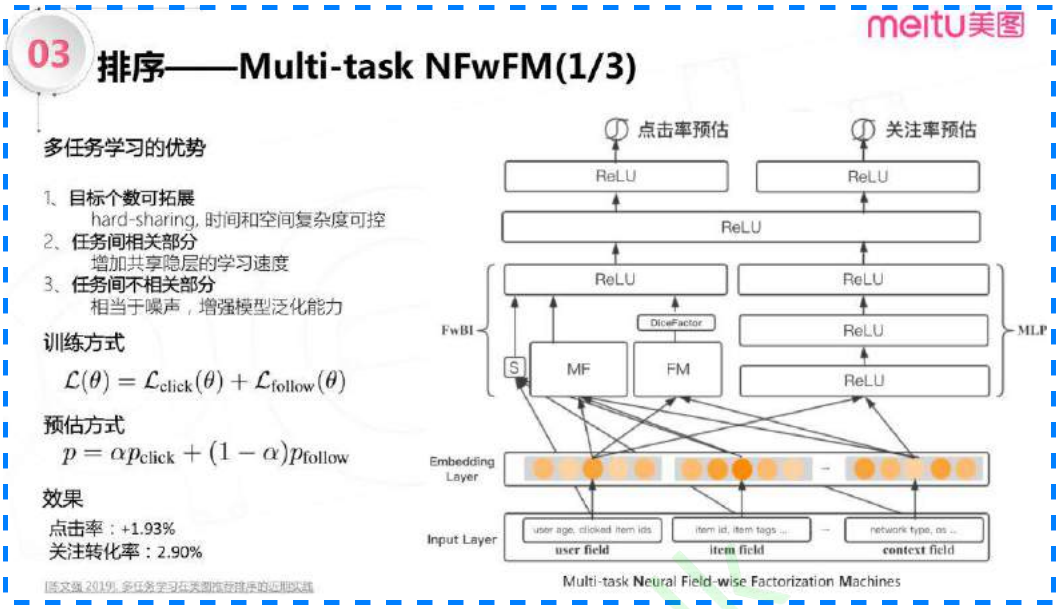
总体来讲，我们从 FwFM 演化出 Field-wise Bi-Interaction 组件，包含线性加和模块用来学习一阶特征，还包括矩阵分解（MF）模块和 FM 模块，用来学习特征 field 粒度和 feature 粒度的特征交叉。相比于上一代 NFM 模型，使用这样的模型，在计算量和参数量都减少了 6 倍的情况下，点击率得到了 5.19%的提升。

3. Multi-task NFwFM

3.1 多任务基本架构

接下来是多任务方面的工作。在深度学习时代，深度模型能够包含多种不同分布的样本，释放了多任务学习的最大价值。从实践角度考量，为使离线训练和在线预估效率较高，目标个数具有可拓展性。业界通常会选择下图中这种底层硬共享（hard-sharing）隐层的多任务架构，在这种架构下，因为点击率和关注转化率任务是强相关的工作，能增加共享隐层的学习速度，从而

增加模型的收敛效率，而这两个任务中不相关的部分可以认为是相互任务的噪声，可以增强模型的泛化能力。



在学习的过程中，是利用两个任务简单加和的方式来学习多目标的。这个方式的离线 AUC 和单独的点击用户模型的 AUC 基本持平；线上点击率提升 1.93%，关注转化率提升 2.90%。

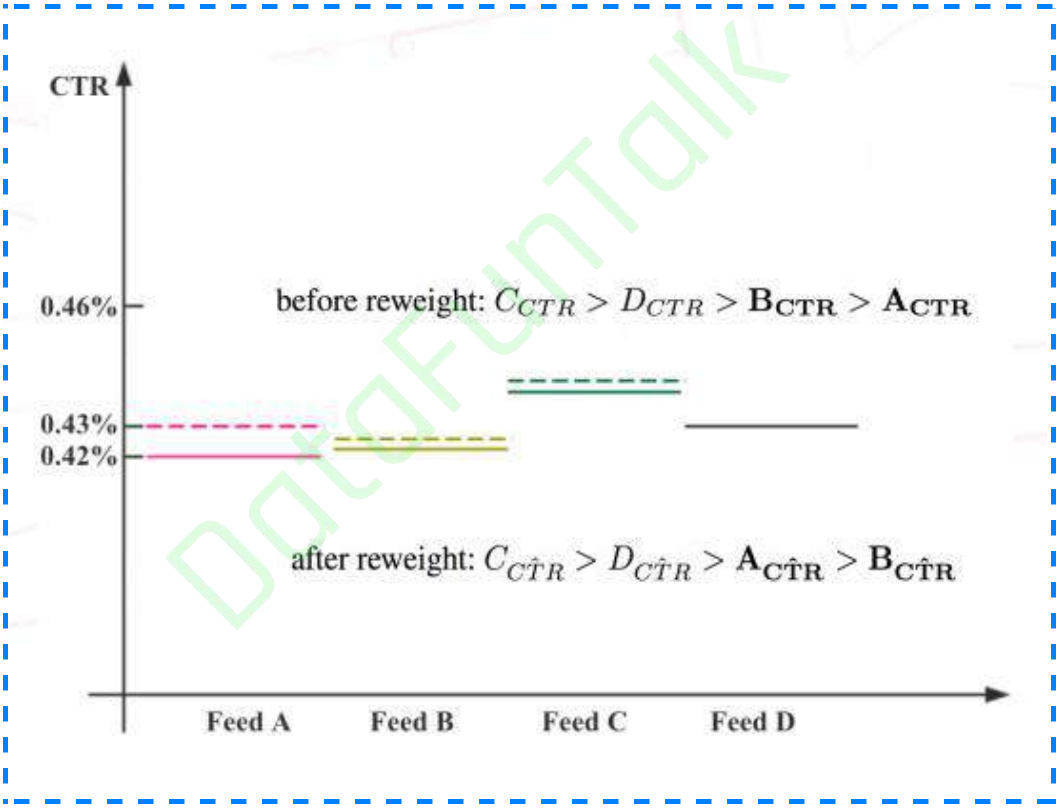
$$\mathcal{L}(\theta) = \mathcal{L}_{\text{click}}(\theta) + \mathcal{L}_{\text{follow}}(\theta)$$

在实际情况中，点击和关注的样本比例大概为 100:1；在这样很少的关注样本的情况下，使用上述的多任务架构就可以得到稳定的提升；这驱使我们引入更多的关注数据，来压榨多任务模型学习更多更高质量数据的能力。

3.2 样本 reweight

具体来说，我们引入一个样本 reweight 的概念，主要目的是为了引入更多更高质量的关注行为数据。因为无数的经验告诉我们这是非常有效的做法。

以下图为例，假设有 A、B、C、D 这 4 个 item，图中实线部分表示 item 的真实 CTR（由大到小分别是 C > D > B > A）；而实际的关注转化率的关系是 A > C > B > D = 0。



我们用实际的关注转化率取 reweight 这个样本之后，得到下图虚线部分的结果：C > D > A > B，即原本点击率最低的 item A，在 reweight 之后由于关注转化率高而变得点击率比 B 更高，这样 item A 更容易被模型推荐出来，这样就能够提升整体的关注转化率，并且因为 C、D 等因为本身点击率

较高或者没有关注行为，它们不受分布改变的影响，因此他们的 CTR 大小关系不受影响。换一种理解，我们是在仅仅改变了有关注 item 的点击率分布的基础中引入了更多更高质量的关注行为数据。对原来的点击率预估模型的侵入很少，整体点击率不会下跌太多。该工作提高关注转化率 14.93%，但是点击率提升很少（ 约 0.84% ）。

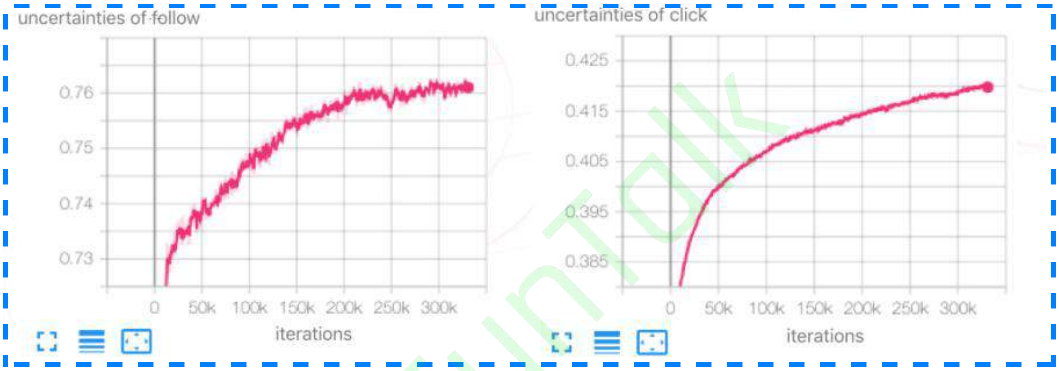
3.3 Homoscedastic Uncertainty 学习方式

上述工作点击率提升很少的原因是 reweight 模型过于简单。如前文所述，多任务的缺点是在参数共享的情况下，如果两个任务有不相关的部分，两个任务就会互相干扰，从而影响效果。学术界将这种现象称为共享冲突。共享冲突这一问题分析和解决起来较为复杂。针对美图的具体场景，减少共享冲突的一种方法是加大点击率预估任务的重要性，让点击率预估任务主导底部共享参数学习，进而让整体模型优先正确预估点击率模型，再去预估点击转化率任务。

在实践中，我们用同方差不确定性来学习每个任务对整体的主导能力。具体来说，分别给点击率任务和关注转化率任务各自一个参数（ θ_{click} 和 θ_{follow} ）用来表示各自的不确定性；不确定性越小的任务对模型整体的主导性越强。

$$\mathcal{L}(\theta) = \frac{1}{2\sigma_{\text{click}}^2}\mathcal{L}_{\text{click}}(\theta) + \frac{1}{2\sigma_{\text{follow}}^2}\mathcal{L}_{\text{follow}}(\theta) + \log\sigma_{\text{click}}^2\sigma_{\text{follow}}^2$$

从下图可以看出，左图的关注转化的不确定性达到 0.76，确实比右图的点击率的不确定性（ 约 0.42 ）更高；因此，让点击率预估任务主导整个模型的学习。



这样可以避免点击率下跌的风险（ 实际上，点击率提升了 1.57%），而关注转化率的提升达到了 15.65%。

4. 总结

排序端的工作，美图经历了从 LR 模型到深度学习模型的引进：

(1) 引入 NFM 模型，点击率提升了 5.53%，人均时长提升 6.97%

(2) NFwFM 模型在引入了特征 Filed 信息后，在模型尺寸和计算复杂度可控的情况下，点击率提升了 5.19%，人均时长提升了 2.93%；

多目标 NFwFM 模型，在引入更多更高质量数据之后，不仅关注转化率提升了 15.65%，点击率也提升了 1.57%。

参考文献

1. Covington P, Adams J, Sargin E. Deep neural networks for youtube recommendations

2. Ma J, Zhao Z, Yi X, et al. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts

3. Rich Caruana. 1998. Multitask learning. In Learning to learn

4. Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection

5. Kendall A, Gal Y, Cipolla R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics

6. [白杨-2018]基于用户行为的视频聚类方案

https://cloud.tencent.com/developer/article/1193177

7. [蒋文瑞 2018]. 深度模型 DNN 在个性化推荐场景中的应用

https://cloud.tencent.com/developer/article/1193180

8. [陈文强 2019]. 多任务学习在美图推荐排序的近期实践

https://cloud.tencent.com/developer/article/1475686

DataFunTalk

推荐系统中模型训练及使用流程的标准化

嘉宾：梁超　编辑：Hoh Xil

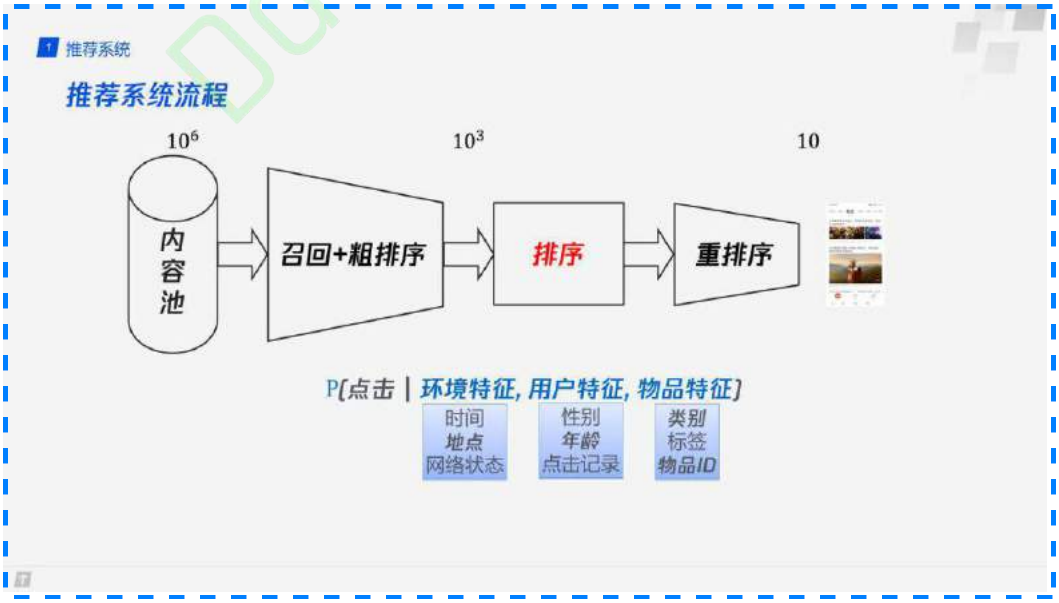
导读：本次分享的主题为**推荐系统中模型训练及使用流程的标准化。**在整个推荐系统中，点击率（CTR）预估模型是最为重要，也是最为复杂的部分。无论是使用线性模型还是当前流行的深度模型，在模型结构确定后，模型的迭代主要在于特征的选择及处理方面。因而，如何科学地管理特征，就显得尤为重要。在实践中，我们对特征的采集、配置、处理流程以及输出形式进行了标准化：通过配置文件和代码模板管理特征的声明及追加，特征的选取及预处理等流程。由于使用哪些特征、如何处理特征等流程均在同一份配置文件中定义，因而，该方案可以保证离线训练和在线预测时特征处理使用方式的代码级一致性。

一．推荐系统

1. 业务简介

这是我们的产品天天快报，会涉及首页以及数十个子频道，对于这些频道我们都需要做召回以及排序模型。如何高效的管理这么多的频道呢？我们就需要一个很好的系统来管理所有的特征和模型。

2. 推荐系统流程



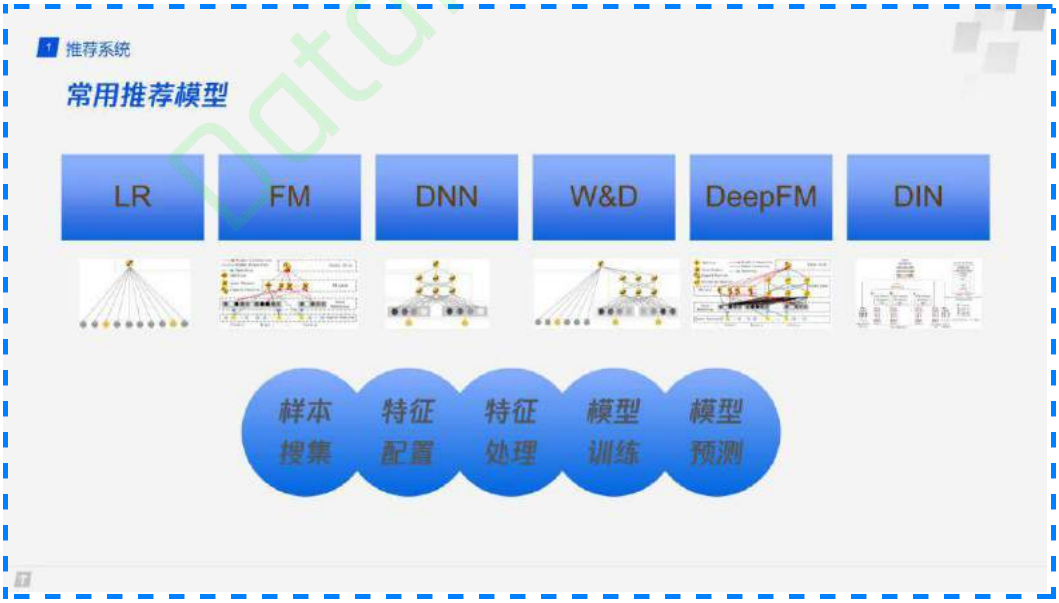
简单回顾下推荐系统的流程，整个推荐系统需要从数以百万计的内容池中筛选出数以十计的文章推荐给最终的用户。在这个过程中主要涉及三个步骤：

第一步，从百万量级中通过环境特征，用户特征，物品特征等信息来找出千级别的文章。

第二步，通过排序模型来预估点击率或者预估用户对这篇文章的偏好程度。

最后，通过一些运营规则和多样性方面的考量（比如用户喜欢王者荣耀，但不能推荐给用户都是这类的视频或文章 ），最终呈现给用户 10 篇左右的文章。

3. 常用推荐模型



常用的推荐模型有 LR、FM、DNN、W&D、DeepFM、DIN 等模型，对于推荐系统，无论使用哪种模型，都需要以下几个模块：

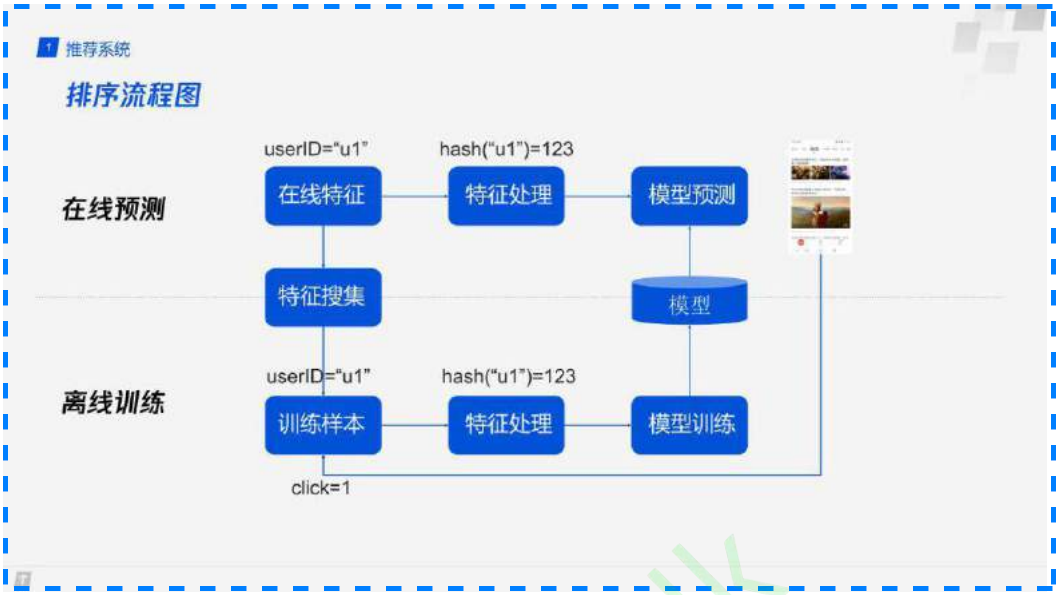
样本搜集：训练模型离不开大量的训练样本，所以需要进行样本（ 特征和标签 ）的搜集；

特征配置：实际的推荐系统中会有上百个特征供模型选择，在模型版本迭代的过程中，有些特征会被舍弃，有些特征会新加进来；因而，我们就需要配置搜集哪些特征、使用哪些特征，在迭代过程中，还需要保证现有模型训练和预测服务的稳定性；

特征处理：对于每个特征，比如用户 ID，该如何离散化成一个最终使用的 int 型的数字，就需要经过一定的特征处理；

模型训练&模型预测：特征处理完之后，如何喂给模型训练程序以及线上的预测模型，如何在修改了特征配置之后，无需人工修改训练和预测代码，从而降低工作量、减少 bug 的引入，都是我们需要考虑的工作。

4. 排序流程图



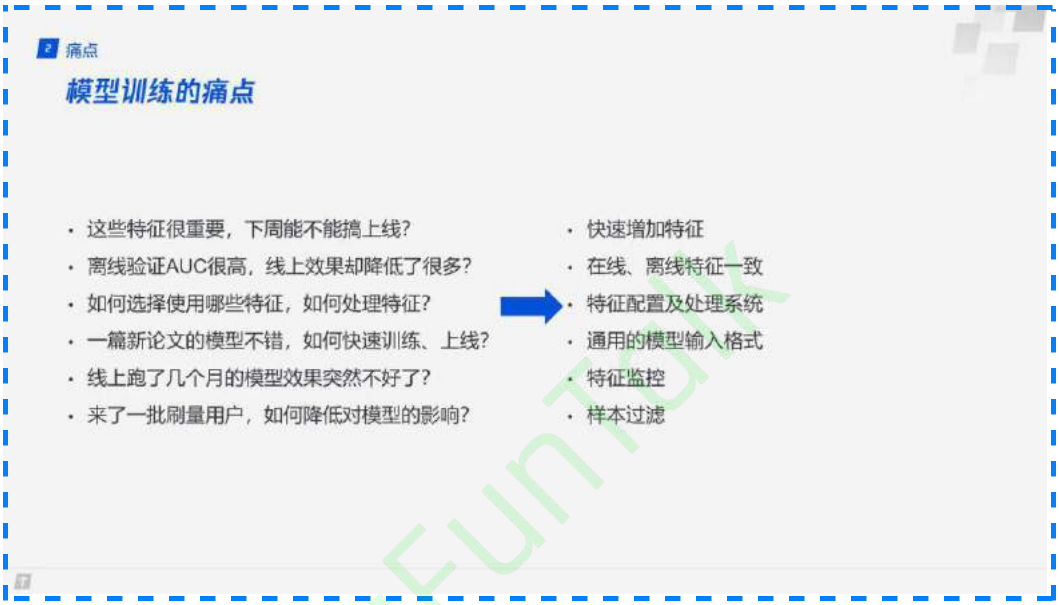
上图为排序系统的流程图：

以用户 ID 特征（userID）为例，在线预测时，会首先把 userID 填入某一个变量中，并通过某种 hash 函数把它变成整数类型（比如 C++ long 类型），再输入到模型中；与此同时，我们需要把在线的特征记录到日志中，作为模型训练的样本。

离线训练：将特征日志和用户行为（是否点击、是否点赞、消费时长等）日志结合起来，形成最终带有标签的训练样本，再通过同样的特征处理流程，形成训练样本进行模型的训练。

二． 推荐系统中模型迭代的痛点

与研究中给定的数据集不同，推荐系统中的模型需要不断地迭代调优。在日常的工作中，我们常常需要在保证现有模型服务稳定的前提下，不断地增加新的特征，训练新的模型。于是，我们会面临下图所示的诸多问题。



三． Write once，run anywhere 的特征处理标准

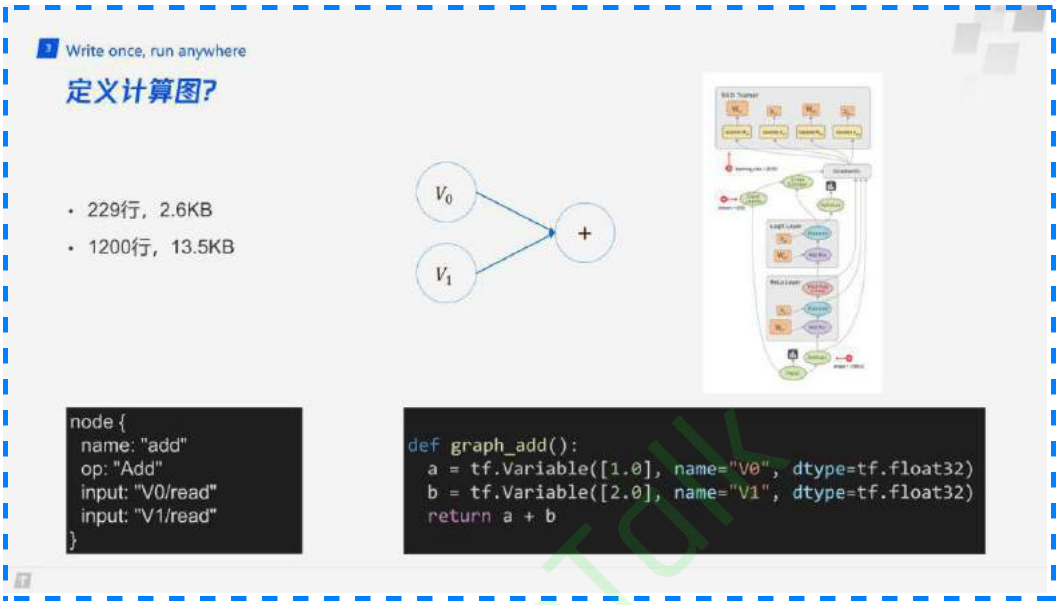
要设计一套特征处理的标准，我们首要面临的问题是如何描述特征处理的流程（C++ 代码？protobuf？XML？）。基于以下几点考虑：

1. 尽量减少人工写代码的量；

2. 易于查看和维护；

3. 易于迭代，我们设计了一套基于 CSV 格式的特征处理框架。

首先，我们来看一下，在模型训练方面，业界是怎么做的。



在工业界，对于的模型训练和预测部分，TensorFlow 等框架已经做得比较完善了。那么，TensorFlow 是如何定义整个数据流的呢？它是用计算图来定义的，以两个变量相加为例，代码非常简单，如果转化为 pb（ 上图左下角 ）也只有这几行。但实际上呢？这里给出两组数据：229 行，2.6KB；1200 行，13.5KB。229 行，2.6KB 为上面的加法操作转化为实际的 pb 的大小，而如果有 10 个加法操作的计算图，则需要 1200 行，13.5KB。所以，用通用的计算图来定义特征的处理流程，虽然很灵活，但却非常不利于人来阅读和管理。在系统设计的过程中，我们期望所有的特征定义及其处理流程都可以一目了然的看到。

如上图所示，在考虑到样本搜集、特征配置、特征处理、模型训练、模型预测等需求后，我们选用了 CSV 来管理整个过程，CSV 中的每一行定义了一个特征，包含了特征的名称、类型、序列化后的位置、处理方式等信息，且可以灵活地增加列来定义新的功能。

下面沿着之前提到模型迭代的痛点，依次看下我们是如何通过一个 CSV 来解决的：

痛点 1：快速增加特征

首先旧的流程中，我们都需要声明一个变量来临时存储在线所需要的特征，编写特征填充代码，同时还需要编写特征变换代码、特征序列化代码、特征反序列化代码以及特征监控代码。以上种种功能，都需要针对每一个变量进行独立的编写。

我们新的流程中，只有在 CSV 中定义变量处理方式和编写特征填充代码两个部分：

如上图右下角有 4 个特征，假设用户 ID，用户性别以及 itemID 是已有的特征。现在，我们需要新加一个特征，我们就会在这个表格第四行新加用户 Tag 特征，同时定义下这个特征的类型以及在日志中的位置，是属于用户特征还是物品特征，剩下的步骤则通过一个 python 脚本和一个代码模板来生

成新的 C++ 程序自动完成。增加了这个变量后，特征日志中会增加上图右上角所示的 tag 信息。

痛点 2：在线、离线特征的一致性

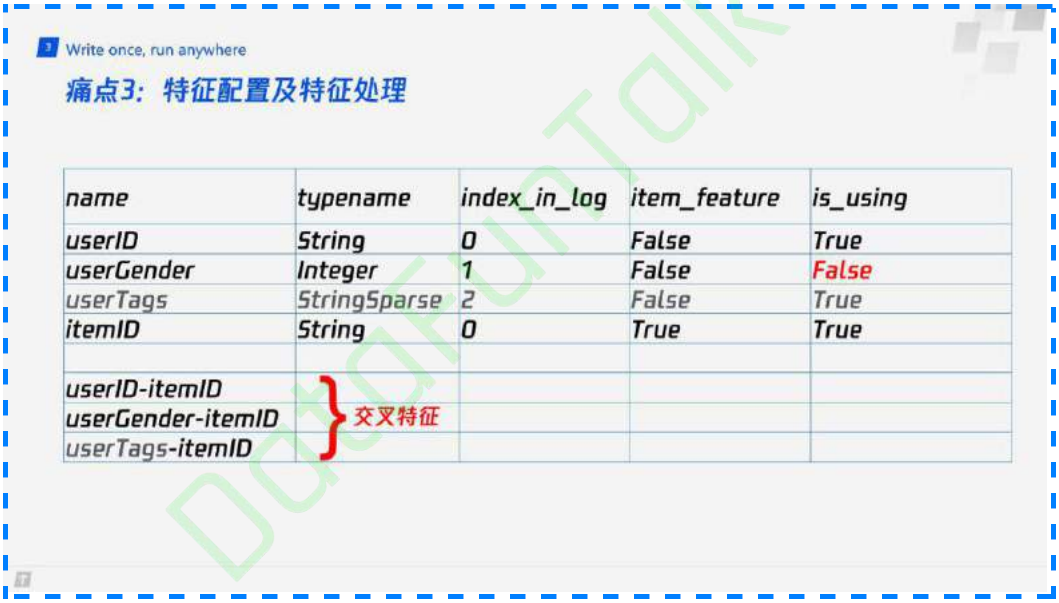
模型训练所需的特征需要和在线预测时的特征完全一致。在工业界中，一般会将在线特征 dump 到日志中，训练时结合标签生成完整的训练样本，从而保证在线、离线特征的一致性。然而，旧的流程中，针对每个特征的序列化，都需要手写代码，反序列化亦然，这就大大增加了算法工程师的工作量，且容易引人 bug。

我们的做法是把特征的类型进行了标准化，抽象出 4 种标准的类型（ 整形、稀疏整形、字符串、稀疏字符串 ），它们都继承自基类 Feature，这个类会包含特征处理的方方面面，如生成特征、序列化、反序列化。我们只需要保

证每种特征类型的特征的序列化和反序列化函数是正确的，就可以保证在线的特征和离线特征是完全一致的。

痛点 3：特征配置及特征处理

① 特征配置



特征的配置包含两方面的内容：搜集哪些特征及模型使用哪些特征。

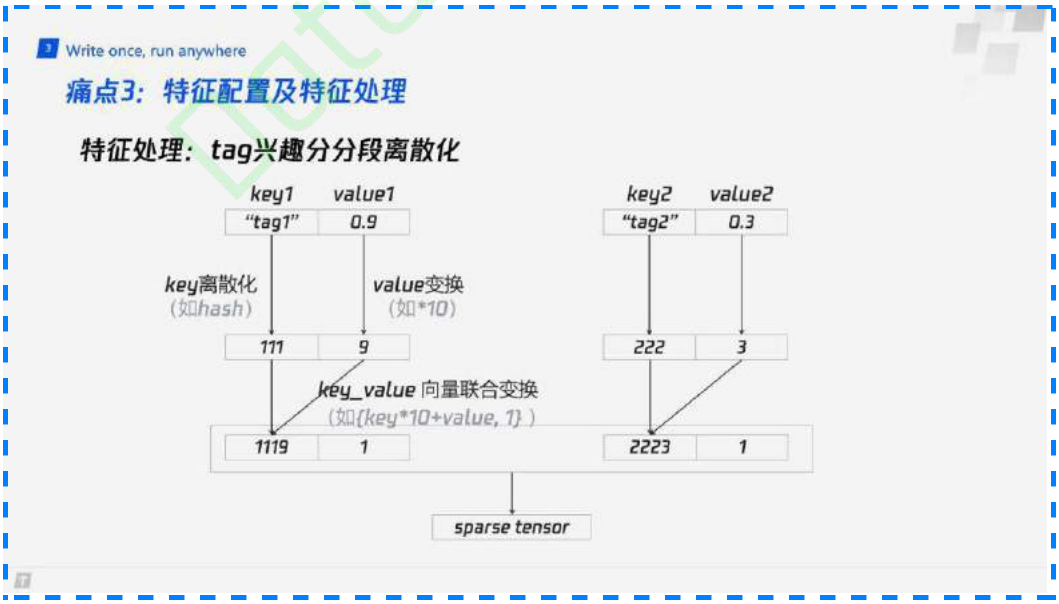
在实践中，我们需要保证已有模型的稳定性，并不断地搜集新的特征。为此，我们将特征搜集服务与 ranking 服务相分离，但复用特征填充代码。服务分离有两个好处：

1. 在特征搜集服务中新增所需搜集的特征无需更改 ranking 服务；

2. 在 ranking 阶段，一般有数千个物品，而我们的特征搜集服务只搜集返回给用户的十来个物品的特征，大大减小了日志量。

搜集到的特征是模型训练和预测所需特征的超集。当需要进行模型的训练或预测时，我们只需在 CSV 中使用 is_using 列来控制是否使用某一特征。需要进行模型迭代时，只需要 CSV 中的配置，并重新生成一份代码即可。此外，交叉特征也只需要在 CSV 中进行配置即可，并且，由于对特征类进行了标准化，我们可以轻松支持任意个特征的交叉。

② 特征处理



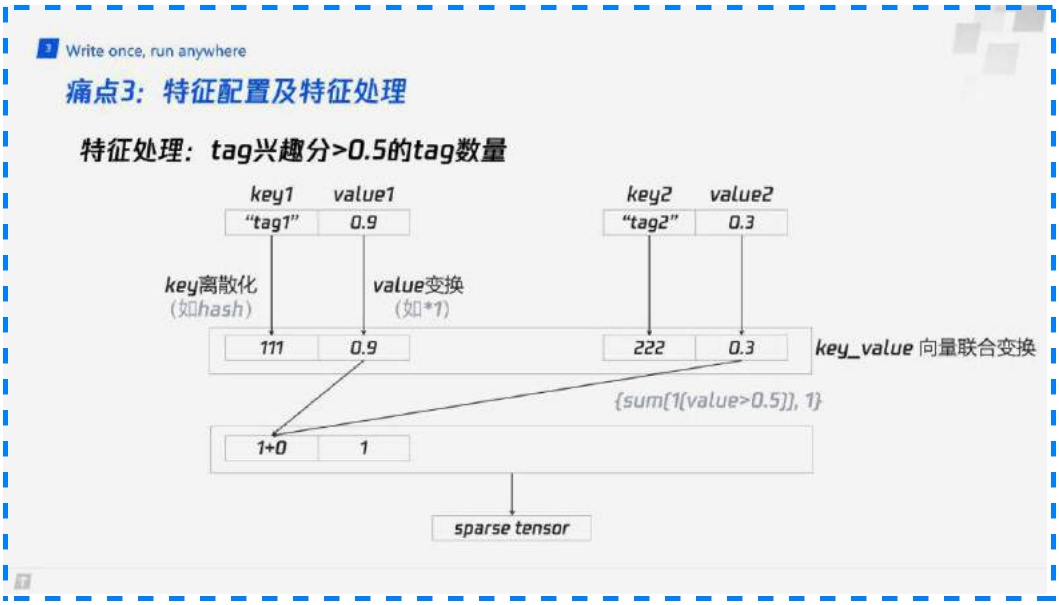
如果我们像 TensorFlow 那样定义一个非常灵活的计算图的话，虽然是很好的，但是不利于模型的管理。因此，我们把单个特征的处理抽象成了 3 步：

特征填充（ 手工编写代码或经由其他特征变换而来 ）；特征值和特征权重变换；特征值和特征权重向量联合变换（ 支持多次变换 ）。

仍以 tag 特征为例：

如上图所示，一般情况下，每个特征都会有 tag 值（key）和权重（value），我们会先将 key 进行离散化，比如 hash；并对权重做一定的变换，比如设为 1；之后，对整个 key、value 向量进行联合变换，比如，key 乘以 10，value 不变（ 举例用，一般不做变换 ）。

此外，有时我们需要一些统计信息，比如 tag 分数大于 0.5 的 tag 数量。那么，就可以真正用到特征值和特征权重向量联合变换，我们只需要在这一步统计整个 key、value 向量中 value>0.5 的个数即可。



那么，如果我们既需要保留 tag 信息（第一种变换），又需要 tag 分>0.5 的个数作为特征呢？我们只需要在 CSV 中重新声明一个变量，并在特征赋值部分将其特征设为第一个变量的内容，并进行相应变换即可（实际中，可以直接在赋值部分写统计函数即可）。

痛点 4：支持多种模型

我们的系统支持两种训练样本格式：libsvm 和 sparse tensor 数组。

其中，libsvm 是线性模型的主流格式；而 sparse tensor 则是 tensorflow 中的支持稀疏特征的主流格式（tensor 可以视为 sparse tensor 的特例）。

以上图中的样本（ 省略了标签部分 ）变换过程为例 ,该样本中包含两个物品信息，因而会生成两条样本。对于 libsvm 格式，只需要将每个特征变换后的结果存储到一个向量中即可。对于 TensorFlow 等框架，内部都是用矩阵来进行运算的。矩阵又会分为两种：稠密的矩阵和稀疏的矩阵。同时，稠密矩阵又是稀疏矩阵的特例。所以，我们会将所有的特征全部以稀疏矩阵的形式喂到模型中，方便程序统一的处理。还是以稀疏特征 tag 为例，该特征会用户两个稀疏矩阵（ 特征值矩阵和权重矩阵 ）来进行表示 ,且共用下标和形状，特征值就是刚刚经过离散化的特征值，这里没有用到权重，所以全部设为 1。我们可以看到，虽然它是一个稀疏矩阵，但是它是一个 2x2 的矩阵，每个都有元素，所以可以用稀疏矩阵来表示稠密矩阵。

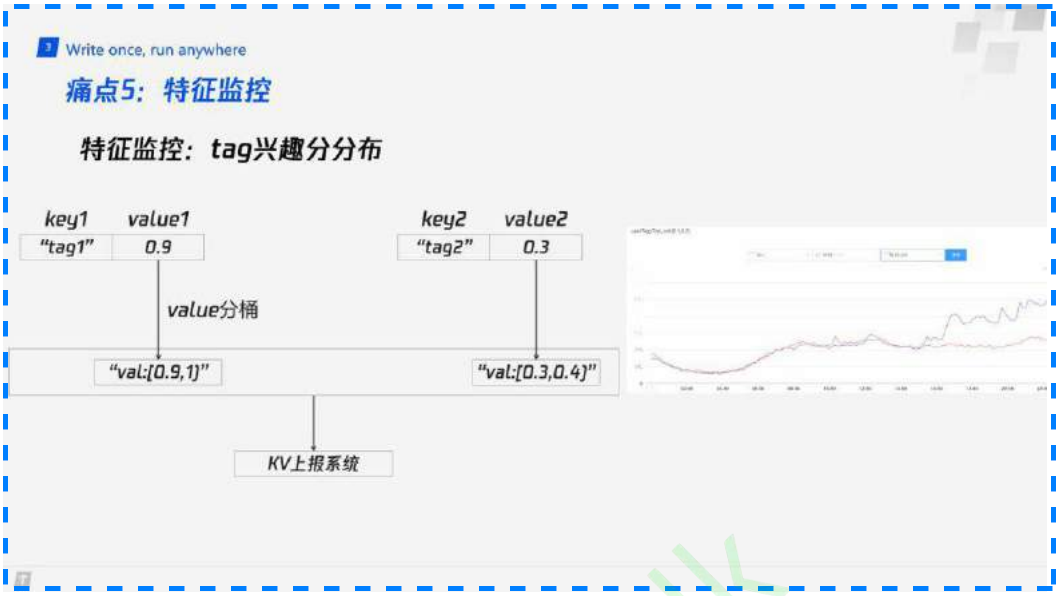
有了训练样本之后，如何进行模型训练？我们提供了 3 种方式：

通过将 CSV 转换为一个 hpp 文件之后，我们会编译出一个专门用于将原始特征日志转换为训练样本的可执行程序，并通过 hadoop streaming 方式，生成 libsvm 格式的训练样本。这种方式有两个缺点：增加了流程的复杂性，且耗费存储资源。原始的特征日志相当于进行了压缩（多个物品共用一组用户特征），展开之后相当于每条样本对应的用户特征是重复的，且会生成大量的交叉特征，这会导致文件的大小增加 10 倍以上。

第二种形式，则是将生成的 hpp 文件通过 JNI 编译成一个 SO，可以直接在 Spark 上调用，生成 libsvm 格式的 RDD 进行训练，该方案避免了训练样本占用磁盘空间的问题，但流程仍较为复杂。

最后，则是我们目前使用的动态编译 so 的形式。由于 tensorflow 模型训练程序是 python 编写的，而我们的 CSV 转 hpp 程序也是 python 编写的，因而，我们在使用 tensorflow 训练前，会检测 CSV 是否更新，然后动态的决定是否重新编译自定义的 tensorflow 算子。在训练时，该算子会将原始特征日志转换为 sparse tensor 格式的训练样本。此外，使用配置文件还有一个好处：训练程序还会读取 CSV 中额外的配置信息，从而知道有多少个特征每个特征 embedding 的维度、大小，是否需要 attention 机制等信息，供模型训练使用。

痛点 5：特征监控



由于推荐系统的复杂性，我们需要对各个环节进行必要的监控，从而保证出现问题时可以及时知道。以 tag 兴趣分分布为例：

类似于特征的变换流程，我们会在 CSV 中配置监控函数。如上图所示，为 tag 特征的 value 分桶监控过程。

首先，对 tag 的兴趣分进行分桶，比如这里有两个兴趣分，我们可以把它们分成 10 段，0.9~1 是一段，0~0.1 是一段等等，再把这些序列化后的字符串通过上报系统进行上报，然后展示在右边的曲线中。通过这些曲线，我们可以对比同一区间内特征数量的同比、环比等信息，从而在特征分布变化剧烈时及时进行告警。

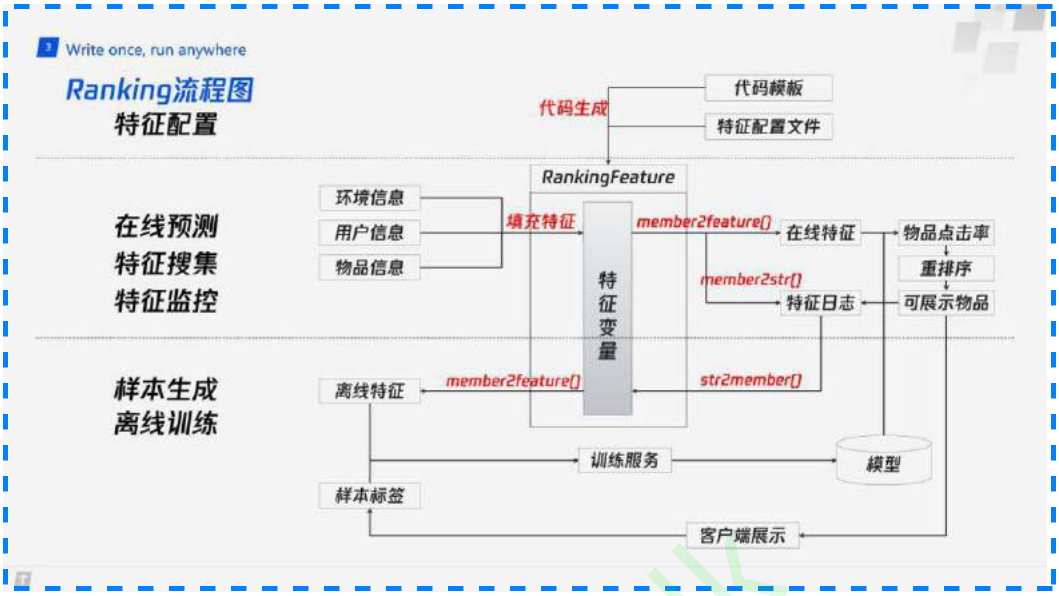
痛点 6：样本过滤 & 加权

我们实际的特征格式如上图左侧所示，我们会在用户特征和物品特征后面分别加上几列，会统计某一段时间内用户或物品的曝光次数，点击次数，以及消费时长。

如果某一用户短时间内曝光超过 1000 次，或者消费时长特别长，或者点击率特别高，则可能是机器刷量的，我们就会将这些样本过滤掉。

此外，对于一条样本，即使用户点击了，如果消费时长过短，我们也会将其设为负样本或者过滤掉，或者设一个比较小的权重。

Ranking 流程图



最后看一下完整的系统流程图：

- 首先通过特征配置文件和一个代码模板，管理所有的特征。
- 通过脚本配置文件生成 hpp 代码，模型预测时使用该代码进行特征的变换。
- 在重排序确定要展示给用户哪些物品之后，重复一遍特征填充的过程，然后再把可能产生曝光的物品特征序列化到特征日志中。
- 在离线过程中，将特征日志通过反序列化的方法，重新填充整个特征类。通过同样的特征变换代码，变换成和线上完全一致的特征（针对同一版模型），等到样本标签从客服端返回之后，生成最终的训练样本，供模型训练。

四．总结

我们将推荐系统中特征处理的流程进行了标准化，该标准化体现在特征类型的标准化和特征处理过程的标准化两方面。我们通过一个 CSV 文件完成了特征配置、特征搜集、特征处理、模型训练（ 部分 ）、模型预测（ 部分 ）的管理工作，大大降低了人工的编码量，提高了工作效率、降低了人为引入 bug 的可能性。

DataFunTalk

如何从 0 到 1 构建个性化推荐？

嘉宾：曾钦榜　编辑：周晓侠

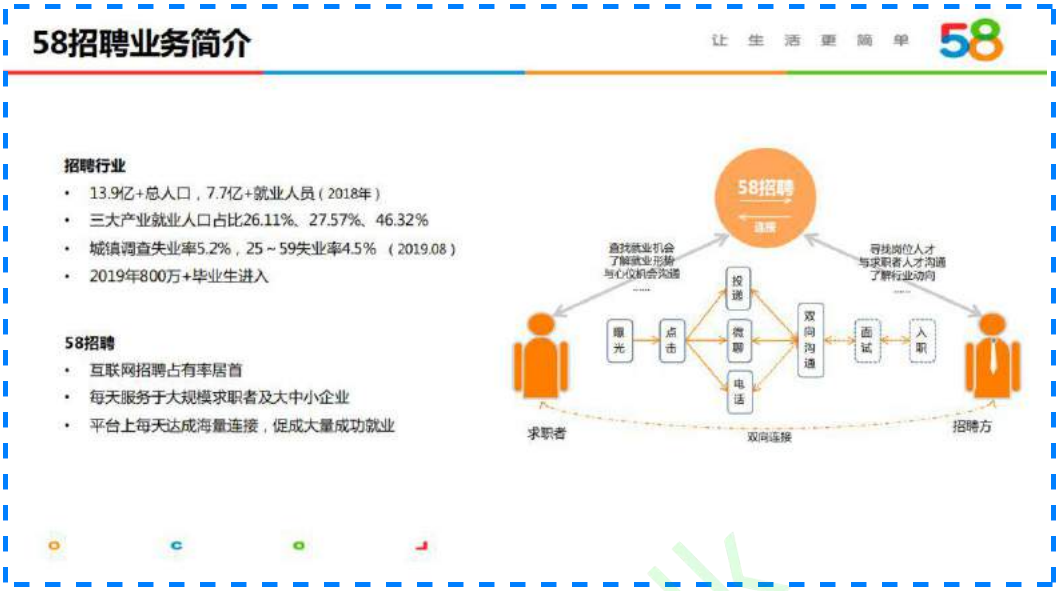
导读：随着科学技术的飞速发展，互联网被广泛应用于各个领域，而以互联网为基础的招聘模式也越来越受到企业的青睐。互联网招聘具有不受地域限制、覆盖面广、招聘成本低、针对性强、方便快捷、时效性强等优点，现已得到广泛应用，其中，58 招聘是互联网招聘行业中规模最大的平台。今天主要跟大家分享下 58 招聘如何通过个性化推荐技术服务大规模求职者和招聘企业。分享题目是**从零到一构建 58 招聘个性化推荐**，主要通过以下三方面进行介绍：

- 招聘业务介绍
- 个性化推荐实践
- 心得分享与规划

——招聘业务介绍——

1. 58 招聘业务简介

DataFunTalk



2018 年我国全国总人口 13.9 亿多 ,其中就业人口 7.7 亿 ,招聘基数庞大。三大产业就业人口占比分别 26.11%，27.57%，46.32%，其中第三大产业占比最大，部分发达国家第三大产业占比已达到 70%~80%，随着经济的发展，我国未来就业市场和就业分布将发生大的变化。2019 年 8 月城镇调查显示我国失业率为 5.2%，其中 25~59 岁失业率 4.5%，同时每年有 800 多万的应届生加入就职市场。58 招聘作为我国互联网招聘行业之首，每天服务于千万级求职者和大中小企业，平台每天生成千万级连接，促成大量求职者求职成功。

58 招聘平台主要服务于求职者和招聘方，接下来主要通过求职者的角度介绍用户在整个平台流转的大致流程，具体如下：

❖ 基于求职偏好搜索职位并点击查看详情。

- 投递有意向职位，或通过平台在线微聊工具、电话与招聘方做进一步沟通。

- 双方达成共识后，进行面试与入职。

相比传统推荐系统，58 招聘的业务漏斗更长更深，并且有一部分转化平台无法完全捕捉，形成了 58 招聘个性化推荐开展的难点及挑战。

2. 58 招聘推荐场景类型

58 招聘推荐场景主要面向 C 端求职者和 B 端企业，推荐内容主要包括：职位推荐、标签推荐、企业推荐、简历推荐。

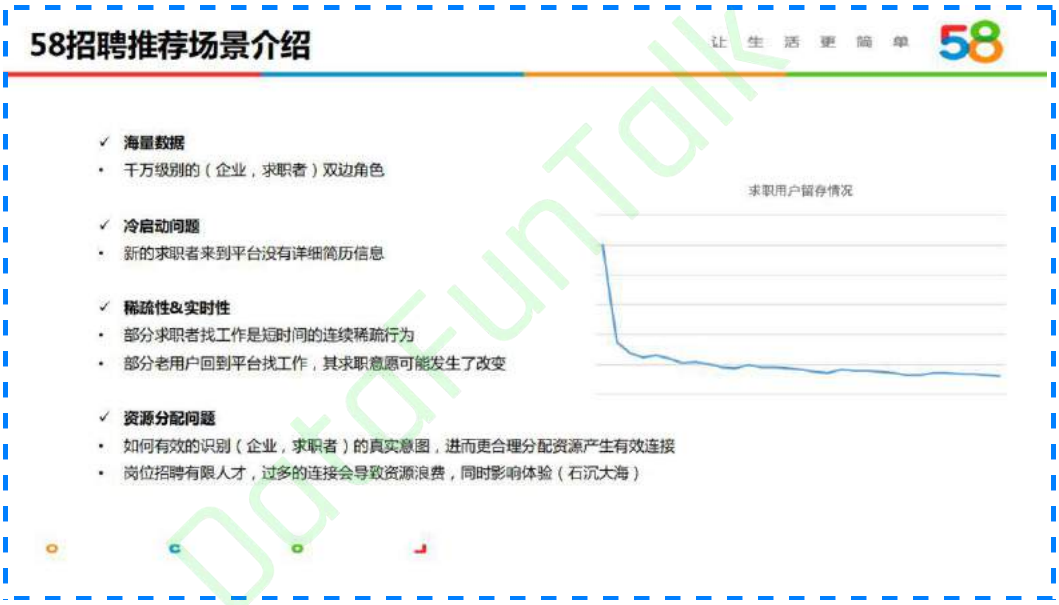
C 端求职者的典型推荐场景包括：

- App 首页招聘大类页：主要包括职位专区聚合、职位 Feed 流。

- 类目推荐：用户点击某个类目后，进行相关职位推荐。

- 相似推荐：用户点击某个具体职位后，在下方展现相似职位。

3. 58 招聘推荐主要问题



58 招聘推荐相对其它行业主要存在以下典型问题：

海量数据计算：大多数公司都存在，此处不做详细说明。

冷启动问题: 58 同城服务于多业务，包括招聘、房产、黄页、二手等，求职者进入招聘板块使用招聘功能，由于当前不强制用户填写简历，导致无简历用户冷启动问题。

稀疏性&实时性：58 招聘的一部分群体为蓝领用户，他们在平台产生的行为是短时间的、连续的以及稀疏的，可能活跃两天找到工作后就不再活跃。其次，有些用户回到平台，求职意愿可能会发生变化，一部分可能想找别的工作（如之前是服务员，现在想找快递），另一部分可能是因为传统职业存在职位进阶的过程，这些都需要系统思考。

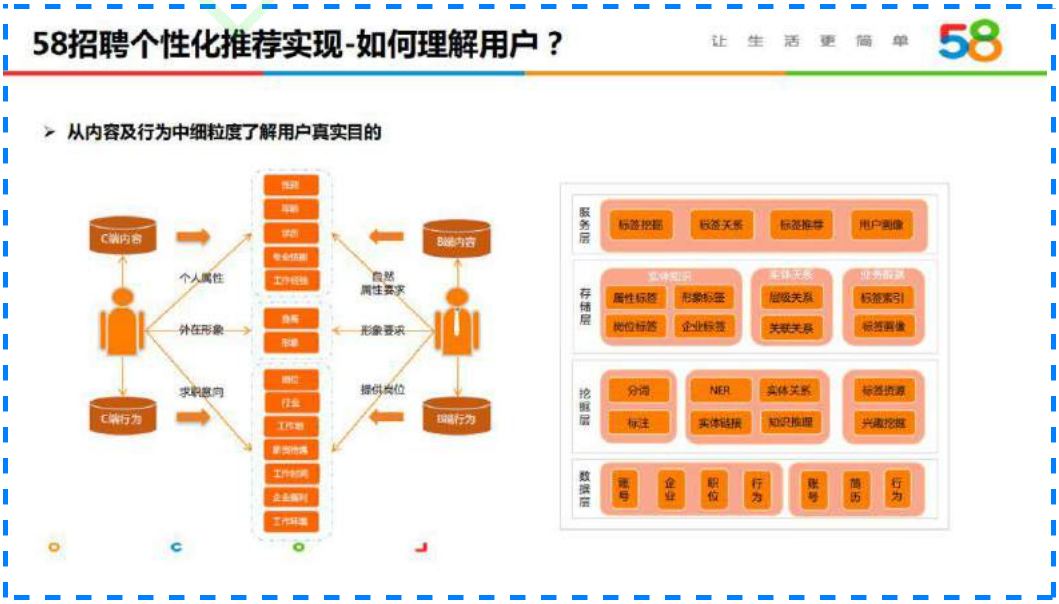
资源分配问题：第一，如何有效识别（企业，求职者）的真实意图，进而合理分配资源产生有效连接，针对不良意图进行差异化对待。第二，招聘对于 C 端和 B 端都是有限的资源，招聘方招聘职位有限，求职者与招聘方交互有限，很大程度上不同于淘宝推荐，因为后者的商品是无限供应的。

——招聘个性化推荐实现——

1. 58 招聘个性化推荐实现

58 招聘个性化推荐的实现过程与大多数公司推荐模块基本相似，包括用户意图理解、内容召回、内容排序、内容展示四个核心模块。下面将结合业务特性，介绍每个模块实现的关键点。

2. 如何理解用户？



58 招聘用户理解主要通过“言”和“行”识别用户真实意图，重点关注的属性主要包括招聘领域求职意向、用户个人属性以及外在形象（如上图左边）。围绕求职者与招聘方在平台产生的内容及行为，我们构建了相应的知识图谱和用户画像。

2.1 无诚意用户识别

在理解用户之前，我们首先需要识别出无真实招聘/求职意图的用户，并进行差异对待。如频繁发布包含联系方式的导流信息、发布高薪诱惑等恶意虚假信息等，将用户引导至平台外进行转化。针对以上业务我们总结了一些特点，主要表现为：

❖ 暴露联系方式

- 内容不成句
- 高薪诱惑
- 在平台很“活跃”

针对以上业务特点，我们主要的识别方法包括：

- 传统的关键词+正则识别方法，如针对"微信"、"QQ"这类联系方式的相关关键词等。
- 针对变形联系方式，基于拼音+滑动窗口进行识别。
- 采用命名实体 NER 识别进行挖掘，如 BiLSTM+CRF。
- 采用相关分类算法进行识别，如 fastText，CNN。

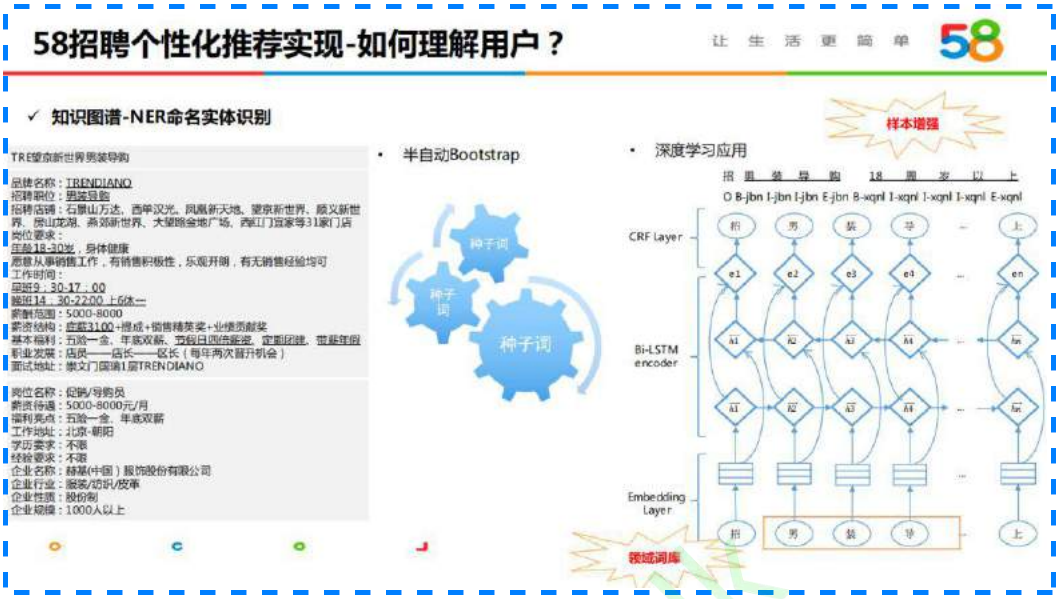
在无诚意用户识别过程中，我们总结了以下心得：

举一反三：问题用户识别是典型的对抗场景，策略构建时需要更多思考对抗能力的刻画，将一些强对抗能力的特征加入到模型中（如文字变形、文字转拼音）。

DataFunTalk

刚柔并济：差异化惩处不同问题类型的用户。对平台其他用户伤害巨大的群体，结合法律手段严厉惩处；处于问题边界的，则主要通过较柔和的方式处理（如内容展示降权），减少剧烈对抗现象的产生。

2.2 知识图谱构建



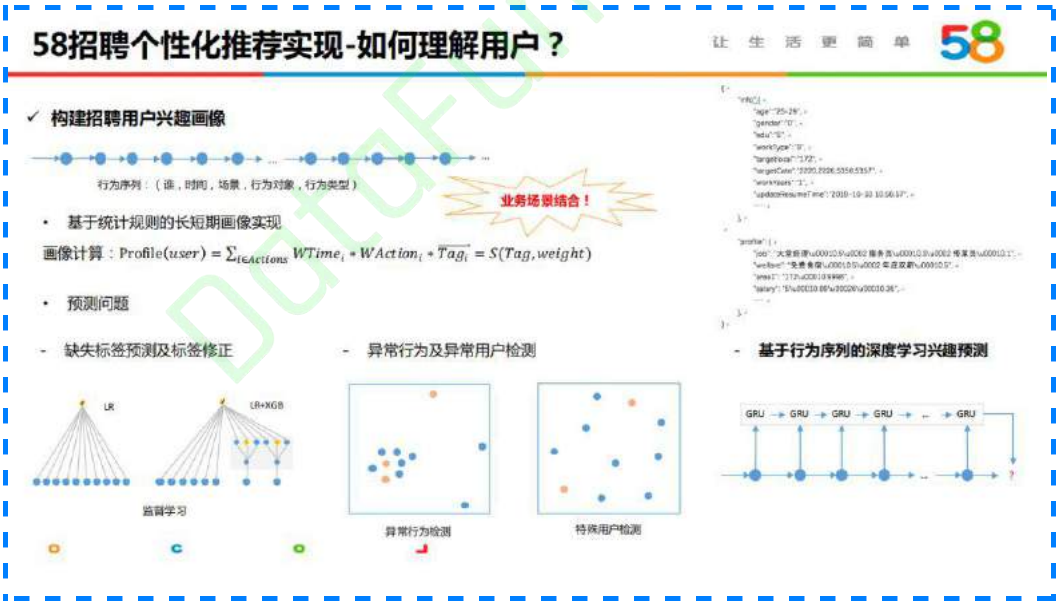
知识图谱是一个非常复杂的系统，包括多元异构数据搜集->知识获取->知识融合表示->知识推理->知识管理多个部分，主题及时间因素，我们重点介绍下在 NER 方面的探索。招聘业务场景含有大量的文本内容，通过 NER 技术能够有效提取文本中的关键信息，进一步提高系统的结构化理解能力。

NER 开展经历了两个阶段：

第一阶段：基于平台已有的部分结构化实体词，以及不少半结构化组织的职位描述基础，我们采用 bootstrap 方法，快速迭代进行挖掘，并结合半人工标注，为深度学习构建更完整的样本数据集。

第二阶段：将第一阶段的内容作为 input，核心采用 BiLSTM+CRF 构建实体识别深度网络，有两个优化点取得了较好效果。第一个是输入层基于字到词的优化，构建招聘领域的专有词库。第二个是采用训练样本增强技术，将相近实体词和同类实体词进行替换扩大样本集，并将模型识别的结果有选择的放回训练集重新进行迭代训练，减弱对标注数据集的依赖。目前命名实体识别仍在不断优化，识别准确率平均达到 0.75+，部分准确率可达到 0.9+。

2.3 构建用户画像



用户画像是个性化推荐系统的基础模块，决定了对用户意图理解的准确与否。基于标签传递思想，我们通过统计规则、传统分类模型和深度模型多种算法结合捕获用户行为的兴趣表达，构建长短期用户画像。

基于统计规则：通过窗口形式，近实时对用户画像进行计算更新，计算时加入时间衰减因子、行为权重因子及标签置信度权重。深刻理解业务场景，进行合理数学设计是关键。如信息列表页的点击数据，在使用时要差异化处理列表页直接展示的显性标签及隐藏在详情页的兴趣标签，避免人为引入噪音。

基于传统分类预测：采用分类算法，应用到用户属性填充、异常用户/行为识别及用户分类多个方面。并非所有的求职用户都会留下较详尽的简历，我们借助历史的招聘简历与用户行为组织样本，可有效预测性别、年龄段、期望工作岗位等用户信息，优化简历缺失或不完善的冷启动问题。同时，针对用户行为的聚焦情况，通过模型能够有效识别出一些异常数据、识别求职目的明确型及发散型两类求职用户，进而剔除掉部分噪音数据提高样本精度，对不同用户定制差异化策略，提升推荐整体刻画能力。

基于行为序列预测：借助统计规则及传统分类，基本建设出一个可用画像，但对用户多个行为之间的信息捕获有限。我们将用户搜索浏览、简历投递、在线沟通等行为组织成行为事件序列，采用 LSTM、GRU、Attention 等训练模型预测用户兴趣，当前还在探索评估阶段。

3. 召回模块

58 招聘推荐围绕个体、群体、全局三个召回不断细化演进，不同召回满足不同需要，三者结合服务于各类场景。从 2016 年到现在，我们先后主要经历了基于上下文内容、协同过滤、精细画像、深度召回几个阶段，演变成当前以上下文与用户画像结合的精准召回、协同过滤召回及深度向量化召回为核心策略的召回模块。

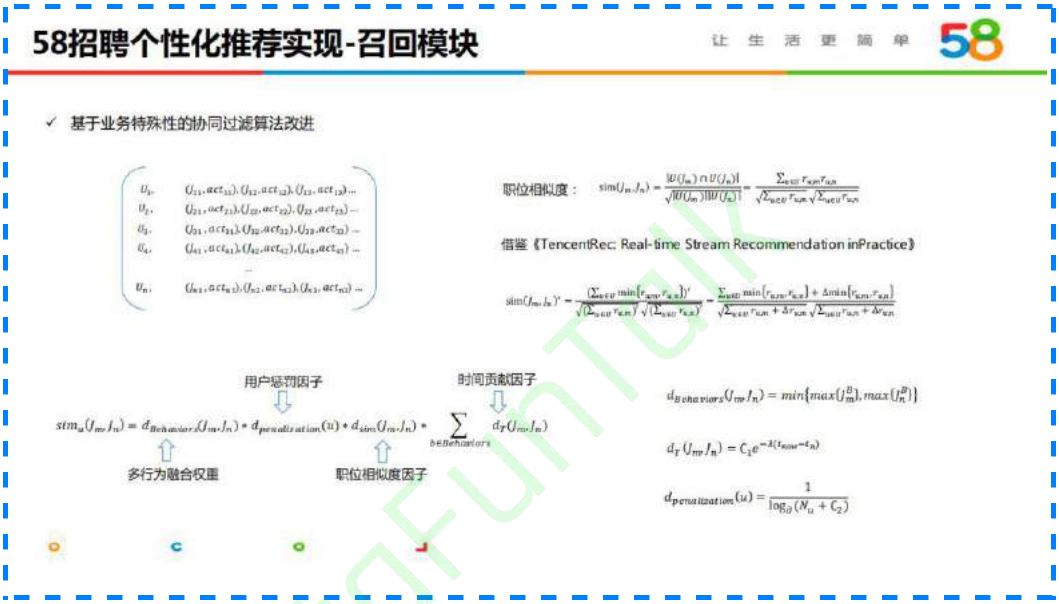
3.1 基于上下文+用户画像的精准召回

该策略是业内十分常用的召回方法之一 ,核心在于结合用户画像对请求进行丰富改写。绝大部分场景，用户主动搜索或点选的条件有限，借助用户画像中的历史兴趣及知识图谱组织的实体关系，我们对岗位、工作地、薪

资、行业等多个维度进行条件扩充或必要改写，多路召回匹配用户的职位内容。

该策略的主要优点：可解释性好、实现时间成本低，缺点和难点是过度依赖标签挖掘的准确性。

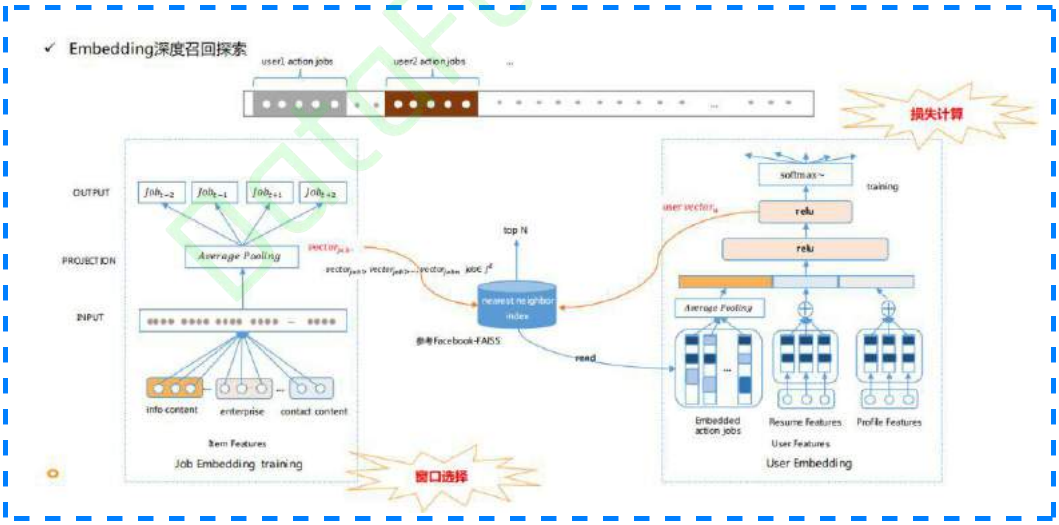
3.2 基于业务特殊性的协同过滤算法改进



协同过滤是推荐系统经典的召回方法，通过用户与物品的行为挖掘用户与用户、物品与物品之间的关联关系。招聘业务的求职者数量巨大，且是短时间的稀疏行为场景，我们采用基于物品的协同过滤，同时希望能近实时的将实时行为信息组织进服务。

在技术实现过程中，我们参考了腾讯 2015 年发表的 Paper《TencentRec: Real-time Stream Recommendation in Practice》，赋予职位点击、投递、在线沟通等不同的行为权重进行多行为融合，基于用户行为序列的长度以及用户质量设计用户惩罚因子，同时通过时间衰减因子增强近期行为的表达，这三个因子的设计与 Paper 基本一致。另外针对业务特殊性，我们改进了职位相似度的计算，加入了职位相似度控制，避免求职目标发散的用户影响职位关系的组织。算法上线后，在点击率、投递率方面都取得了正向收益，其中详情页的相关职位推荐提升超过 25%。

3.3 Embedding 深度召回探索



协同过滤虽然取得了不错的业务收益，但其依赖于用户与物品的行为矩阵，对于行为稀疏的场景天然表达有限。而恰好，58 招聘业务的流量构成中，有一部分是三四五线城市，城市越下沉数据稀疏的情况也越凸显。

针对这类问题，我们希望进一步挖掘行为数据的信息，很自然的想到基于深度学习的向量化 Embedding 召回。我们核心参考了 Youtube 的 DNN 召回思想，基于业务现状做了调整优化。

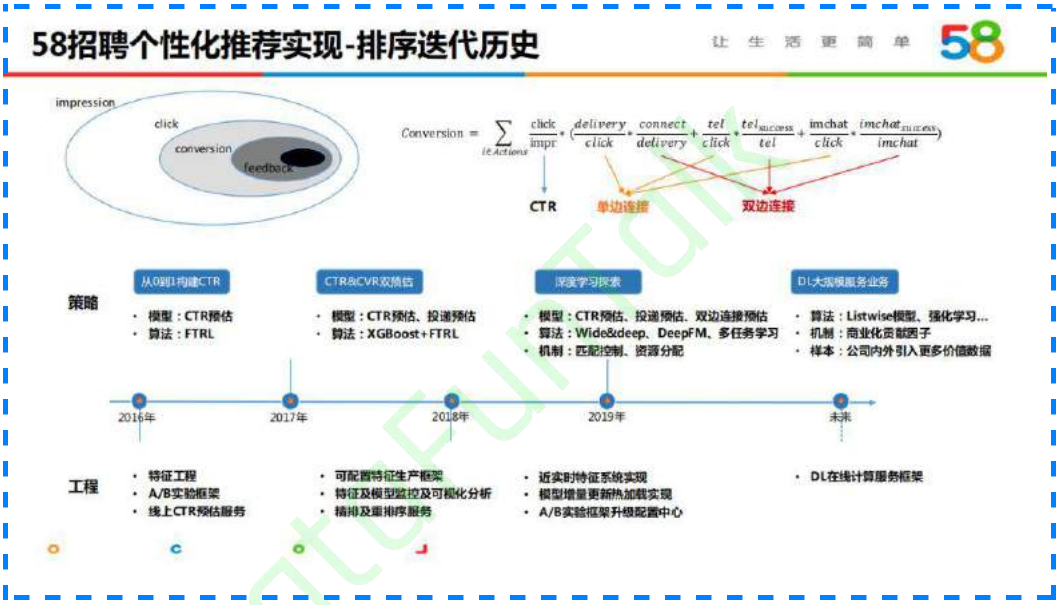
职位向量化：我们将求职者对职位的行为序列看作一系列上下文，利用 word2vector 思想进行向量化表达。Input 部分，包括职位特征、职位所属的企业特征和求职者反馈特征。Output 构建，业务漏斗越深的行为选择的窗口越大，并基于用户平均的行为长度作为窗口设定的参考值。针对无历史用户行为的新职位，使用职位的文本结构化信息，通过历史训练所得的标签向量表达经过 average-pooling 作为初始向量，解决冷启动。

用户向量化：构建一个多分类 NN 网络，Embedding 层将用户发生行为的职位向量化直接迁移过来使用，输入用户的简历及画像信息进行向量训练。最上层理想情况是一个极限分类，以用户真实发生行为的数据作为正样本，未发生行为的数据作为负样本，构建损失函数进行最优化训练。58 招聘场景有千万级别的职位，极限分类需要巨大的计算消耗，当前资源无法满足。因此在负样本选择上，我们使用降采样机制，随机从求职者关注的城市及岗位下未发生行为的职位中按一定比例抽取负样本。线上会实时的采集用户行为，以窗口形式对用户向量进行更新。

线上服务：借鉴 Facebook 的 FAISS 实现，线上用户发起请求时，通过求职者的向量表达，去获取与其最相似的 TopN 职位，返回给推荐系统。

Embedding 向量化召回，还处于初期探索，仍需要在样本、输入特征及网络参数调优开展大量工作，期待有更显著的业务收益进一步与大家分享。

4. 排序迭代历史



相比其他推荐场景，58 招聘的漏斗更深，并且是典型的双边业务。系统不断优化提升求职者点击、投递职位的同时，还需要关注职位背后的招聘方是否反馈形成了有效双边连接，进而达到更接近求职链条的预测目的。结合不同时期的业务目标，我们先后经历了几个主要阶段。

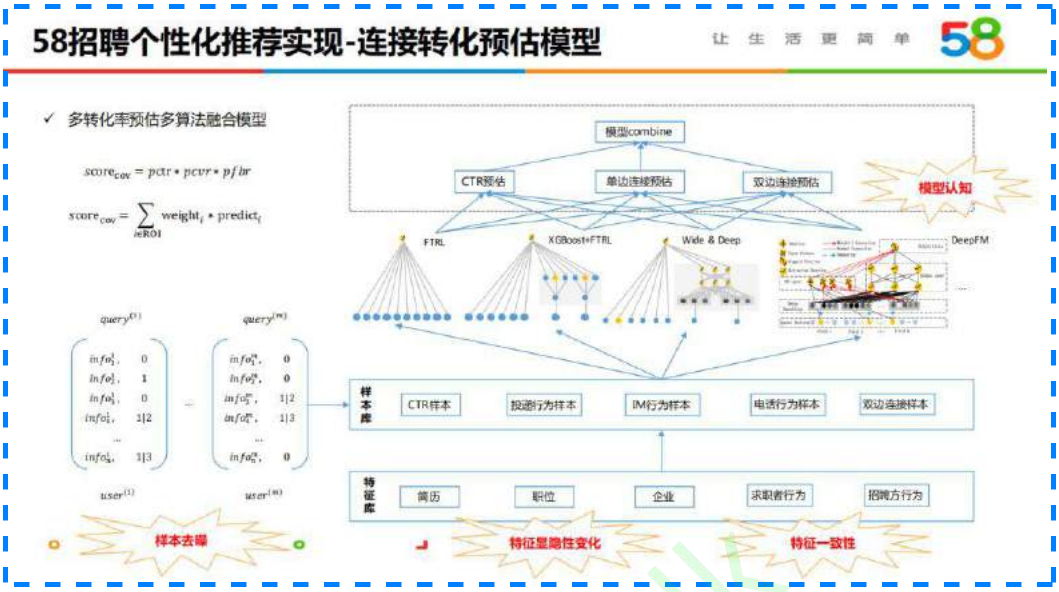
第一阶段：以提升点击规模为主要目标，从零到一构建点击率预估模型，开发模型建设的基本框架，包括特征工程、AB 实验框架及线上 CTR 服务。该阶段在较少人员的情况下，建立了排序模型及服务的大体框架，在点击层面支撑业务增长。

第二阶段：业务目标深入，从点击过度到单边连接直至双边连接，在 CTR 预估模型的基础上，增加了 CVR 预估及 ROR 双边连接预估。同时在工具上开展了针对性建设，包括特征生产 Pipeline、AB 实验框架升级为可配置化中心及特征模型的可视化分析监控等，解耦算法和工程依赖，支持更多算法和工程人员的并行高效迭代。

第三阶段：围绕深度学习的算法探索，wide&deep、DeepFM、多任务学习、强化学习等，不断提升算法对高维特征的表达能力，提高预估模型的刻画能力。预计在 2020 年全面落地业务，达到更为理想的迭代状态。

4.1 连接转化预估模型

DataFunTalk



58 招聘转化预估模型是多目标学习，设计实现如上图，底层共建样本及特征，使用不同算法对 CTR 点击率预估、CVR 单边连接预估、ROR 双边连接预估进行建模，最后对多个模型进行融合支撑线上排序。

整体排序实现是业务常见的方法，总结开展过程中比较关键的点：

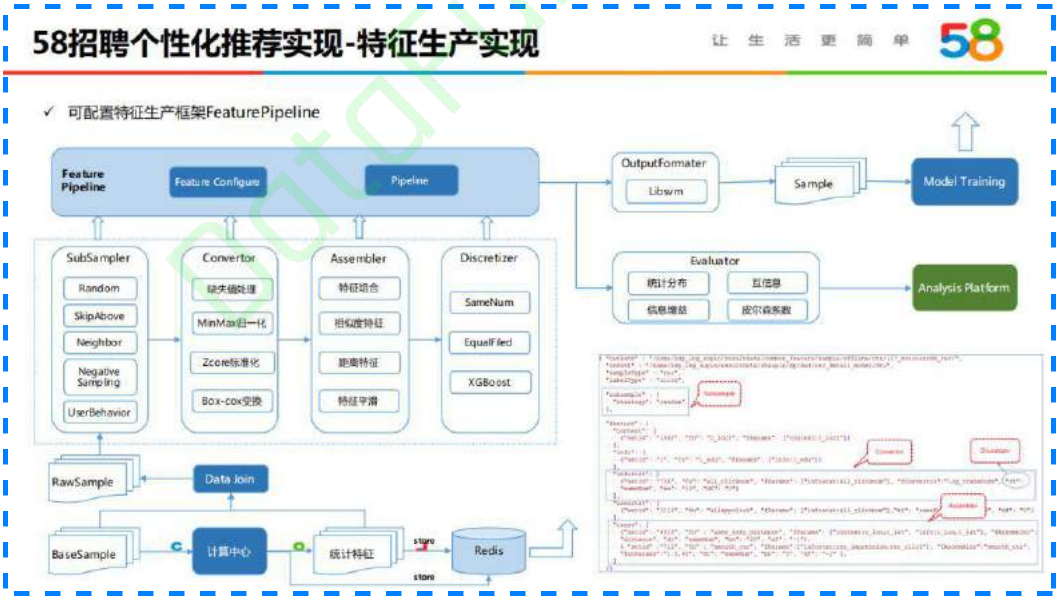
样本处理：围绕减少样本噪音，我们开展了多个优化。去除异常用户及异常数据，包括非招聘意图的用户数据、误点击数据；增加真实曝光及停留时长埋点，去除用户下拉信息流过程中非真正看见的数据，将停留时长作为样本置信权重加入到模型训练中；基于求职者维度进行采样，去除对同一职位多次正负样本的矛盾可能。

特征工程：关注及监测特征显隐性的变化，尤其是信息列表展示样式的产品调整，需及时进行特征调整及模型迭代。58 招聘业务的特殊性，实时

类特征很重要，需要关注特征一致性方面的保障机制，避免发生特征穿越现象或线上线下特征不一致问题。

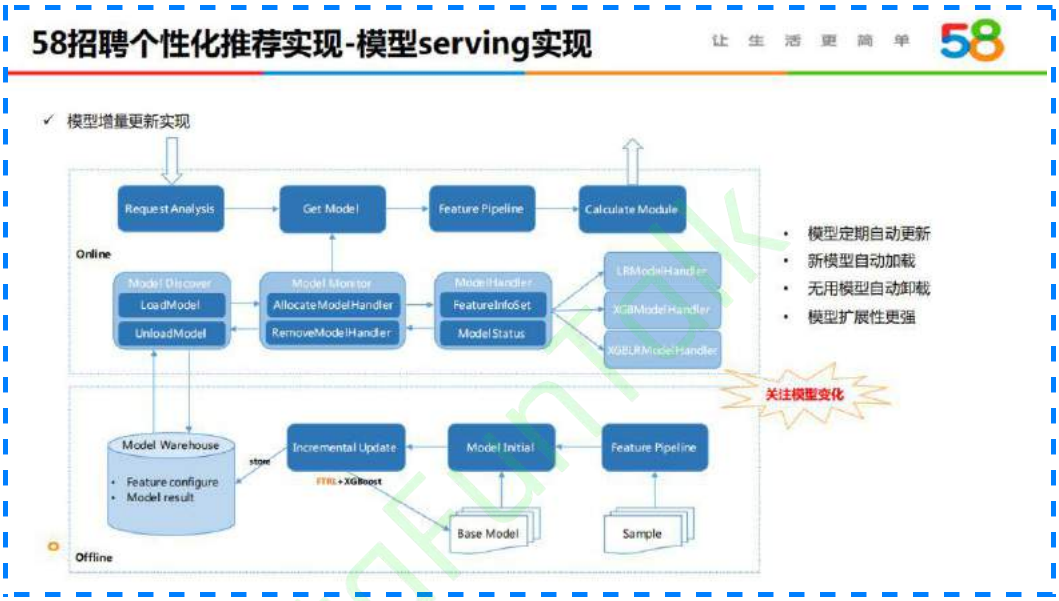
模型：重视模型认知，并不是简单的关注离线 AUC 或者线上转化率 AB 对比，在特征表达上多些分析，迭代过程中重视前后模型的特征比较，能够有效提高模型实验迭代的有效性。

4.2 特征生产实现



特征 Pipeline 的构建，减少了大量特征工程重复工作，显著提高模型迭代效率。其核心功能是实现配置化的方式，集成了样本采样、特征变换、特征组合、特征离散化，整合后得到训练样本，一方面输送给模型进行训练评估，另一方面也输出到分析平台支持可视化分析。

4.3 模型 serving 实现

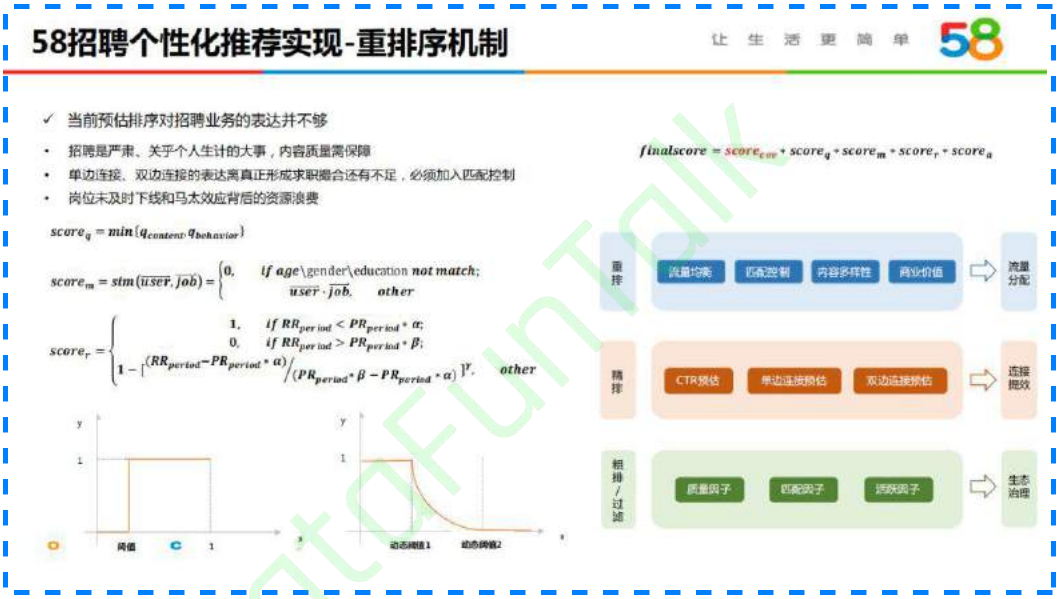


线上模型服务有定期更新及大量 AB 实验的要求，随着服务演进构建了当前的模型 Serving 框架，实现了对模型的定期自动更新以及模型的自动加卸载功能，同时也具备了更强的扩展性，可接入不同算法的模型实现。离线部分，样本经过特征 Pipeline 构建增量训练数据，模型训练模块会获取 Base 模型文件初始化并进行增量模型训练，模型评估无异常，系统会将模型存储至模型仓库及 HDFS 文件。线上部分，模型仓库增删改模型后，会发起模型热加载或卸载指令更新至线上服务内存；对于线上的排序请求，实时修改相应使用模型的存储生命周期，对于长期无用的模型，模型仓库将自动删除。

模型 Serving 能够自动化管理线上模型，但我们也不能完全托管系统，依然需要关注模型变化。一方面在离线部分的模型评估环节，除了对 AUC 等评

估指标的自动监测，也将模型内存大小、模型特征表达作为监测的一部分；另一方面线上监测业务转化指标的变化，当指标发生较大波动时发出警报，人工进行模型检查。

4.4 重排序机制



由于业务的特殊性，以 CTR 预估、CVR 单边连接预估、ROR 双边连接预估支撑排序仍然存在刻画能力不足的问题，体现在以下几个方面：

招聘关系到个人生计及国家民生，是件极为严肃的事情，内容质量是基础保障。但连接预估模型无法有效刻画质量问题，存在一些职位连接效率很不错但属于问题职位，因此推荐系统需要增加质量相关的因子。

转化率高不等同于双边匹配。线上招聘，无法很好追踪到面试及入职环节，求职者与招聘方形成的双边连接，可能是出于其他原因（如对自己或对方的错误判断）。因此，系统需要考虑匹配方面的控制。

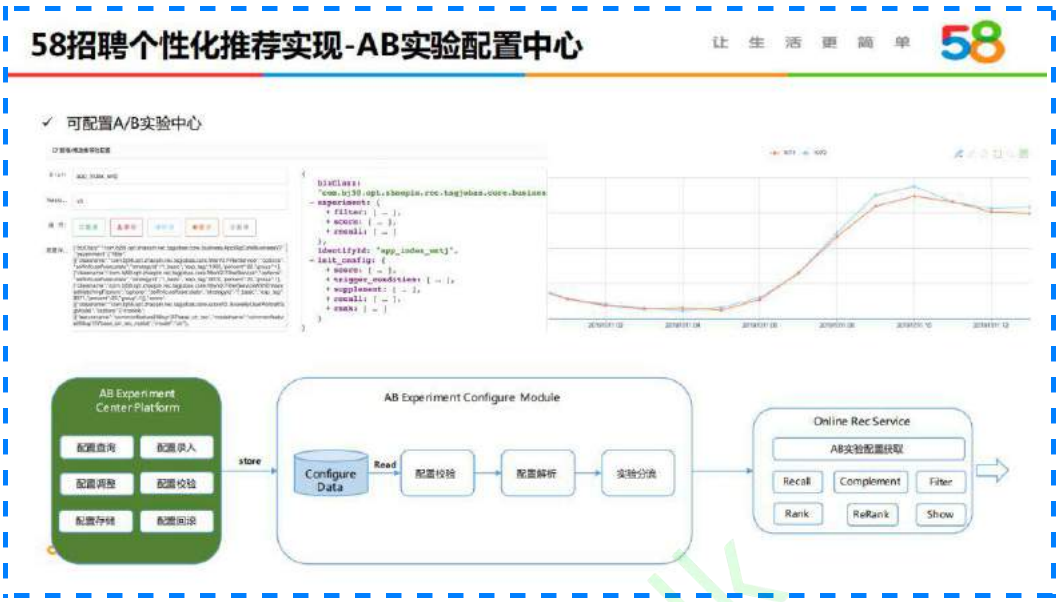
资源浪费问题，对于绝大部分用户，求职及招聘都是周期性行为，一个已经招满人的职位可能依然在线上展示。系统还需要增加职位活跃度或周期方面的刻画，减少相应的资源浪费。

针对这些需要，系统增加了重排序机制，通过分段处理手段，在粗排阶段打压甚至过滤掉低质量内容，在重排序对不活跃/不匹配内容进行降权，达到保障平台质量生态、提高有效连接规模的目的。

5. 列表展示内容控制

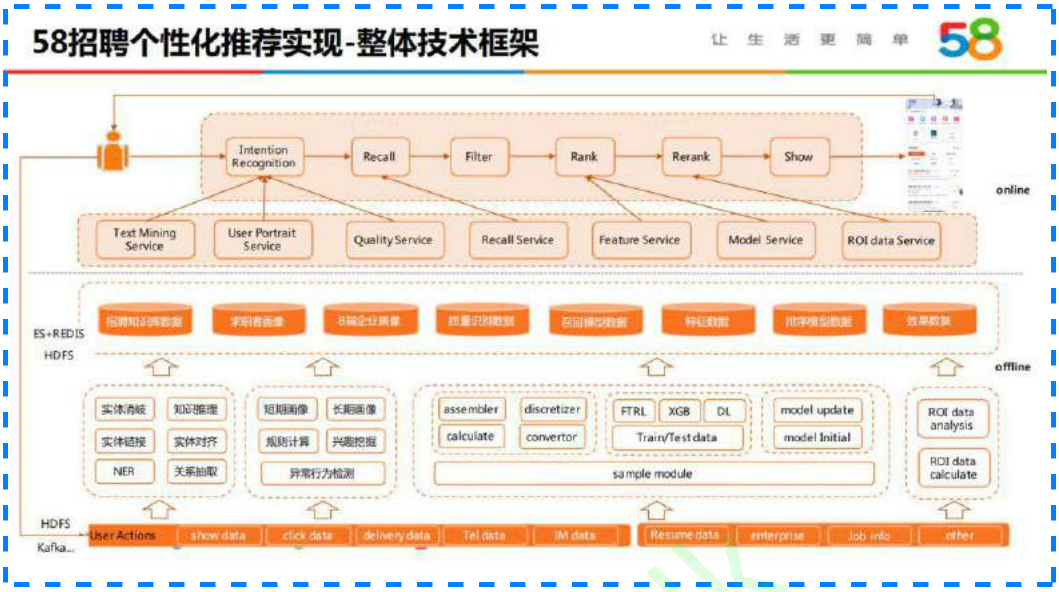
内容展示方面，我们也结合算法做了一些工作，来提高内容的可解释性、提供更多有价值的信息来辅助用户决策。结合个性化模型挖掘亮点标签，将更深预估模型的核心特征包装成标签形式展示在列表页，如距离多远、职位的福利标签、职位的热门情况等；使用 NLG 文本生成技术，自动生成简短描述进行展示，弥补标题及其简单职位的文本信息不足。

6. AB 实验配置中心



推荐系统包括召回、过滤、排序、展示几个核心模块，且每个模块都有长期进行实验迭代的诉求。我们搭建了 AB 实验配置中心，实现可视化配置，与线上服务及数据分析平台联动，更灵活高效地开展实验迭代工作。

7. 整体技术框架



58 招聘个性化推荐经过不断演进，最终形成了如上图的技术框架。离线部分包含数据仓库层，知识图谱、用户画像、预测模型的挖掘层，知识数据存储层；线上部分包含数据服务及推荐引擎。线上产生的行为数据，实时流转至离线的计算挖掘模块，反馈到线上达到个性化体验效果。

——心得分享及规划——

58 招聘推荐系统最近四年优化收益整理如上图，贡献大小依次是召回、特征、数据、算法、样式、工程。深入理解业务及算法、注重细节积累是做好算法工作的保证；前期在样本及特征上多下功夫，不仅能获得不错的业务增长，也是之后算法深入的基础；工具性建设尽可能先行，能够提高整体迭代效率。

未来的核心工作：

多任务学习、强化学习等的全面探索落地。

集公司内外资源，丰富招聘数据源，提高用户画像的覆盖率，更好的支持千人千面。

RALM: 实时 Look-alike 算法在微信看一看中的应用

嘉宾：刘雨丹　编辑：Jane Zhang

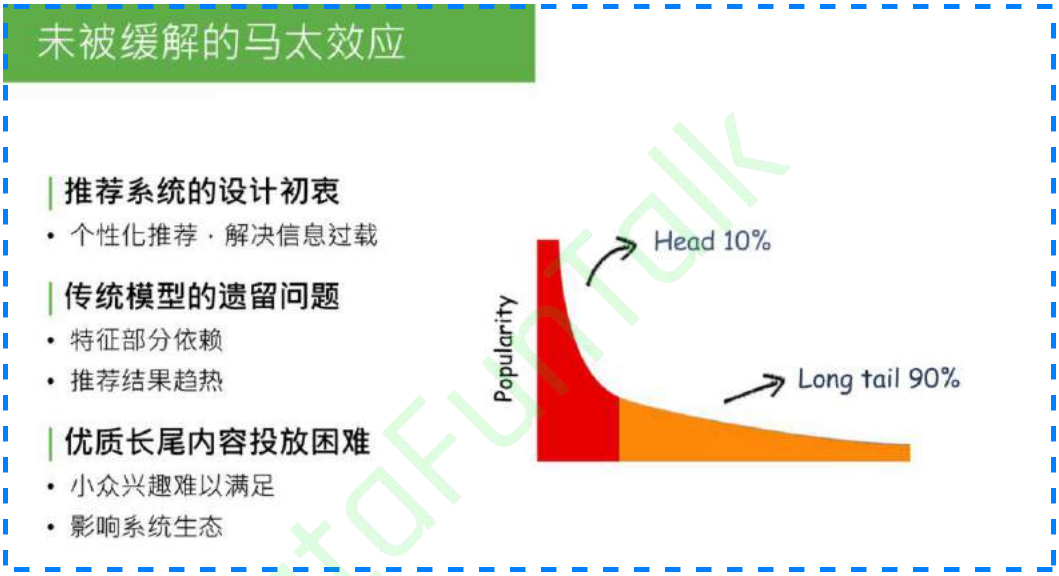
导读：本次分享是微信看一看团队在 KDD2019 上发表的一篇论文。长尾问题是推荐系统中的经典问题，但现今流行的点击率预估方法无法从根本上解决这个问题。文章在 look-alike 方法基础上，针对微信看一看的应用场景设计了一套实时 look-alike 框架，在解决长尾问题的同时也满足了资讯推荐的高时效性要求。

背景

微信大家可能都用过，微信中的“看一看”是 feed 推荐流的形式，涵盖了腾讯整个生态链的内容分发平台，包括腾讯新闻、公众号文章、腾讯视频等。

每天总分发量在千万级以上，面对如此大的分发量，要满足不同兴趣偏好的用户需求，使用传统的方法时遇到了一些问题。我们针对发现的问题做了优化和改进，接下来分享下我们优化的过程。

未缓解的马太效应



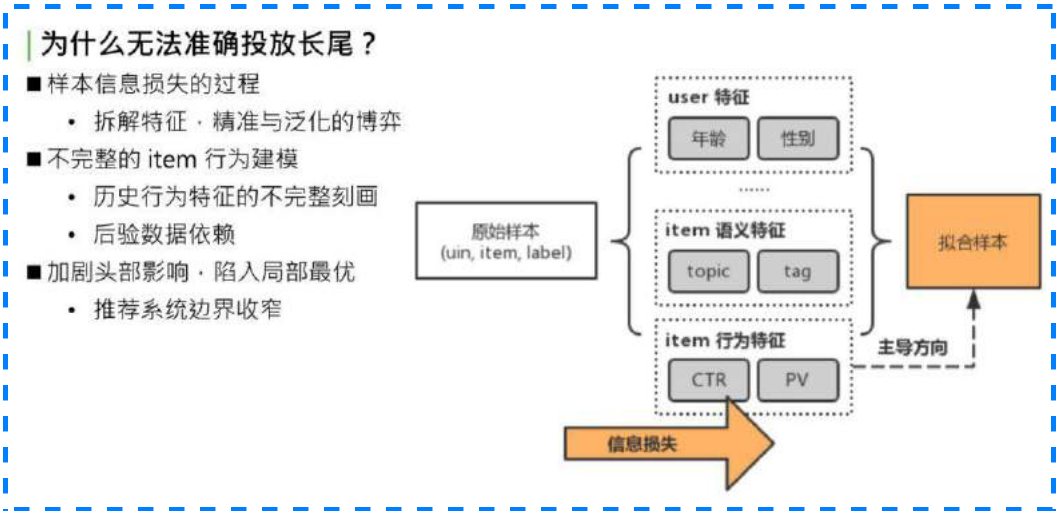
马太效应，简单解释，在内容的生态系统中，自然分发状态会造成一种现象：头部 10%的内容占据了系统 90%的流量、曝光量 or 点击量，剩下 90%的内容，集中在长尾的 10%里。这对于内容的生产方、内容系统的生态和使用系统的用户来说，都是不健康的状态。造成这种现象的原因，是因为系统分发能力不够强，无法处理信息过载的现象，推荐系统设计的初衷就是为了解决马太效应问题。

回顾推荐系统的发展，从最开始的规则匹配 -> 协同过滤 -> 线性模型 -> deep learning，逐步缓解了马太效应现象，但没有完全解决。

造成这个现象的原因是传统模型、CTR 预估和 deep model，都对部分特征有依赖，没有把特征完全发掘出来，导致模型推荐结果是趋热的，使生态系统内优质长尾内容投放依然困难。因为 CTR model 最终趋向于行为特征，或者后验结果较好的数据，对于优质长尾内容，如小众兴趣的音乐、电影、深度报道的新闻专题等，获得的相应曝光依旧困难，处于马太效应 long tail 90%的部分，这会影响推荐系统的生态，导致推荐系统内容越来越窄。

为什么无法准确投放长尾？

怎样解决这个问题？这个问题归根结底是对内容的建模不够完整。我们尝试分析下问题出在哪：



先看下推荐系统建模流程。首先得到原始样本，这是业务下的训练数据，形式是三元组：userid，itemid 和 label。如果是 timeline 的样本，那就是点击或者不点击。原始样本中，一条样本可以完整表示一个用户在某个时间点对一个 item 产生了一次行为，把这个三元组当作信息的最完整形式。对于这个完整形式，直接建模很简单，如传统的 item CF，或者协同过滤。协同过滤是最初级的方法，直接对 uid，itemid，label 做拟合，因为可以完全利用初始样本的信息，拟合的准确性非常好。弱点也很明显，对原始样本中没有包含的 userid 或者 itemid ,没有泛化推理能力 ,后续新曝光的 user 和 item 是无法处理的。这个问题，就是我们要做的第二步骤，对原始样本做抽象。既然无法获取所有的 userid 和 itemid ,那就要对 user 或者 item 做一层抽象，如 user 抽象成基础画像：年龄、性别或所处地域；item 抽象成语义特征：topic、tag 等；item 历史行为特征，简单做统计：过去一段时间的点击率、曝光率、曝光次数。最后基于泛化过的特征做拟合，得到最终模型。

问题出在哪？做原始特征抽象，抽象意味着发生了信息损失，这部分信息损失导致模型拟合时走向了比较偏的道路。举个简单的例子：同一个 item，有相同的 topic tag，历史点击率和历史曝光次数和点击次数也相同，可以说这两个 item 是相同的吗？显然有可能是不同的。使用统计特征无法完整表达，同样的 item 点击都是 0.5，PV 都是 1000 or 2000。有些 item 被这群用户看过，有些 item 被那群用户看过。尽管语义特征和行为特征都相同，但

两群 user 不同，Item 的受众也不同。这里说的抽象的方式，是不完整的 item 行为建模，也是对 item 历史行为不完整的刻画，这就导致了整个 model，对 item 后验数据十分依赖，导致推荐结果趋向于 CTR 表现好或者 PV 表现好的 item。最终后验数据表现好的数据又会更进一步被模型推荐且曝光，这样会造成恶性循环：一方面，加剧了头部效应的影响，使模型陷入局部最优；另一方面，整个推荐系统边界效应收窄，用户趋向于看之前表现好的数据，很少看到能拓宽推荐系统边界或者用户视野的长尾数据。

Look-alike 模型

问题就是这样产生的， 可以思考一下，问题的本质是什么？就是因为模型无法对 item 行为完整建模，这一步信息损失太大，怎么解决这个问题呢？我们首先想到了一种方案：look-alike。

这是广告领域的经典方案，这类模型的方法也很简单，首先可以有一个候选集合的 item，我们要推这部分 item，怎么推呢？第一个步骤：找到历史上已知的、广告主提供的对 item 表达过兴趣的用户，这部分用户称为种子用户。然后使用用户相似度法方法，找到和种子用户最相似的目标人群，称为目标用户，把这部分 item 直接推给目标用户。这个方法在广告系统中，是用来做定向投放的，效果很好。为什么呢？我们来看下模型的整体思路。

把相关的 item 找到对它发生过历史行为的种子用户，直接用种子用户的特征，作为模型的输入，这是正样本；从全局用户中负采样一部分用户作为负样本。用历史行为的用户的特征来学习 item 的历史行为，相当于把不同用户看过的 item 区分开，其实是对 item 的历史行为特征的完整建模。之前提到，行为样本是信息量最大的样本，它们没有经过抽象，如果能完整的用受众用户的行为来计算 item 的特征，可以说是最完整的 item 历史特征的建模。

Look-alike 在广告领域的应用已经很完善，也有很多方式。可以把 look-alike 相关的研究分成两个方向：第一种是基于相似度的 look-alike，这种 look-alike 比较简单，大体思路是把所有用户做 user embedding，映射到低维的向量中，对它做基于 k-means 或者局部敏感 hash 做聚类，根据当前用户属于哪个聚类，把这个种子用户的类感兴趣的内容推给目标用

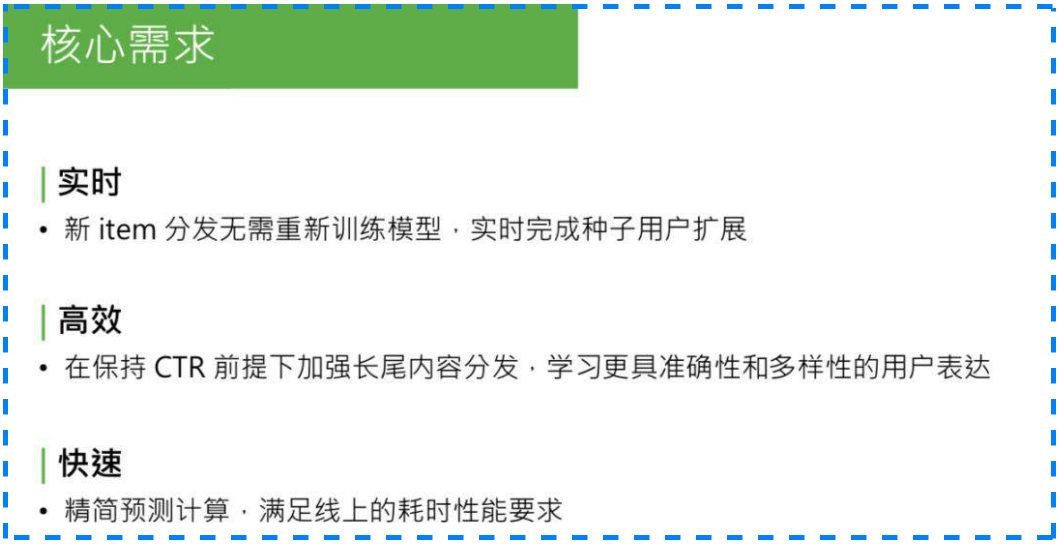
户。这种方法的特点：性能强。因为简单，只需要找簇中心，或者向量相似度的计算，因为简单、性能好，模型准确性低。

第二种是和第一种相反的，基于回归。包括 LR，或者树模型，或者 DNN or deep model 的方法，主要思路是直接建模种子用户的特征。把种子用户当做模型的正样本， 针对每个 item 训练一个回归模型，做二分类，得出种子用户的特征规律。这种方法的优点是：准确性高，因为会针对每个 item 建模。缺点也明显：训练开销大，针对每个 item 都要单独训练一个模型。对于广告来说，可以接受，因为广告的候选集没有那么大，更新频率也没那么高。

但是对于我们的推荐场景，有一些问题：1. 对内容时效性要求高，如推荐的新闻专题，必须在 5 分钟或 10 分钟内要触达用户；2. 候选集更新频率高，我们每天的候选集上千万，每分钟、每一秒都有新内容，如果新内容无法进入推荐池，会影响推荐效果。

核心需求

DataFunTalk



在我们的场景下，如果还用广告领域的经典的 look-alike，是无法解决的。如果要对每个候选集建模，采用 regression-base 的方法，如每分钟都要对新加进来的候选集做建模，包括积累种子用户、做负采样、训练，等模型收敛后离线预测 target user 的相似分，这对于线上的时效性是不能接受的。

对于 similarity base 的方法，它的问题是计算过于简单，如果直接和 CTR 模型 PK，核心指标会下降，得出来的结论是：传统的 look-alike 不能直接照搬到我们的系统中。

针对我们的需求，我们整理出来了应该满足的 3 点核心需求：

1. 实时。新 item 分发不需要重新训练模型，要能实时完成种子用户的扩展；

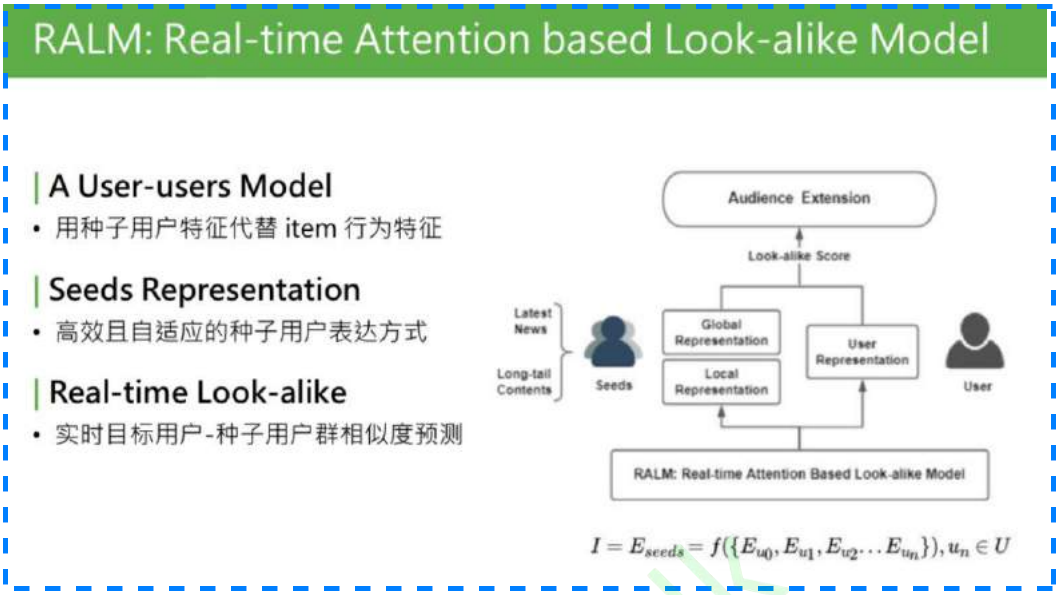
2. 高效。因为线上加到 rank 模型 CTR 的后面， 要保持模型核心指标 CTR 的前提下，再去加强长尾内容分发，这样模型才有意义。要学习准确性和多样性的用户表达方式。

3. 快速。Look-alike 模型要部署到线上，实时预测种子用户和目标用户群体的相似度，要满足线上实时计算的耗时性能要求，也要精简模型预测的计算次数。

RALM：Real-time Attention based Look-alike Model

基于这三个核心需求，我们提出了一个新的方法，全称是 real-time attention based look-alike model，简称 RALM。我先简单讲下 RALM 核心的三个点。

1. 核心点



① 模型可总结为 user-users 的 model。回想下经典的 CTR 预估模型，是 user2item 的 point-wise 的处理流程建模。User、item、label，我们做的最大的变化，是借鉴了 look-alike 的思想，把 item 替换成种子用户。用种子用户的用户特征，代替 item 的行为特征。所以模型从 user2item 的 model，变成 user2user 的 model。图中右侧是 target user，左侧是 seeds。

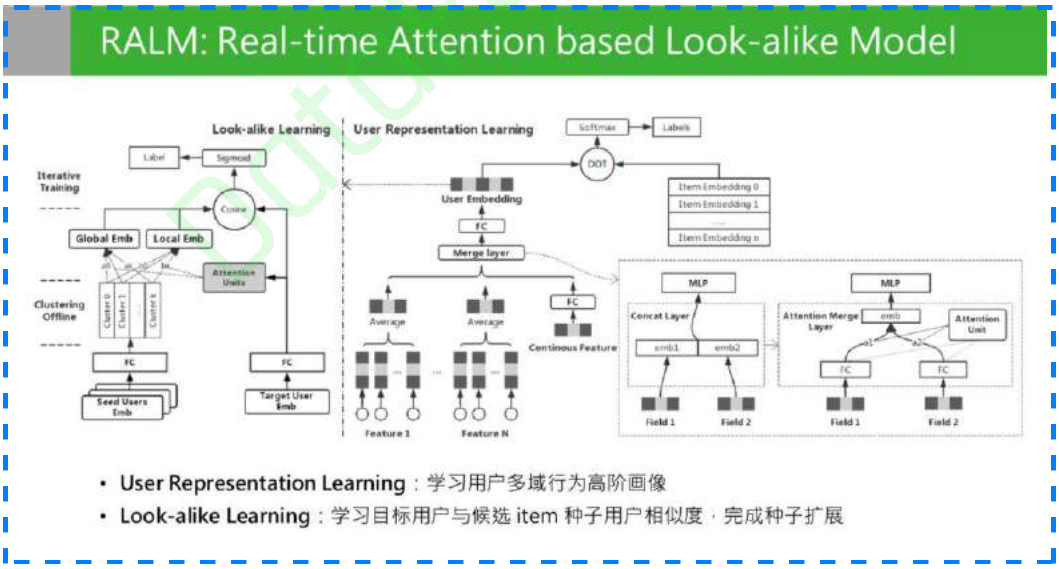
② 完善的 seeds representation。用种子用户代替 item 行为特征。这样面临的问题是：怎样更好地表达一个人群。这个 seeds representation，是我们研究中的核心步骤，要得到一个高效、自适应更新的种子用户的表达方式。

③ real-time。最终目标是部署在线上，实时预测种子用户群体相似度，需要是能够实现 real-time 的框架。

上述是模型表达的思路。I 是一个 item，把 item 用 seeds 的 embedding 的集合来表示，seeds embedding，是组成这个种子用户的每个用户的 embedding 的函数。学习了 seeds representation，就是这个函数 f。

2. 整体结构

接下来看下离线训练部分，这是离线训练的整体结构。



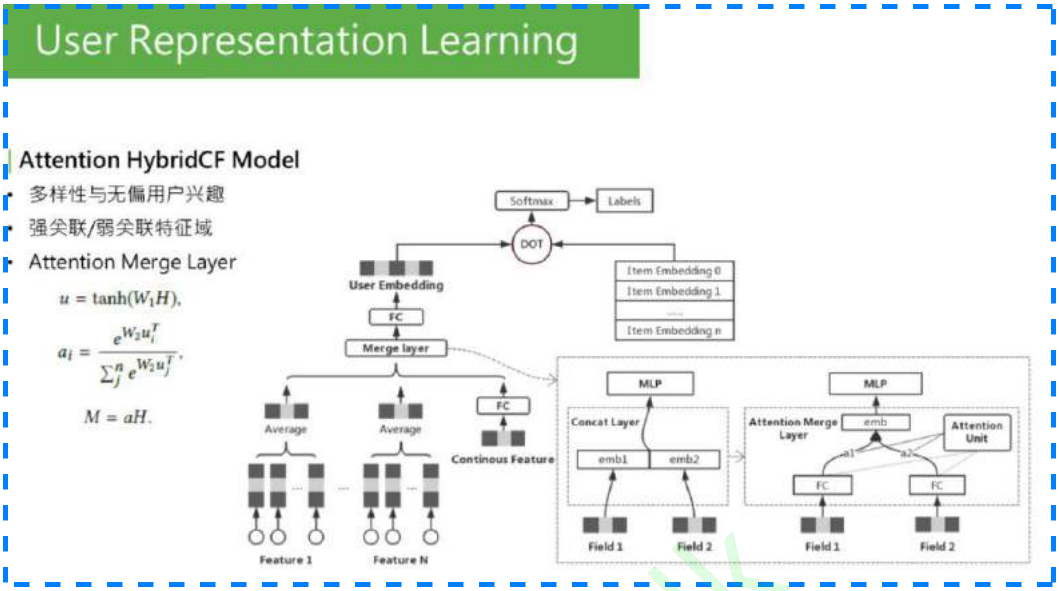
模型离线训练分成两个阶段：右侧 user representation learning， 左侧第二阶段是 look alike learning。user representation learning 模型结构，最后的目标是通过一个用户在不同领域的行为，学习到用户在所有领域的多

样性且兼顾准确性的用户兴趣的高阶画像。这个画像在这个位置是低维特征，向量特征通过 user presentation learning 的目标学到了所有用户的 embedding 之后，第二阶段是 look alike learning。Look alike learning 模型，是一个 user to user 的 model，右侧是目标用户的特征输入，左侧是种子用户人群的 embedding 输入，左边种子用户是一群用户的 embedding 堆叠到一起，输入其实是一个矩阵。这两边的输入来源都是第一阶段 representation learning 输出的 embedding。Look alike 的目标是学习目标用户和候选 item 种子用户的相似度，最上面是学习两次相似分的，最后完成种子用户的扩展。

User Representation Learning

按顺序来分析下，第一阶段，是用户的表示学习，user representation learning。

DataFunTalk



这个模型大家看着会比较眼熟，它是用 Youtube 的 representation model 中演化过来的。Youbute 的基础模型很简单，下面是用户在不同领域的行为，下面的基础特征可能会有离散值，也可能是连续值。如果是离散值，可以通过 embedding lookup，再过一个 pooling，再和所有领域的特征做 merge，上面过一个全连接，最后输出 embedding。右侧是感兴趣的 item，也会做一些 embedding lookup，整个做 sce loss，或者是多分类。要预测的是：用户在点击了这么多 item 之后，下一个要点击的 item，最后要预测的就是表达用户兴趣的 embedding。这层 merge layer，最初 Youtube 的版本是用一个 concat。可以看到最初版模型在训练时遇到了一个问题，最下层是用到了用户很多个域（每个 field 称为一个域，可能是每个用户在每个分布下的行为，如电商购物下行为，或者是公众号阅读的行为）。

训练时看到一个现象，有些域的行为学的非常强，参数来看学的非常充分，某些 field 参数分布不大，最后的权重值较小，对最终预估的分数没有影响。这里有两个名词：强关联和弱关联。最终预估结果关系比较大的 field、参数学习较强的，称为强关联特征域；相反，学的不充分的、对最终结果影响小的，称为弱关联特征域。对于强关联和弱关联，如果看到参数分布是这样的，是不是就表明弱关联特征不重要呢？并不是。举例来说，representation learning 如果训练目标是在“看一看”中的阅读行为，对于某些经常使用微信公众号、或者阅读的用户来说，他们在公众号平台的阅读历史就是非常强的关联特征，能够决定再看一看中的兴趣。对于这些特征来说，这些特征是很强的，对于其他的如在电商中的购物或者是在搜索中的 query，这些是比较弱的，对看一看的影响很有限。再思考另一种 case，比如，看一看通过某种形式，吸引了很多新用户。新用户进来之后，没有在公众号平台的阅读历史，但是他们在购物或者搜索中有历史行为，此时这些历史行为会影响他下一次阅读的文章，或者感兴趣的 item。这些特征对这些用户来说是非常重要的。但目前，显然这些用户是没有学到这些变化的。

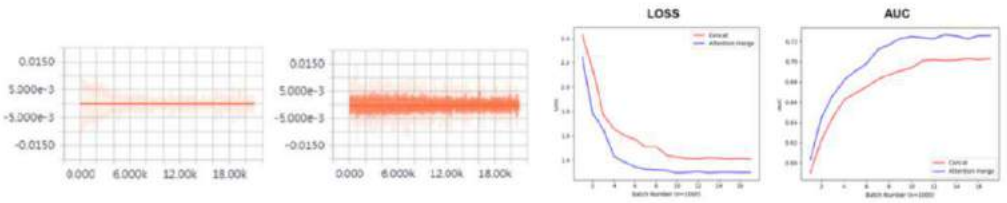
排查了下模型训练的过程，可以把结果集中在这一点上，就是这个 merge layer，其实是负责把用户不同域的特征 merge 到一起。Merge layer，可以看到右侧的图，原始的 deep model 用的是左侧的实现方法，直接用 concat。Concat 的优点是，可以学到所有 field 的参数，缺点是，无法根据输入的不同分布，来调整权重值。也就是说，如果 80%的用户的阅读历史

都是看一看的种子用户，阅读历史都是很丰富的，就很有可能对所有用户都把这个特征学的很强。如果是少量用户，就学不到了，少量用户关注对其它特征的启发作用，concat layer 是学不到的。因为它对于大部分用户来说，已经把参数学的非常重了，小部分用户不足以对它产生影响。所以需要一个机制，针对不同用户的特征域的输入动态调整 merge layer 的方式，我们想到的最好的办法是 attention。Attention 是最近在 NLP 中非常火的，很多模型都会用到。为什么要用 attetnion？

右下角的结构，就是 attention。我们用到的 attention 是把用户的输入的所有的域当做 attention 的 query，key 和 value 都是自身 field 的本身。这是一个典型的 self-attetnion，我们最后要做的是，让模型根据用户自己的输入领域的情况，动态调整不同领域的融合方式，相对于之前的 concat 的方式来说，concat 其实是把所有领域的 field 强行放在同一个向量空间中来学习，自然会有学习不充分的情况。Self-attenion merge 是让不同的域在自己的向量空间中学习充分，再通过不同的权重组合在一起。其实是相当于让用户能有属于自己的表达，而不是被历史丰富的用户带着走。这是一个优化，可以明显改善强弱特征、训练不均衡的问题。

可以看一下这是我之前训练的时候在某个特征域用 tensorboard 打出来的参数分布的情况，可以看到 attention merge layer 前后，训练参数有很大变化，之前这些参数基本上都是 0，之后会激活出一些值，这个是最后 user

presentat learning 的值：precession、recall、auc。也可以看到模型加完 attention 之后，在 auc 和 loss 上都有所优化。

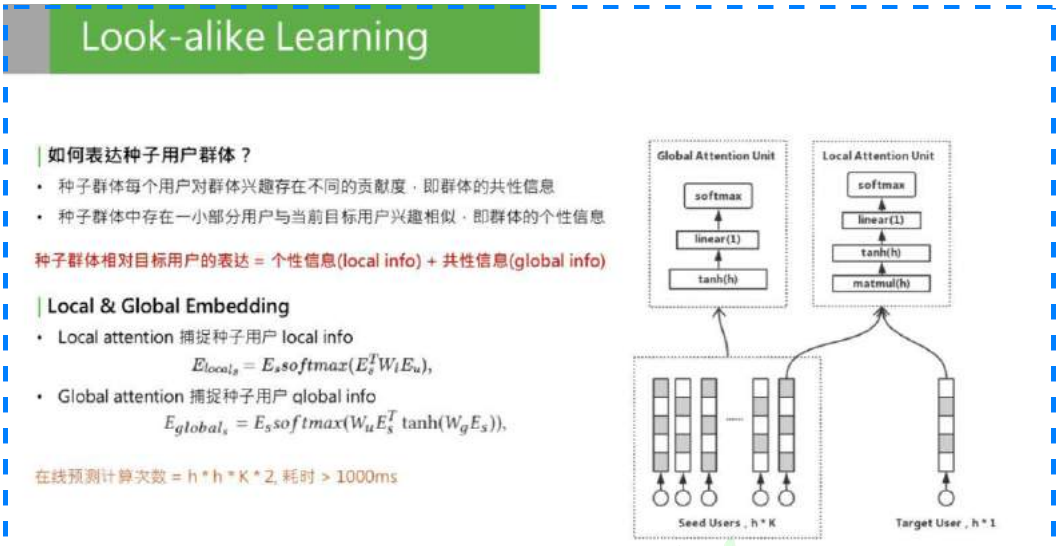


Concat / Attention Merge Layer 训练参数分布对比　　Concat / Attention Merge Layer 评价指标对比

经过 user representation learning 之后，我们现在拥有了所有用户的兼顾多样性和准确性的 embedding 表达。接下来要做的是怎么用 embedding 来表达种子用户人群？

Look-alike learning

Look-alike 要做的第一步就是如何表达 seeds user。



一个种子用户应该包含什么信息，这里我们做两点假设：

1. 每个用户都有自己的兴趣，但对整个群体的人群信息存在不同的贡献度，我们称为群体的共性信息：global info。共性信息和目标用户无关，只和用户群体自身有关。

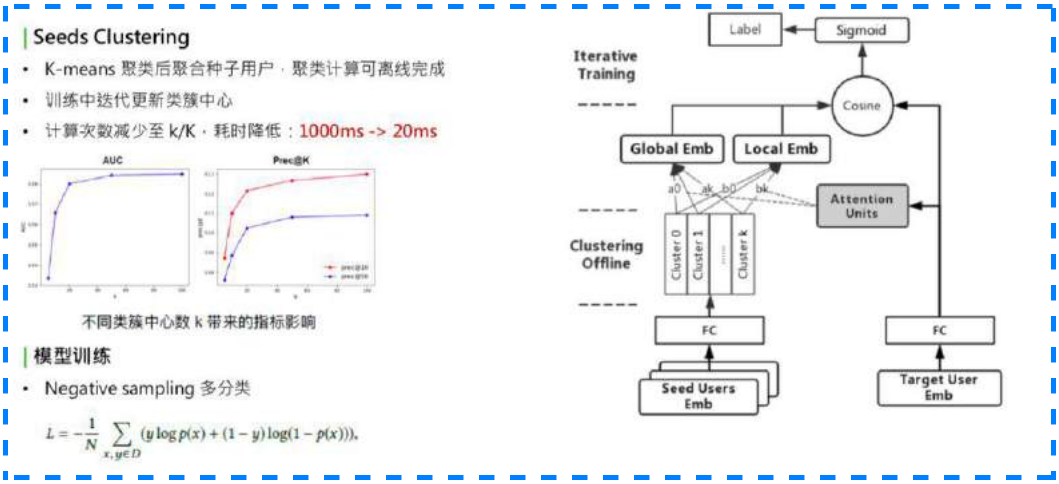
2. 种子用户群体的个性信息。种子群体中一定存在一小部分用户和 target 用户兴趣相似，这时，当 target 人群变化时，信息会变化，称为 local info。

种子用户的相对表达=个性信息+共性信息。怎样学习 local info 和 global info 呢？我们想到的是用不同的 attention 机制，学习出两个 embedding :local & global embedding ,分别表示这两种信息。对于 local embedding，是右上角的图，称为 local attention unit，这个 attention，是一个乘法开始，它的公式是把种子用户的矩阵乘以 w，再乘以 target user 的 embedding，再做一层 softmax，再乘以种子用户自己，这是一个典型的乘法 attention。它的作用是提取种子用户群体中和 target user 相关的部分。捕获种子用户的 local info。

第二部分是 global info，用 global attention，只和 user 相关，和 attention merge 的方法类似，也是一个 self-attention。作用是把种子用户乘以矩阵转换，再乘以种子用户自己，所做的就是捕捉用户群体自身内部的兴趣分布。得到的这两种 local & global embedding 之后，进行加权和，

这就是种子用户群体的全部信息。另一个问题来了，采用两种 attention union 来捕获信息，这意味着要计算很多次矩阵乘法，对线上开销很大。两个 embedding 需要多少次计算？这里有个表达公式，这个 h 是 embedding 的维度，K 是种子用户用户的数量，总的计算次数 = h * h * K * 2。对于线上耗时，一次预测超过 1000ms，无法接受。

优化耗时，第一个方法是减少种子用户的数量，这样会影响种子用户的表达完整性；另一种是我们线上采取的方式，使用聚类。找到种子用户内部比较相似的，把它们聚在一起。这种方法：1. 减少 key 的数量，2. 保持种子用户的全部信息。聚类的方式比较简单，用的是 k-means。



简单看下这个模型，右侧是 target user embedding，经过全连接，左边是 series user embedding 矩阵，两边都经过 embedding 之后，首先对种子用户的 embedding 做聚类，得到 k 个聚类中心，把种子用户的向量根据 k 个聚类中心做聚和，在类似中心内部做类似于 average 的聚和，然后得到 k

DataFunTalk

个向量，在这 k 个向量之上，一边做 global embedding，另一边和 target user 做 local embedding。有了这两个 embedding 之后，通过加权和的方式，做 cosine，再去拟合 user 到 item 的 label。这里的 label 用的是点击。

细节：

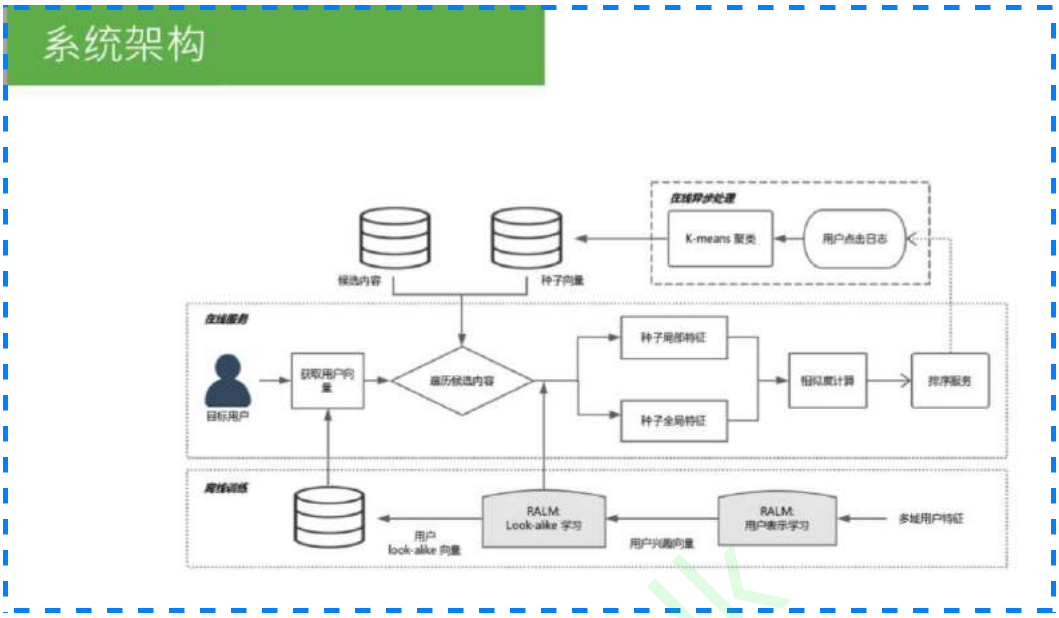
聚类的过程需要迭代，比较耗时，并非每个 batch 都去更新聚类中心，而是采取迭代更新的方式，比如把 1000 个 batch 一轮，训练完 1000 个 batch 之后，这 1000 个 batch 中，不更新聚类中心；到了第二轮，根据全连接参数的变化，再去更新种子用户的聚类中心，每通过一轮更新一次聚类中心，保证和核心参数是同步的。这样既保证了训练的效率，也保证了训练的准确性。聚类的优化，使线上的计算次数减小到了 k/K 中，之前 K 是万级别的数量，现在 k 是百级别的数量，耗时也下降了很多。

根据实验结果，确定不同聚类中心数 k 带来的影响，选择了合适的 k。实验中，k=20，线上 k 是 100 左右。模型训练的 label 优化方式，是一个多分类。对不同的种子用户人群选择最相似的用户。多分类的优化方式和 deep model 相似，采用 negative sampling 的方式。

系统架构

DataFunTalk

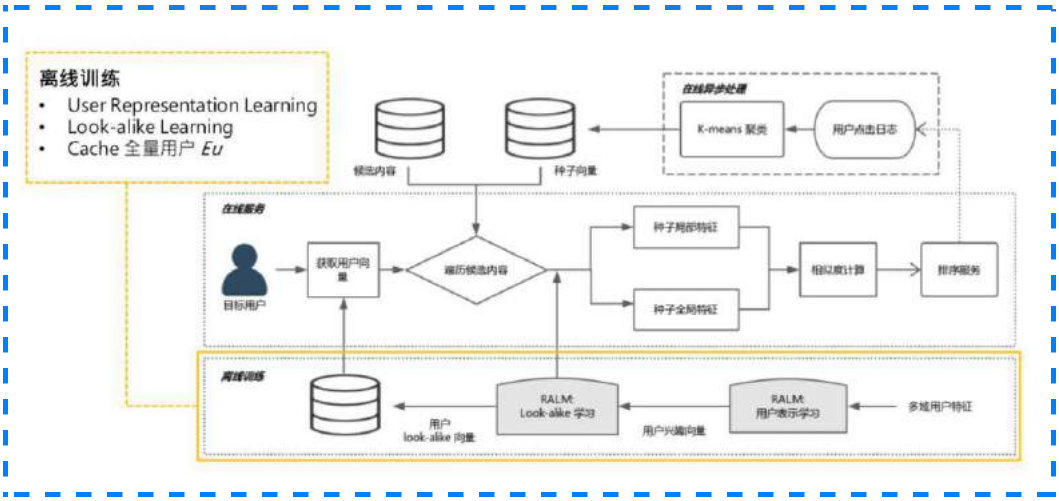
线上需要实现实时预测，系统实际部署到线上，需要整套系统架构。简单介绍下 RALM 的配套体系。



大体过程，分成三个模块，从最底下的离线训练，到在线异步处理，到在线服务，接下来分别讲一下。

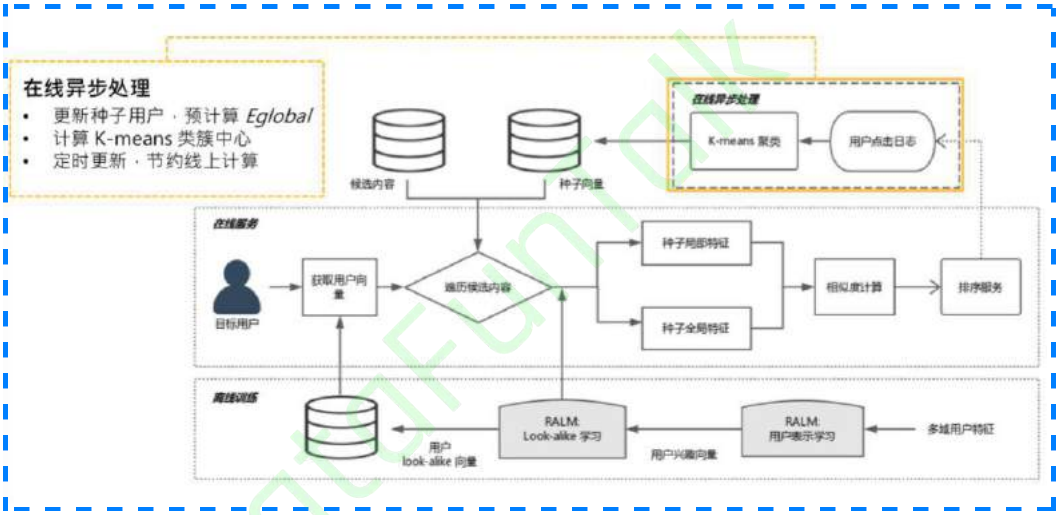
1. 离线训练

DataFunTalk



离线训练，就是两个阶段的训练，representation learning，look alike learning ,需要一提的是 ,进行完 look alike learning 之后 ,可以把 user 经过全连接层的 user 表达缓存起来。全量用户，有 10 多亿，可以 catch 到 KV 中。可以提供给在线服务做缓存，线上不用做实时全量传播。

2. 在线异步处理



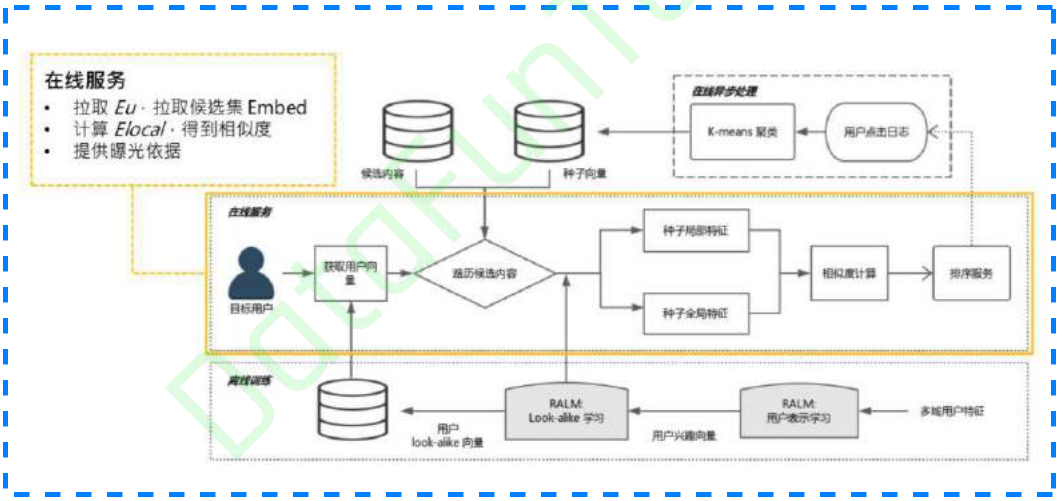
离线训练结束后，是在线异步处理，主要作用是某些可以离线且和线上请求无关的计算，可以先计算完，如更新种子用户。每个 item 候选集都会对应一个种子用户列表，更新种子用户列表，可以每一分钟更新一次，这和访问用户无关，只和候选集的 item 有关。可以实时拉取用户的点击日志，更新点击某个候选集的种子列表。

1. 可以把 global embedding 预计算（gl 只和种子用户有关，是 self-attenion，可在线做异步处理，如每隔一分钟算一次 ）。

2. 计算 k-means 聚类中心，也是只和种子用户有关，可以提前计算好，如推到推荐系统内存中。

3. 所有的东西都是定时更新，不需要线上实时计算。

3. 在线服务



线上把聚类中心、global embedding 和所有用户的 embedding 都已缓存好，只需要拉取 user embedding，和候选集的 global embedding 和聚类中心。线上只需要计算 local embedding，是 target user 到种子用户的 attention，这需要根据线上请求的 urn 来实时计算。再计算一次 cosine，就可以得到相似度，这个计算量很小。

实验结果

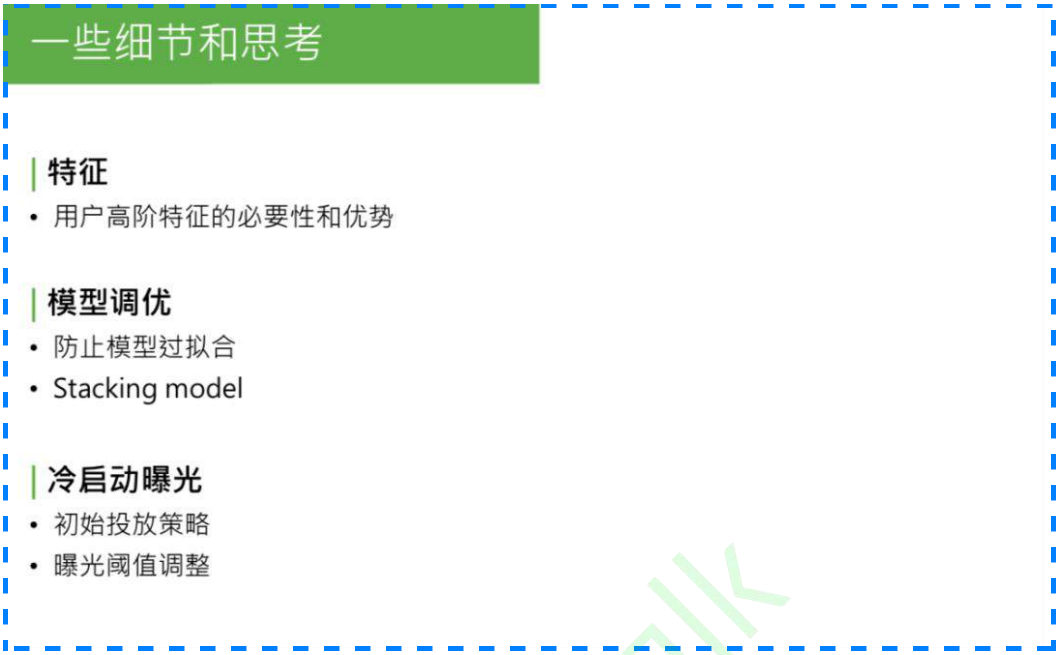
Table 2: Relative changes in metrics compared to control group in online A/B testing.

Metric	Change
Exposure	+9.112%
CTR	+1.09%
Category Diversity	+8.435%
Tag Diversity	+15.938%
GINI Coefficient	+5.36%

扩大曝光规模后，CTR趋向稳定，多样性指标提升

算出 look alike 相似度之后，相似度的分数，可直接给到排序服务，做曝光依据。这是当时写论文之前做的 ABtest，对比的是用户画像匹配推送的策略，上线之后，在扩大曝光规模的前提下，CTR 基本取向稳定，而且有微小提升，多样性也提升了很多，这都是相对提升。

一些细节和思考



特征：

为什么要用第一阶段的 user representation learning 得到用户的高阶画像？高阶画像的作用：包含了用户在某个领域的全部信息，信息量很大，结合 look alike learning 中的行为，需要去学习用户群体的特征。不用高阶特征，怎样学习用户群体？比较简单的方法是通过统计的方式：平均年龄分布和平均阅读倾向。这些都是基于离散的统计，信息损失很大。如果有了高阶的特征，高阶特征也是从低阶特征，如基础画像、年月分布，这些都是可以学到高阶特征中。如果能够直接输出所有领域的高阶特征，之后的利用、或者作为召回、作为 CTR 特征，都很方便。

模型调优：

① 防止模型过拟合。look alike 的结构很简单，这样做的原因：直接使用用户的高阶特征，使用了用户特征，如果模型不做处理，容易对高阶特征过拟合。采取了 2 种方式：

1. 尽量保证 look alike learning 结构简单；

2. 全连接层做 dropout。

② 采用 stacking model 的形式。看一看阅读、电商、新闻、音乐领域都做一次 user representation learning，这些特征用 stacking 的模式都放到 look alike model 中学习，这就是不同特征根据不同目标来训练的，更加减少了在同一个模型中过拟合的防线。

冷启动曝光：

Look alike model 中用了种子用户的表达，如果线上有新的 item，怎样做曝光？

1. 初始投放策略。使用基于 user item 的语义特征做线性模型的预测，当做冷启动 item 的初始投放。这个初始投放不需要积累很多种子用户，大概到百级别的种子用户就可以切到 look alike 逻辑了。

2. Look alike 出来的相似度分数，怎么做曝光的依据？如果直接用相似度分数，需要确定曝光阈值，如对于某个 item，高于多少分才曝光。我们使用的是线上试探曝光机制：最初给 1000 条流量，做曝光，这次曝光后，收集在用户侧的打分，取打分的分布统计，根据不同业务的要求，曝光 top 5% 或者 top10%， 来砍一个阈值分数，最后取曝光阈值。

本次分享就到这里，谢谢大家。

Q & A

Q：这个算法有没有在召回环节用，曝光该如何理解？

A：目前的策略有两种方式：

1. 直接采用召回的方式，定一个曝光阈值，直接确定是否曝光；

2. 把相似分数给到下游的 CTR model 作为参考。

Q：能否将两阶段学习合并成一个端到端学习？

A：End-to-End 方式存在两个问题：

1. 整个模型参数量很大，结构比较复杂，采用 End-to-End 方式不一定能学习到或者学习的很充分；

2. 刚刚讲到的 stacking 方式，我们最后需要的是尽可能全的表达用户的方式，所以右侧的 user representation learning 并不是从单一业务领域得出的结果，有可能是在多个领域得到的结果，比如在看一看训练一版 user representation learning，然后用社交或者电商上的行为，再做一版用户的表示，最后用 stacking 的方式把它们拼接起来，作为特征输入，这样达到的效果会更好。

Q：如果将第一阶段用户表征学习换成其他通用能学习表征用户向量的模型，效果会有什么影响？

A：我们单独用 user representation learning 和其它模型做过对比，比如 CTR 中的 user embedding，是针对当前业务比较精准化的表达，所在在泛化性上没有 user representation learning 效果好。

参考资料

Real-time Attention Based Look-alike Model for Recommender System
https://arxiv.org/abs/1906.05022

广告篇

阿里妈妈：电商预估模型的发展与挑战

嘉宾：周国睿　编辑：鞠 芳

本次分享内容提纲：

- 电商数据个性化预估的特性
- 阿里妈妈模型迭代路径
- Where to Go

一、电商数据个性化预估的特性

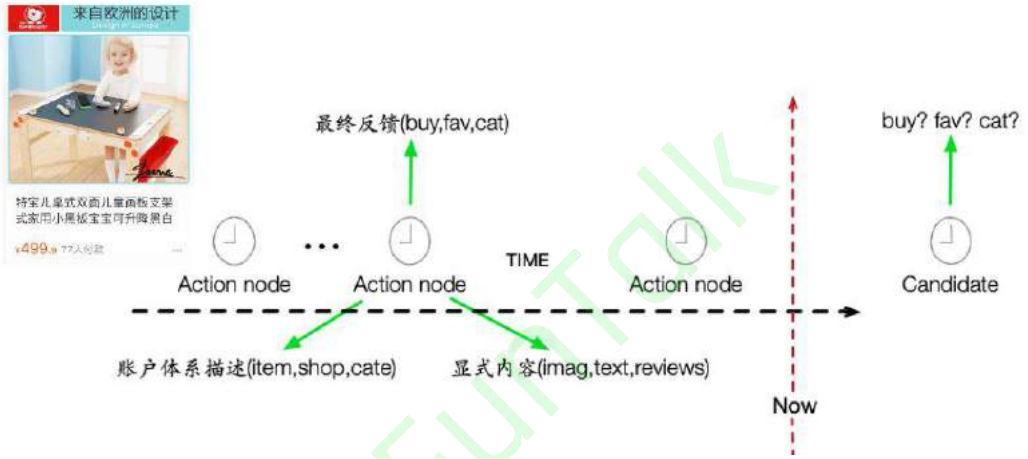
首先介绍一些基础的背景知识，阿里的展示广告按展示元素主要分为 Banner 广告和单品广告两种，所有广告内容由 id 、图、文三部分组成。因为展示广告面向的是用户的非搜索行为，用户不会向平台输入一个明确的意图，需要我们推测用户的意向。广告的个性化预估问题主要是，根据用户的历史数据，来预估其对于某个广告商品点击、收藏、购买的可能性。与 facebook 等社交媒体网站通过上传照片、点赞等行为来推荐多种元素广告相比，电商场景中有一个显著特点，预估的 candidate 节点内容与用户的历史行为节点内容是同质化的，都是商品。

可以通过三个主要维度来描述：

显式内容：imag，text，reviews 等内容信息

账户体系描述：item，shop，cate 等 id 信息

最终反馈：buy，fav，cat 等行为信息



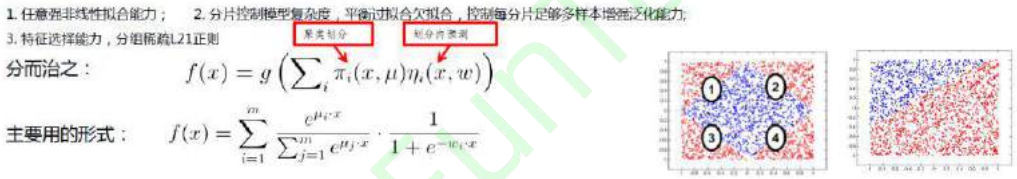
二、阿里妈妈模型迭代路径

随着历史的推演，从 2012 年起，阿里妈妈的预估模型也在不断的迭代和创新，演进路线如图所示：

MLR

大规模特征和逻辑回归模型的组合是点击率预估模型的传统经典方法。通过大规模的 id 和特征能够获得更高分辨率的样本描述。这种方法下，模型结构相对固定，优化方法上讲也更注重系统效率。从效果上看工作重心主要在大量的特征工程，通过人工经验对特征进行组合、运算，解决 LR 无法表达非线性关系的问题。但是这样的迭代模式整体受到人工能力制约，不够智能。

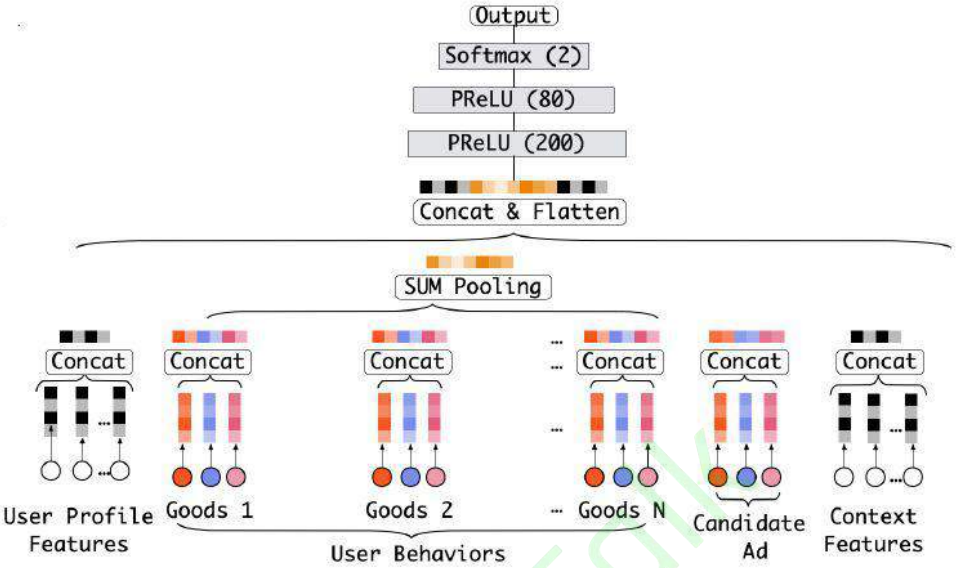
MLR（ 分片线性学习 ）算法是解决此类问题的一个非线性算法，其主要思想是将在一个平面线性不可分的数据，分为多片，投影到不同的高维平面，实现线性可分。同类的解决这类问题的常用方法还有 GBDT+LR 的组合。



第一代深度 CTR 模型

随着硬件的发展，算力、数据量的进一步提升，深度学习在诸多领域如图像、NLP 、语音都取得了突破性的进展。阿里妈妈在经历了从 2015 年的一系列探索后，在 2016 年尝试了第一代深度学习算法，网络结构相对简单，主要就是 embedding 层+MLP，效果却相对 MLR 得到了显著提升。分析原因主要是传统算法因为算了和数据问题，模型的 capacity 有限，非常依赖人的先验知识在有限的 capacity 下设计高效的模型。而算力和数据利用量提升

后，模型的 capacity 通过深度学习出现了量级的增加，拟合能力得到了显著的提升。



阿里妈妈认为和传统的研发模型相比，深度学习具有通用的优化器，使得模型设计和优化问题解耦，模型设计组件化，从而能够更快的设计验证更加复杂的模型。

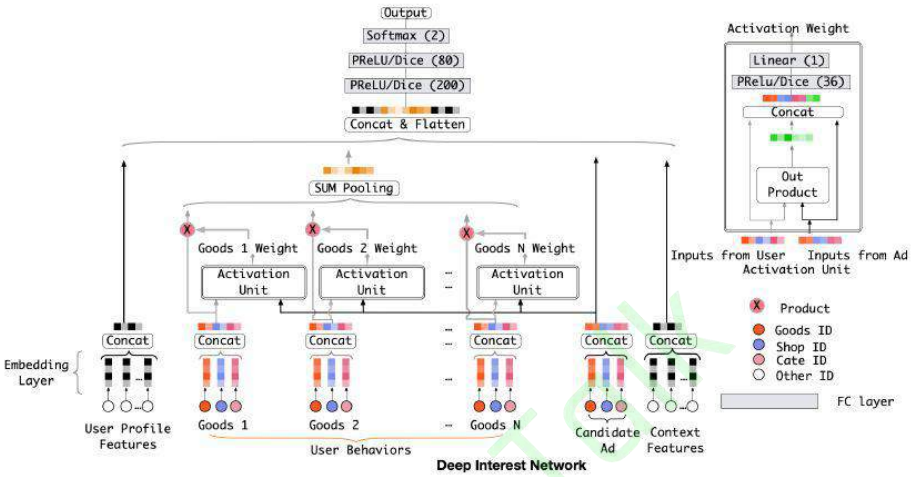
但是，复杂并不是终点，算力和数据都存在上限，采用深度学习要走的更远，模型结构和数据分布的匹配将是研究的方向，例如 CNN 对于图像数据，RNNs 对于语音和文本的数据都有很好的适应性。那么适合电商数据的模型结构是什么？这一问题，成为深度兴趣网络诞生的源头。

深度兴趣网络（DIN）

电商用户的兴趣存在多样性的特性，即一个消费者会同时有多种多样的兴趣，在喜欢一件大衣的同时，也可能喜欢一个杯子，又同时是电子产品的爱好者。在第一代深度学习网络中，我们将用户的历史行为 encode 成为一个固定的向量，实际上，直接通过一个固定向量去表示这种多种多样的兴趣爱好是十分困难的。我们可以通过扩大这个向量的维度来提升它的表达能力，但是这样会带来计算复杂度增加和过拟合等问题。

这时我们注意到，用户一系列的历史行为和一个目标商品相关的通常只有其中的几个。采取类似 attention 的机制（Activation Unit），选取和某一商品相关的用户历史行为，就可以动态的使用一个定长的向量来表征用户多样的兴趣，这是 DIN 模型的核心内容。而前文提到的用户行为和商品的同质化，使得这种模型的效果更好。

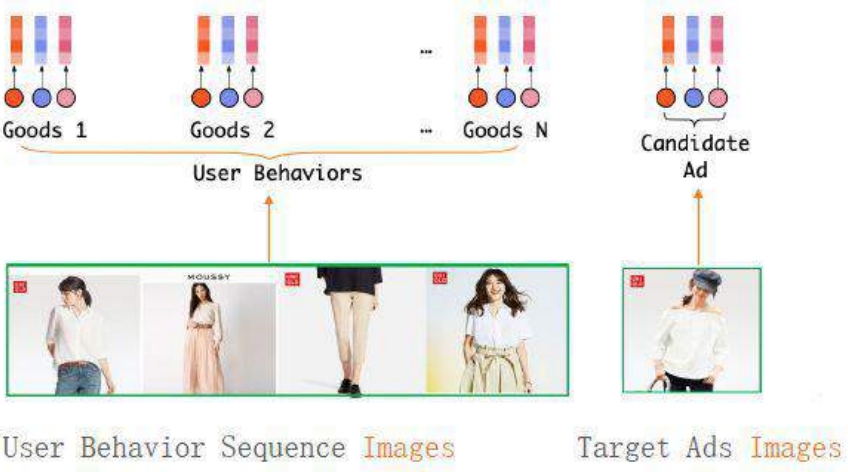
具体的网络结构如下图，其中 Activation Unit 设计的比较简单，主要是由于工业界时延等要求的限制：



DIN 模型使得阿里妈妈广告的 CTR（ 点击率 ）上升了 10%，CVR（ 转化率 ）提升了 3.3%，GPM（ 千次展现用户成交金额 ）提高了 12.6%。

CrossMedia 网络

在网络结构调整告一段落后，阿里妈妈致力于将图文数据作为特征增加到模型中去。主要原因是，图文是用户直接接收的信号。并且图问信息和 ID 类特征不同，ID 类特征的所有信息都源于样本里和 label 共现信息的反馈，而图文本身具有物理含义，所以其具有更好的泛化性。



在这一过程中的挑战主要在于图片的数据过大。一个用户如果有一千多个历史行为，和商品组合后有 100 亿样本，涉及图片数据会达到上百 T，对于存储和 I/O 都会造成很大的压力。

最终的解决方案是，将图片数据存储在远端 server，并在 server 上采用神经网络提取图片为一个低维的向量，在样本中将图片存储为一个 id，计算时通信获取压缩后的图片向量，实现效果和效率之间的一个 tradeoff 。

Deep Interest Evolution

在 DIN 研究的时候，阿里妈妈发现 RNNs 的时序网络，不能在电商预估场景中得到良好的表现。原因是由于用户的历史行为可以被看做是很多个兴趣的很多采样点混合在一起的综合序列，其中存在着很多随机跳转。例如，形式化某个用户的一段历史行为序列：

$$S = [A1,A2,B1,B2,A3,A4,B3,B4],$$

其中，$A,B \in I$ 代表兴趣空间内的两种不同兴趣。

$$[A1,A2,A3,A4] \in A,$$

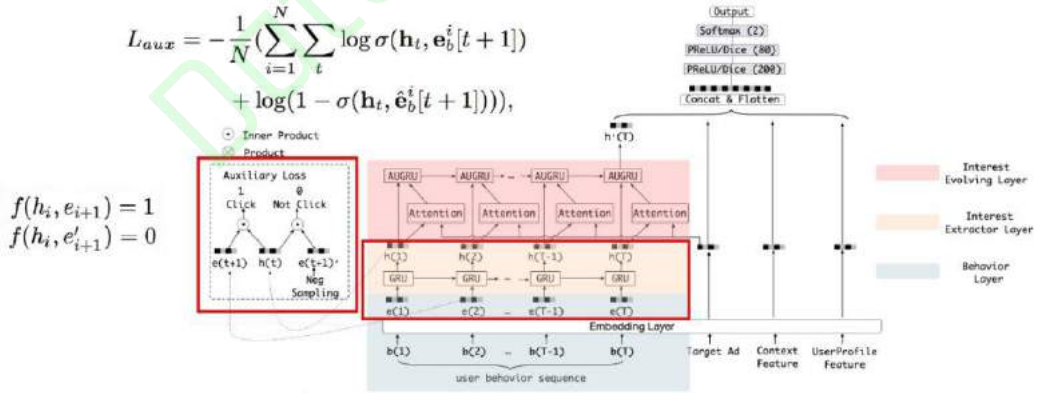
$$[B1,B2,B3,B4] \in B$$

同时现有的模型直接将用户的历史行为的 embedding 当做兴趣缺乏抽象、概括能力。其实，行为只是抽象兴趣的具体体现，用户的每一次行为的结果只是去兴趣下的一种选择，每一次点击的商品可能只是部分满足了用户的兴趣，并不是这个商品的所有属性都代表了用户的兴趣。而且多种兴趣存在不同的演化路径。阿里妈妈通过兴趣提取层和兴趣演化层对模型进行了优化。

兴趣提取层

$$L_{aux} = -\frac{1}{N}(\sum_{i=1}^{N}\sum_{t}\log\sigma(\mathbf{h}_t, \mathbf{e}_b^i[t+1]) + \log(1-\sigma(\mathbf{h}_t, \hat{\mathbf{e}}_b^i[t+1]))),$$

$$f(h_i, e_{i+1}) = 1$$
$$f(h_i, e'_{i+1}) = 0$$

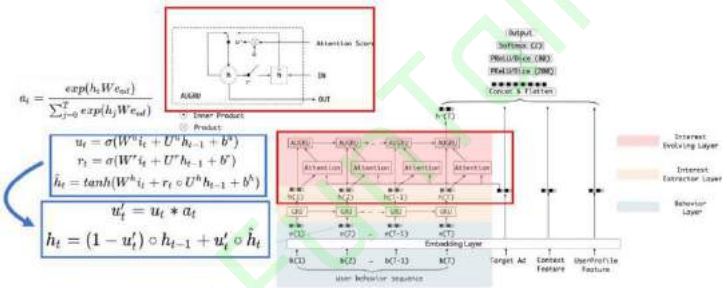


该层的构想，目的是挖掘出行为背后用户抽象兴趣的表达。

通过 Embedding 的学习得到用户行为更多语义信息后，使用 GRU 隐层状态表达抽象兴趣；使用有监督的点击序列 pattern 实现辅助损失（auxiliary

loss），而非 ctr 最后点击信号——通过 t 时刻用户兴趣的表达 h(t) 和 t+1 时刻用户的点击动作 e(t+1) 和非点击负样本 e(t+1)' 的动作实现一个有监督的 loss，使得 h(t) 的隐层不仅能够表达对最终 output 的影响，也能预测用户下一个行为动作，能更好的表达用户兴趣，同时有效解决长序列梯度传播问题。

兴趣演化层



兴趣不光是行为的抽象，具体到某个兴趣，还会随着时间逐渐演化。兴趣演化对行为的影响是用户可能在一段时间内对各种书籍感兴趣，而在另一段时间内需要衣服。虽然兴趣可能相互影响，但每个兴趣都有自己的发展过程，例如书籍和衣服的发展过程几乎是各自独立的。我们只关心与目标产品相关的兴趣演进过程即可。

在兴趣提取层上，阿里妈妈增加了 Attention Unit GRU，与传统 GRU 不同的地方在于在 update 门和 reset 门上乘上了一个 attention score 。首先，通过用户的历史行为计算出一个 attention 值，由此体现和未来预估

目标的一个相关性权重，通过相乘，使得和目标不相关的行为，不会对 GRU 结果产生影响，相关行为才会使 GRU 结果发生迁移。

由此，从繁杂的综合序列中抽取出相关兴趣，与广告越相关的行为更新隐状态向量越多，精确捕捉当前广告相关兴趣。

模型效果

公开数据集效果

Table 2: Results (AUC) on public datasets

Model	Electronics (AUC)	Books (AUC)
BaseModel (Zhou et al. 2018c)	0.7435 ± 0.00128	0.7686 ± 0.00253
Wide&Deep (Cheng et al. 2016)	0.7456 ± 0.00127	0.7735 ± 0.00051
PNN (Qu et al. 2016)	0.7543 ± 0.00101	0.7799 ± 0.00181
DIN (Zhou et al. 2018c)	0.7603 ± 0.00028	0.7880 ± 0.00216
Two layer GRU Attention	0.7605 ± 0.00059	0.7890 ± 0.00268
DIEN	**0.7792 ± 0.00243**	**0.8453 ± 0.00476**

线下数据验证效果

Table 3: Results (AUC) on industrial dataset

Model	AUC
BaseModel (Zhou et al. 2018c)	0.6350
Wide&Deep (Cheng et al. 2016)	0.6362
PNN (Qu et al. 2016)	0.6353
DIN (Zhou et al. 2018c)	0.6428
Two layer GRU Attention	0.6457
BaseModel + GRU + AUGRU	0.6493
DIEN	**0.6541**

线上 A/B test 测试效果

Table 5: Results from Online A/B testing

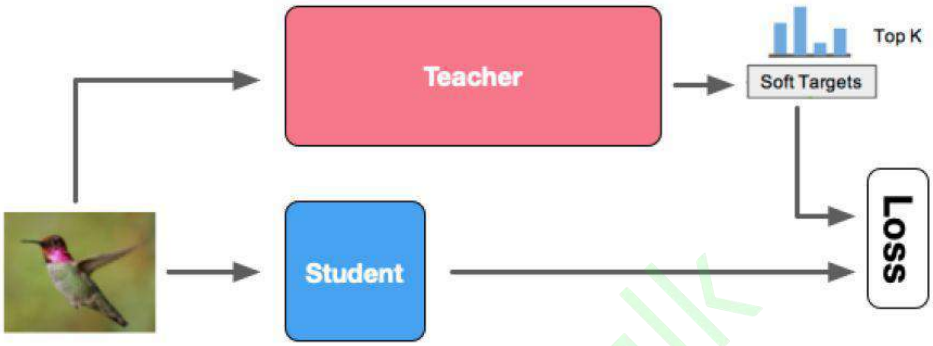
Model	CTR Gain	PPC Gain	eCPM Gain
BaseModel	0%	0%	0%
DIN (Zhou et al. 2018c)	+ 8.9%	- 2.0%	+ 6.7%
DIEN	+ 20.7%	- 3.0%	+ 17.1%

Rocket Launching

随着模型复杂度的上升，在线预测的时延和 tps 就越来越难以满足。对于离线的模型，为了追求模型表现，可以足够复杂；但对于在线预测模型，需要控制模型复杂度，实现效果和性能的平衡。

为此，阿里妈妈提出了一个类似的模型蒸馏的 teacher-student 方案，压缩模型的计算复杂度。在离线的时候训练一个复杂的 teacher 模型，同时训练

一个很简单的 student 模型，通过 teacher 对目标的 soft 预测结果来监督 student，同时也用真实的 label 来监督 student 。



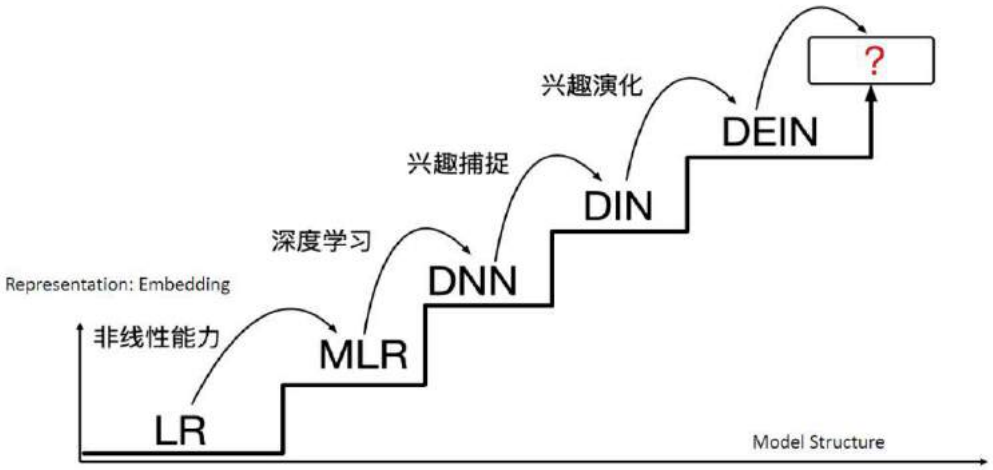
与一般模型蒸馏不同的三个点是：

协同训练，teacher 和 student 一起进行训练。通过细到 batch 的数据，使 student 学习的不再是一个最优的 teacher 解，而是 teacher 模型的学习轨迹；

参数共享，模型一般分为表达层和决策层，student 通过参数共享能够获取 teacher 表达层训练的成果；

梯度 block，通过特定损失项，防止 teacher 向 student 学习。

三、Where to Go



回顾阿里妈妈模型的演化路径，从线性到非线性，再到深度学习的不断演化，模型的结构一直在变化，但是 Representation 的技术上一直没有明显的进展，展望未来的发展方向主要有两点：

1. Representation 的研究，目前一些预训练和类似 NLP 的方案在我们的场景没有良好的效果。推测可能是因为电商数据商品之间并没有如 NLP 一样词与词之间从属，双关，同义等关系，然而商品之间的结构关系已经被天然的账户体系，店铺、类目、品牌等特征很好的表达。但是我们依然会继续推进，后面会尝试 Disentangled Representation，学习电商领域的抽象表达，提取出有效的 Concept，更好的解释我们的 representation 每一维的语义信息是什么；

2. 黑盒的模型让平台离用户/商家的距离越来越远，平台自身也不能完整的理解模型的机理。希望能够构建更白盒的模型，知晓影响用户决策的 Concept，并依此联系用户和商品。产品上我们可以透传出推荐的理由和用户做更多的

交互，商业营销上可以结合商家自己的卖点是什么，可以让商家更精准的实现营销诉求。

深度学习在 360 搜索广告 NLP 任务中的应用

嘉宾：高凯明　编辑：Hoh Xil

360 搜索广告成立于 2012 年，到今年是第 8 个年头了。

今天分享的内容分为两部分，第一部分是搜索广告和广告召回。我会介绍搜索广告的业务逻辑，以及召回模块的逻辑。第二部分是语义相关和深度学习，这部分会介绍语义相关的计算方法以及使用的深度学习模型。

搜索广告 & 广告召回

360 的搜索广告系统从逻辑上可以划分为三个模块：广告召回、广告排序和广告展示。

广告召回：这个模块决定召回哪些广告，涉及到广告索引和相关性计算；

广告排序：这个模块决定展示哪些广告，涉及到点击率预估和竞价机制；

广告展示：这个模块决定如何展示广告，涉及到广告创意的选择。

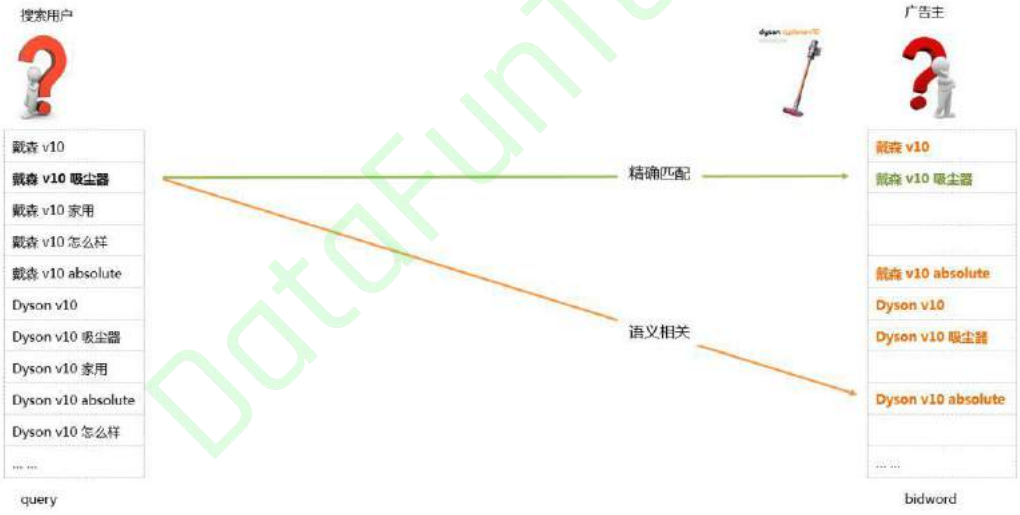
下面通过一个 Case，来说明 360 搜索广告系统的业务逻辑：

假设，我们两个电商行业的广告主，都购买了某品牌吸尘器的广告。他们向广告系统提交了关键词，和出价。

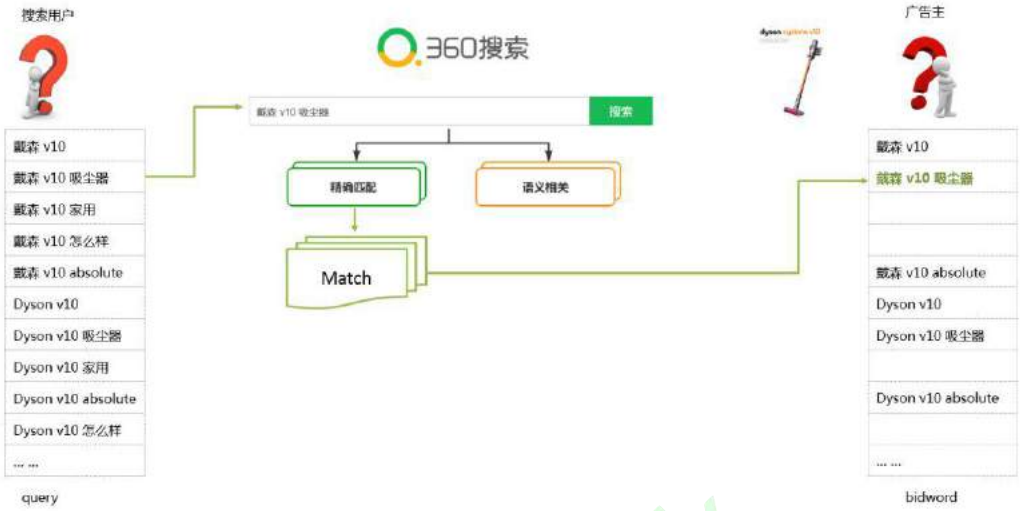
召回模块会计算 query 与广告关键词的相关性，根据相关性来决定是否召回这两个广告，结果是两个广告都相关，所以两个广告都召回。

广告排序模块，会计算两个广告的点击率和质量分，并根据竞价机制，来决定他们是否可以展示，以及展示位置。

最后，广告展示模块会根据广告主提供的物料以及展示位置，来选择广告创意。



接下来继续应用这个 Case 来说明广告召回模块的召回逻辑：左边是用户搜索的 query，右边是广告主购买的关键词，这里有俩种召回方式，第一种是精确匹配召回，第二种是语义相关召回。



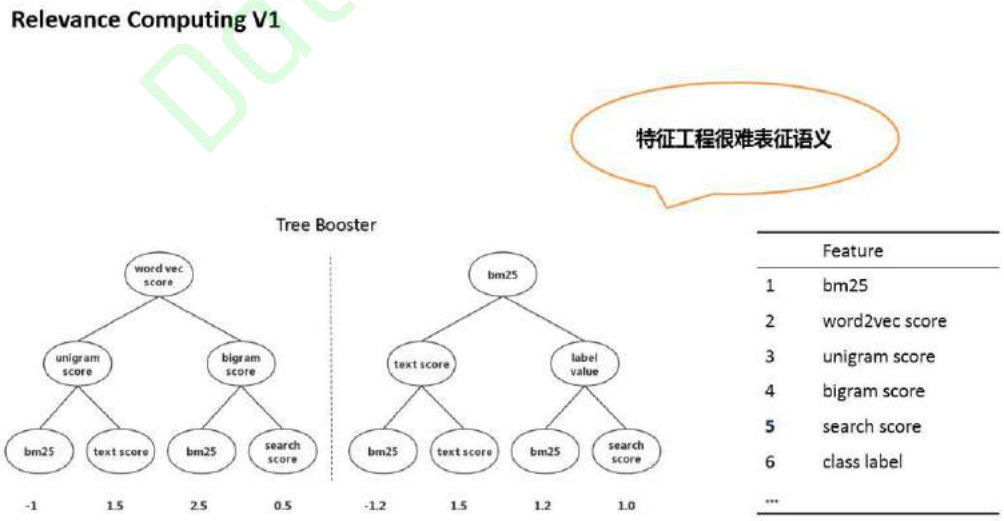
精确匹配召回对应的是一个 Match 的逻辑，可以理解成字符串的匹配，当然还有一些规则的匹配，比如这里会做一个切词之后的乱序的重排序，重排序之后如果能匹配上，我们也算是精确召回。

语义相关召回对应的是一个 lookup 的逻辑，这里查找的是一个 query 到 bidwords 的映射表， 这个映射表是由一个离线挖掘的流程提供的。而离线挖掘的流程是一个漏斗的逻辑。漏斗的上方是多种数据挖掘方法，比如随机游走，文本检索，等。这些方法提供 query 到关键词的候选集。漏斗的下方是一个相关性模块，这个模块对候选集进行相关性过滤，把过滤后的数据提供给线上的映射表。

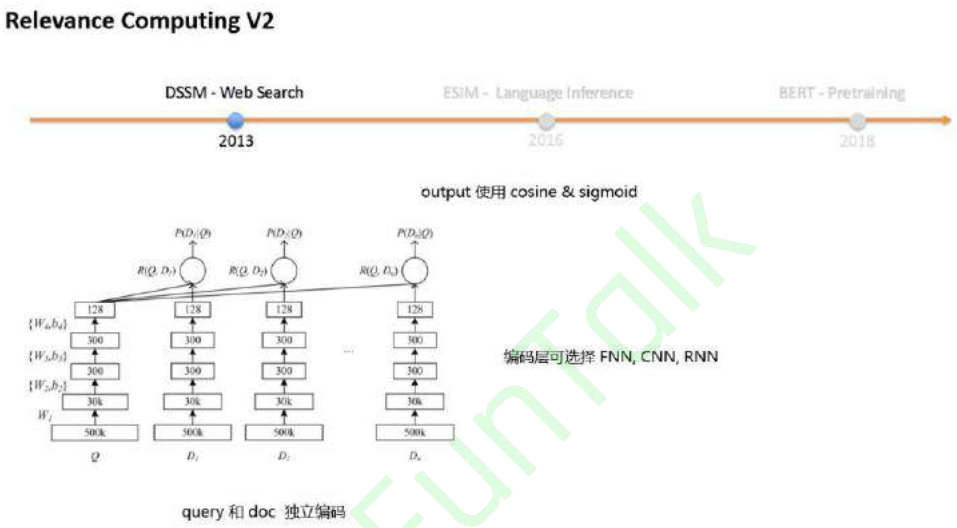
所以处在漏斗下方的相关性模块直接决定了线上数据的质量。

语义相关&深度学习

我们进入第二部分，语义相关和深度学习。

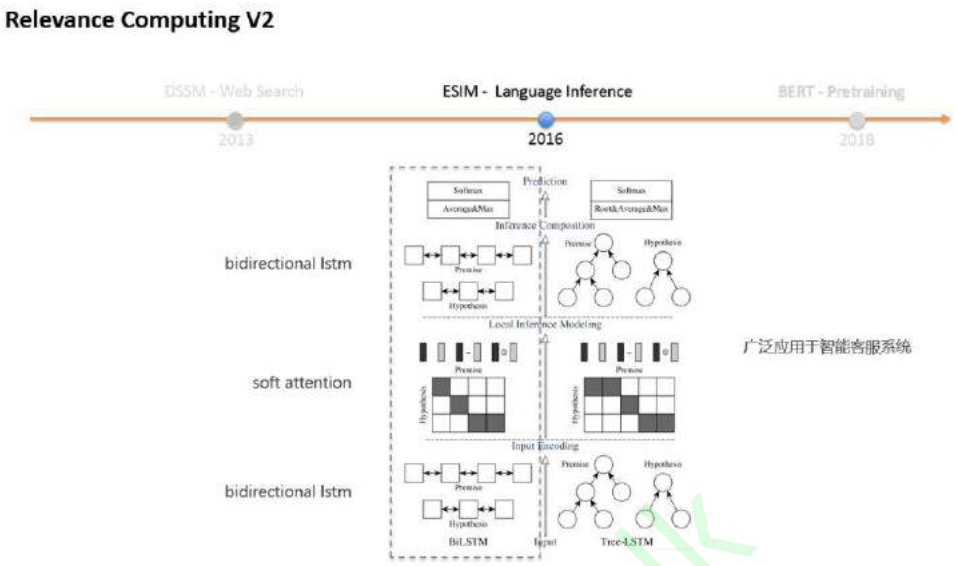


早些时候，计算语义相关性的方法，主要是特征工程 + GBDT。特征包括：文本相似度，embedding 相似度，bm25，以及搜索引擎提供的一些特征，等等。特征工程的问题在很难表征语义，所以准确率得不到保证。

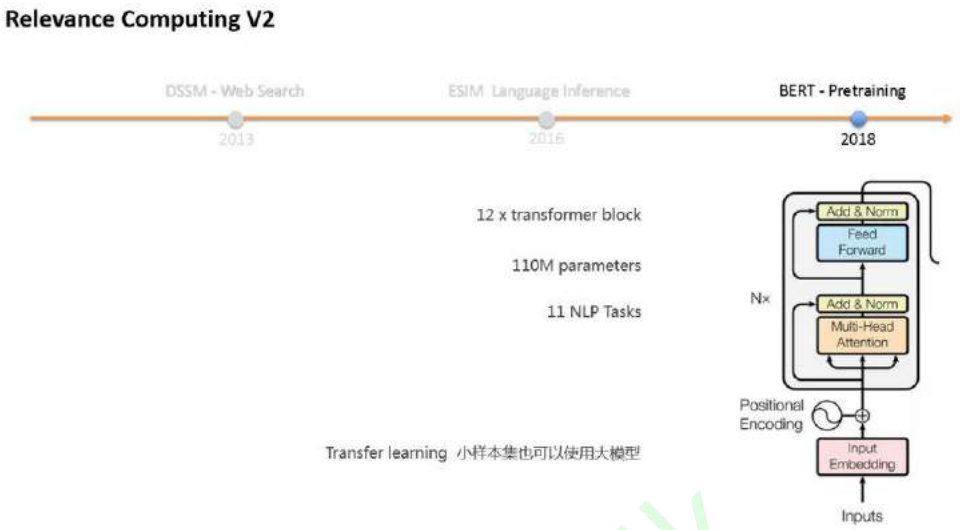


随着 DeepLearning 技术的发展，nlp 领域的多个任务，相继提出了深层语义模型。

2013 年，web search 任务提出了 DSSM 模型，DSSM 对 query 和 doc 独立进行编码，编码层可以选择 FNN，CNN，或者 RNN，输出层用 cosine 和 sigmoid 来计算相关性。DSSM 的特点是 query 和 doc 独立编码，doc 的编码可以离线计算，线上只做 cosine 和 sigmoid 计算。



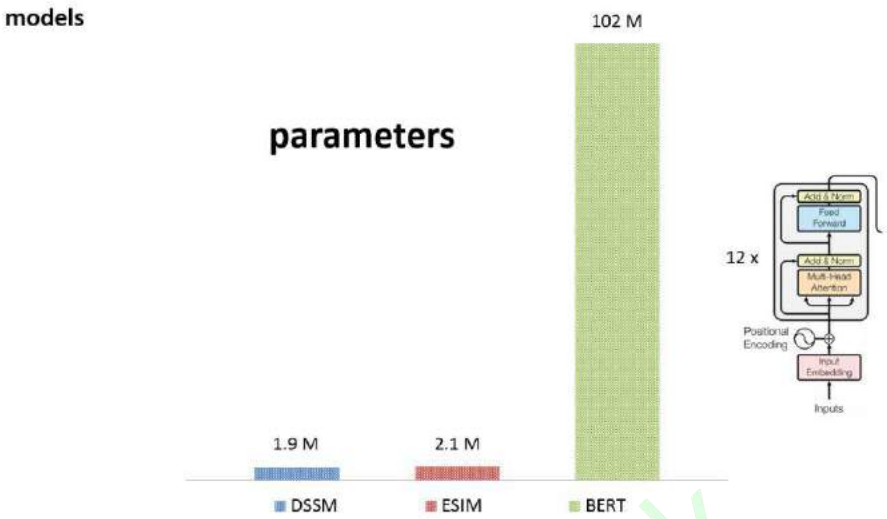
2016 年，language inference 提出了 ESIM 模型，ESIM 采用了两层 bidirectional LSTM，中间引入了 soft attention。这个模型广泛应用于智能客服系统。如果大家对智能客服了解的话，大概会知道客服系统一般会构建一个问答库，ESIM 用于计算问题和问题的相关性，然后把语义相同的问题归结到同一个答案上。这个就跟我们的广告词召回非常像了，我们的任务是让语义相同的 query 可以召回相同的广告关键词。



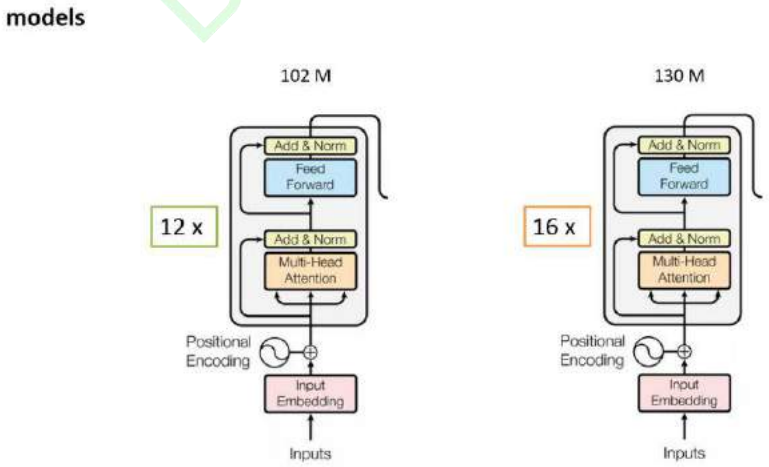
2018 年，也就去年，我们说，Bert 模型横空出世。Bert 采用了 pre-training 和 fine-tuning 的方式，真正意义上 实现了 NLP 领域的迁移学习。Bert 刷新了 11 项 NLP 任务的记录，其中就有两项语义相关的任务。

我想大家对 Bert 的评价主要是两种，第一种是，Bert 让我们看到了深层的 Transformer encoder 具有强大的语义表征能力。第二种更实际一点，是 Google 发布的 pretrain 的 model，中文方面就是那个 base 版的 Chinese model。他的意义在于，对于一些小样本的问题，我们用有限的样本，去 fine-tune 这个 model，就可以获得不错的效果。小样本使用大模型，这在之前是做不到的。

1. Models



这是我们在实际工作中使用用的三个模型的参数对照。其中 ,DSSM 和 ESIM 参数数量都在 2M 这个量级。而 BERT 参数总量超过了 100M。之所有这么多的参数，是因为 bert 将 transformer 层累加了 12 层。而且每个 transformer 层又是一个 12 个 head 的 multi head Attention。

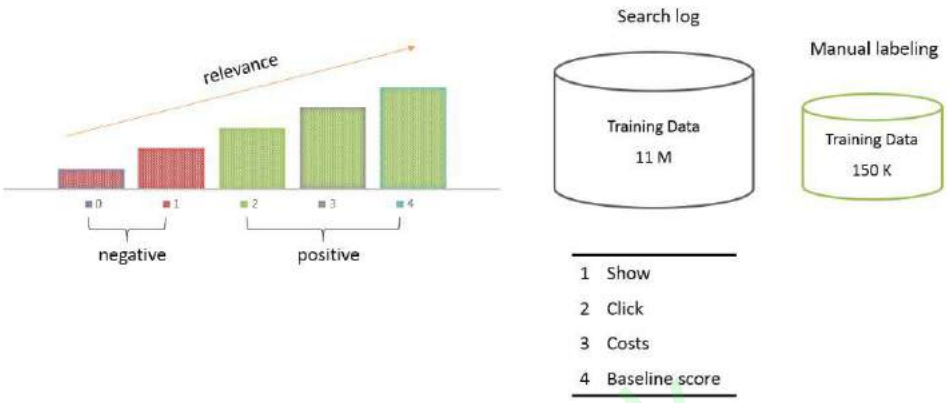


我们知道，Google 发布的中文的 model 是 12 层的 base 版 ，没有发布 24 层的 large 版。出于一个贪婪的想法，我们在 12 层上又累加了 4 层 transformer block，这时候，参数量从 102M 增加到 130M。如果大家有足够的训练数据，可以往上增加层数，如果数据量不够的话，使用原始的 12 层是足够的。

2. Data

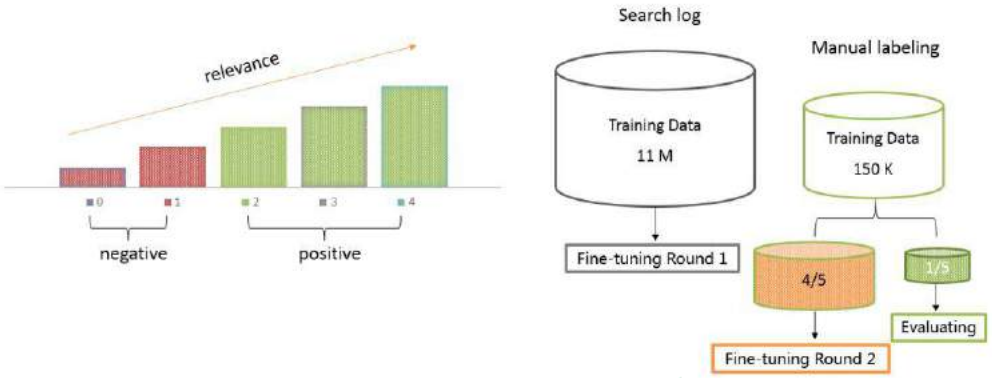
关于数据，我们把相关性的大小，定义为 0 到 4 的五个分值，相关性依次升高 。这和 bert 刷榜的 11 个任务中的 Semantic Text Similarity Benchmark 这个任务是一样的。同时，我们把 0 分和 1 分定义为负例，把 2 分，3 分，和 4 分定义为正例。

data



我们维护了两个样本集。大样本集有近 11M 样本，主要是通过广告点击日志筛选出来的。通常我们的筛选方式是结合展示，点击，消费数据，然后再加上一个 baseline 的预测值，综合几个维度，按照经验阈值来筛选。这个样本集的特点是样本量大，混淆度也大。我们还有一个小样本集有近 15 万样本，通过长期的人工标注和运营同学反馈得来的。

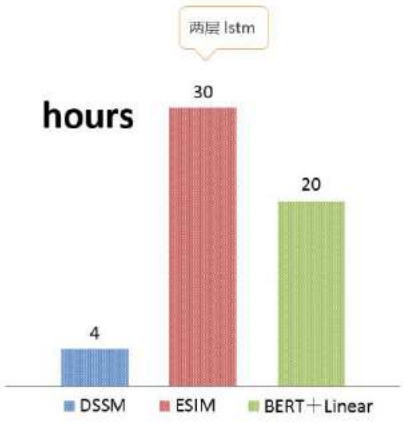
data



两个数据集使用的方式：首先用大数据集，来做第一轮的 training 或 fine-tuning；然后，从小数据集中选取 4/5 来训练第二轮，用剩余的 1/5 来做评测。

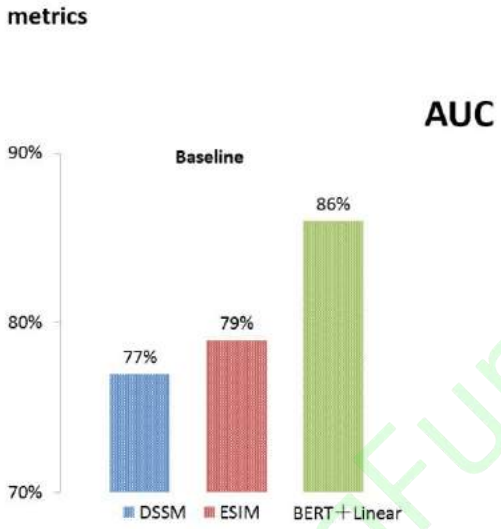
training

Tesla P40

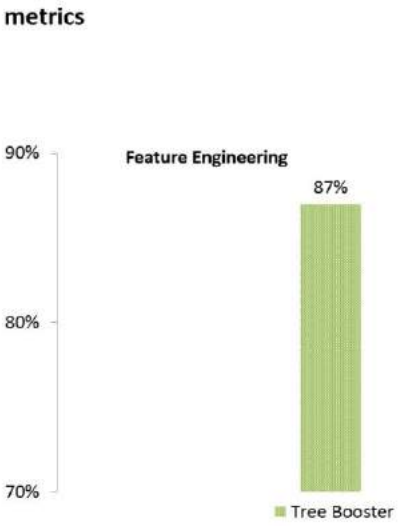


这是我们在一个 Tesla P40 上，训练三个模型的耗时。可以看到， 由于采用了两层 LSTM，ESIM 的耗时是最长的。

3. 性能评测



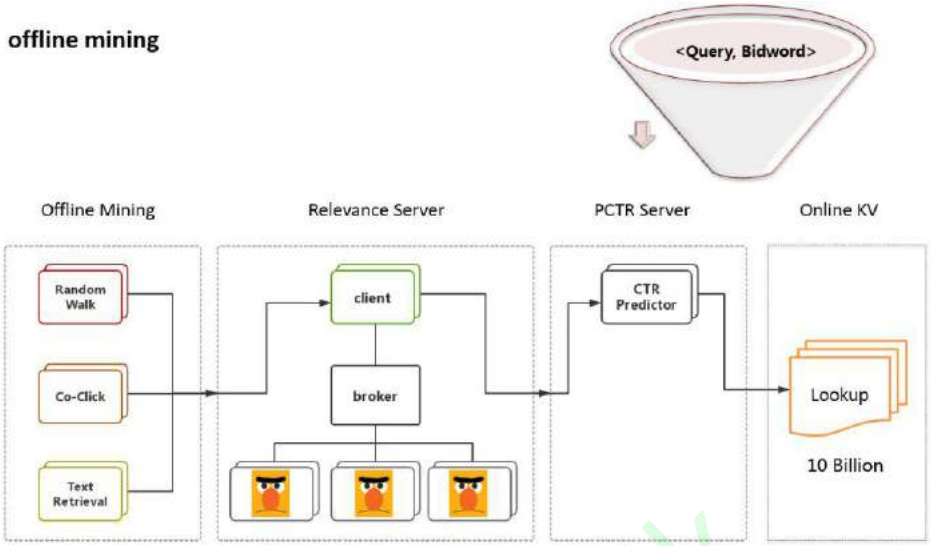
在衡量模型的指标上，我们选择了 AUC。DSSM 和 ESIM 的 auc 是比较接近的，ESIM 表现更好一些。通过分析 badcases，我们看到 DSSM 和 ESIM 还是比较依赖字面相关的，对语义的捕捉不是很好。而 Bert 的 Auc 达到了 86%。



	Feature	importance
1	bert	1
2	dssm ex	0.83
3	dssm	0.36
4	bm25	0.29
5	word2vec	0.26
...		

以前面的模型为基础，我们又尝试了一些特征工程。我们结合了 bert 和 dssm 和其他十几维特征，训练 tree booster，AUC 可以达到 87 %。不过考虑到特征工程的复杂性，我们没有采用这个方案。最终，只采用单独 BERT 去做相关性服务。

4. 离线挖掘



这是我们的一个离线挖掘流程：

整个流程是个漏斗的逻辑。首先是用一些离线挖掘的方法，从日志中找到 <query, bidword> pairs，或者用文本检索的方法，获取到候选集。然后通过相关性的预测，再通过 CTR 预测，最后把过滤后的结果发布到线上的 KV 系统。

所以，整个 360 的广告召回模块就是这样，按照两种召回方式，其中语义相关召回方式会提供一个非常大的空间，这个召回方式采用了一个离线的挖掘流程。然后采用相关性计算、CTR 预估过滤流程来提供映射表。这是今天分享的全部内容，谢谢大家。

Hulu：视频广告系统中的算法实践

嘉宾：韦春阳　编辑：Hoh Xil

导读：Hulu 是一家美国领先的互联网专业视频服务平台，商业广告是 Hulu 的重要变现手段之一。视频网站中的广告以担保合约式品牌广告为主，本次演讲分享了机器学习、人工智能技术如何在 Hulu 的广告系统中实践落地，如何帮助广告业务更加高效的运转，介绍了包括精准广告定向、广告流量匹配、广告库存预估等项目中机器学习算法的应用实践。

概述

首先介绍一下 Hulu 以及 Hulu 的广告产品形态。

Hulu 是一家美国的提供专业视频点播与直播的服务平台，成立于 2006 年，目前由迪士尼控股，拥有近 3000 万付费订阅用户，是全美用户数量增长最快的流媒体平台。

① 产品形态：点播，直播，回看等；

② 广告形态：以视频流中的 15 秒、30 秒的视频广告短片为主；

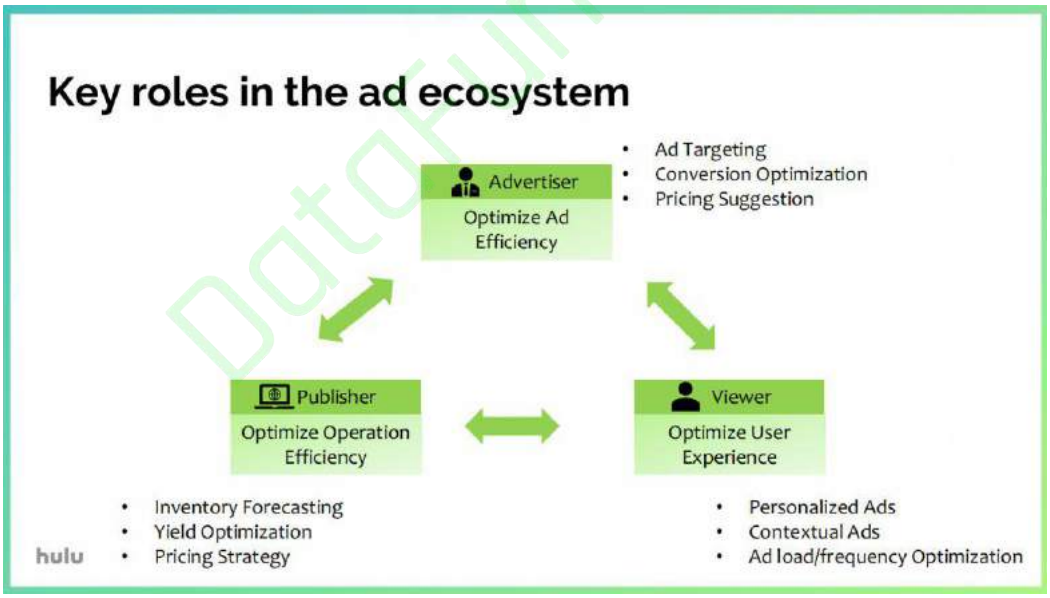
③ 广告特点：

以品牌类广告为主，品牌类广告看重长期效果，品牌的曝光度，用户认知等等；

采用 CPM（Cost Per Mille，千次展示 ）方式计费，即无论用户是否点击，按照展示次数计价；

以担保式广告订单为主，广告主希望其品牌曝光有一定量的保障，例如，某品牌希望在指定地区给指定用户群投放指定的次数；

质量要求非常高，包括清晰度、创意、制作的精良程度等等。



上面简单介绍了 Hulu 和 Hulu 的广告产品形态，接下来介绍下广告系统中我们主要面对的核心算法问题有哪些，以及算法发挥的主要作用。

广告生态系统有三个核心要素：广告主，用户，媒体方（Hulu）。每个要素都有自己的优化目标。

广告主：

广告主的核心优化目标是广告的投放效率、有效性，以及 ROAS（Return-On-Ad-Spend，广告支出回报率），算法的主要应用为：

① 广告定向：使用机器学习算法进行受众定向、基于上下文的定向、lookalike 定向等；

② 订单及价格优化：给广告主一些订单定向条件及价格的建议，使其在 Hulu 上更好的进行订单的购买和投放；

③ 转化率优化：越来越多的品牌类的广告主也开始在意短期或者长期的转化率，使用机器学习方法提高转化率，把正确的广告投给正确的用户；

用户：

对于用户的核心优化目标是优化用户体验，尽可能减少广告对用户的干扰，给用户更有连贯性的观看体验，算法的主要应用为：

① 个性化广告：给用户推荐更感兴趣，更相关的广告

② 情景式广告：使用 AI 技术进行图像音频的识别、广告创意的生成，为用户带来浸入式广告体验

③ 个性化频次控制：为用户提供个性化频次控制，避免重复及过量广告为用户带来的反感

媒体方（Hulu）：

最终的优化目标：广告收入最大化，吸引更多的广告商和用户；与此同时我们也会进行生产环境的工作效率优化，算法的主要应用为：

① 库存预估：这是在担保式广告里面是一个必不可少的环节，在售卖广告以及做广告预算时发挥重要作用

② 广告流量匹配：对广告订单与用户流量进行匹配，保证担保式广告能够按时按量且均匀的进行投放

③ 定价策略优化：根据流量的热门程度及售卖压力，进行动态的定价，让整个系统得到一个收入的最大化。

下图总结了在 Hulu 的视频广告系统中的核心算法问题，也是我们研究员每天在做的事情：

① 广告定向，涉及到：

❖ 预估模型：例如用户标签的补全，构建 lookalike 模型等等

❖ 主题模型：使用无监督方式挖掘用户属性

❖ 图像识别技术：识别视频内容中物体、场景、氛围等等，投放上下文相关视频广告

② 库存预估，涉及到：

时序预测算法：库存预估一般被建模成时序预测问题

③ 流量匹配，涉及到：

❖ 凸优化：使用凸优化进行离线流量匹配

- 自动化控制理论（PID Controller）：使用自动化控制理论进行线上动态调整

④ 转化率优化，涉及到：

- CVR 预估模型
- Casual Inference（ 因果推断 ）：使用因果推断，得到一个更加公正、有效的指标来评估转化是否有效。
- Bandit/增强学习：使用 Bandit 以及增强学习的方式来解决广告冷启动的问题

⑤ 程序化交易广告，涉及到：

- RTB（Real time bidding）策略研究
- 流量预估/CVR 预估/投放节奏控制

⑥ 用户/广告体验分析，涉及到：

- 统计理论
- Casual Inference（ 因果推断 ）

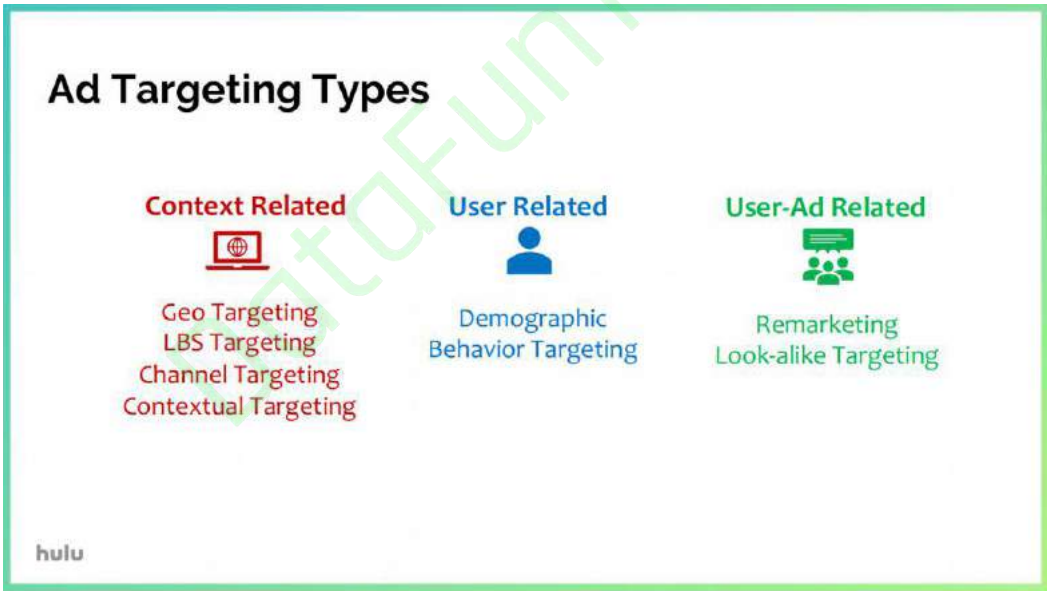
⑦ 价格机制设计，涉及到：

- 博弈问题
- 增强学习

核心业务场景中的算法实践

由于篇幅有限，这里着重讲一下在三个比较核心的业务场景下，算法在 Hulu 广告系统中的实践与落地。

1. Ad Targeting 广告定向



广告定向在品牌类广告中，是最最重要的一个环节。

在 Hulu 的场景下，主要有三种类型的广告定向：

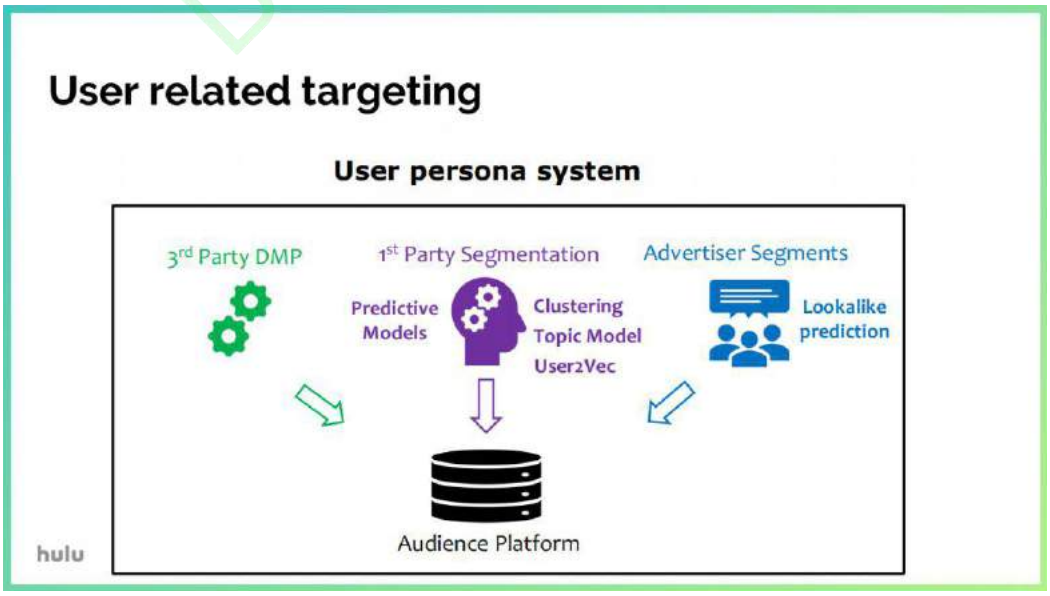
① 上下文相关：例如用户用的设备，在什么位置，在哪个频道，看什么样的视频内容，插播广告点前后的视频内容是什么情景等等。

② 用户相关：例如用户年龄性别、用户基本属性以及根据用户历史行为挖掘出来的属性等。

③ 用户广告交互相关：

- Remarketing（ 再营销 ）：例如投放广告给最近访问过广告商网站的用户，为他加深品牌印象；
- Look-alike Targeting（ 相似访客定向 ）:投放广告给与种子用户相似的受众，扩大受众人数，以及挖掘具有潜能的用户。

下面分别介绍一下针对这三种类型的广告定向使用到的核心算法：



a. 用户相关定向：

我们构建了用户画像系统作为用户定向的基础。底层是一个核心用户平台，负责用户数据的管理，版本控制，每天的 ETL 等等。注入的数据来自三种数据源：

① 来自第三方数据平台的用户标签、属性，

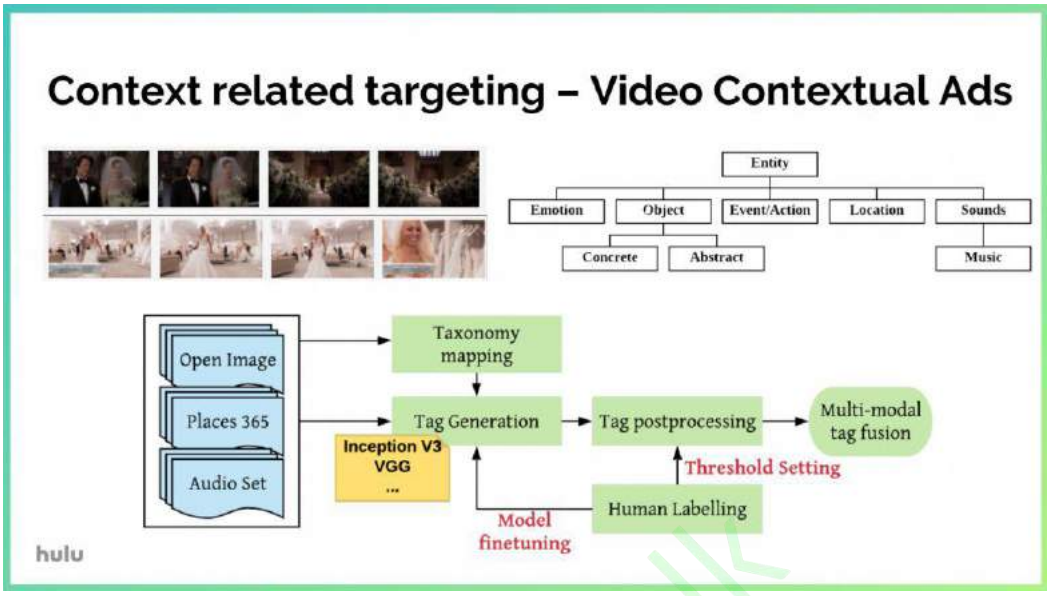
② Hulu 自己本身的一些标签，这里有两类工作：

第一类是工作是标签的补全，我们会用一些例如 XGBoost、DNN 的预测模型来对这些缺失标签的用户进行一个预估，得到用户具有哪些标签，以及概率是多少。

第二类是无监督模式，我们从用户的历史行为以及其他属性中进行挖掘，挖掘出一些新的属性出来，会涉及聚类、主题模型、user2vec 的方式来生成用户的标签。

③ 广告商提供的用户标签，针对这种标签最常用的就是进行 Lookalike 定向，我们会用机器学习的方式来进行预测，找到与种子用户相似的用户群体。

b. 上下文相关定向：



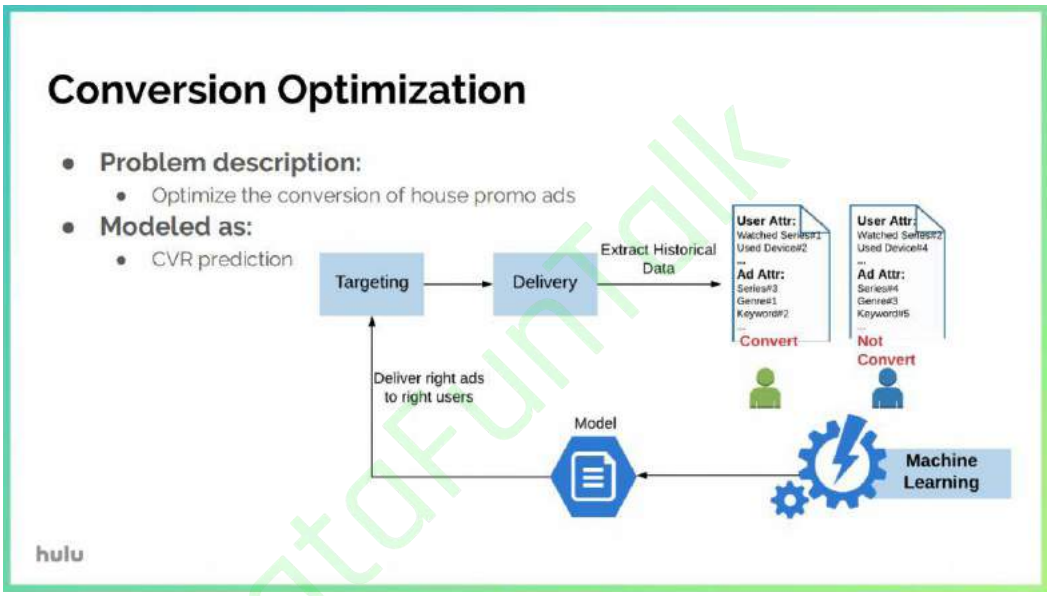
这里主要介绍一下 AI 在视频内容上下文情景广告中的应用。视频广告会插播在视频内容播放前以及播放中， 投放与视频上下文情节比较相关的广告，会减少广告对用户的干扰，举个例子：用户当前所看的是婚礼视频，如果此时投放的广告是珠宝广告，那就毫无违和感，甚至会激起用户的购买欲望。另外，我们需要探测视频中的的一些暴力血腥画面，在这些地方不适合投放广告。

这个问题的核心是使用图像识别技术检测出视频每帧都出现了什么样的实体、情景以及氛围等。Hulu 结合业界的标准构建了自己的 Taxonomy 体系，分为情绪、物体、场景、位置以及声音几个大的分类。

简要介绍一下图像检测大致流程：考虑到获取大量数据样本的人力物力成本较大，我们最开始会借助于公开数据集（比如 Open Iamge、Places 365 等 ）,使用 Inception V3、VGG 等模型去生成一些标签 ,接下来会根据 Hulu

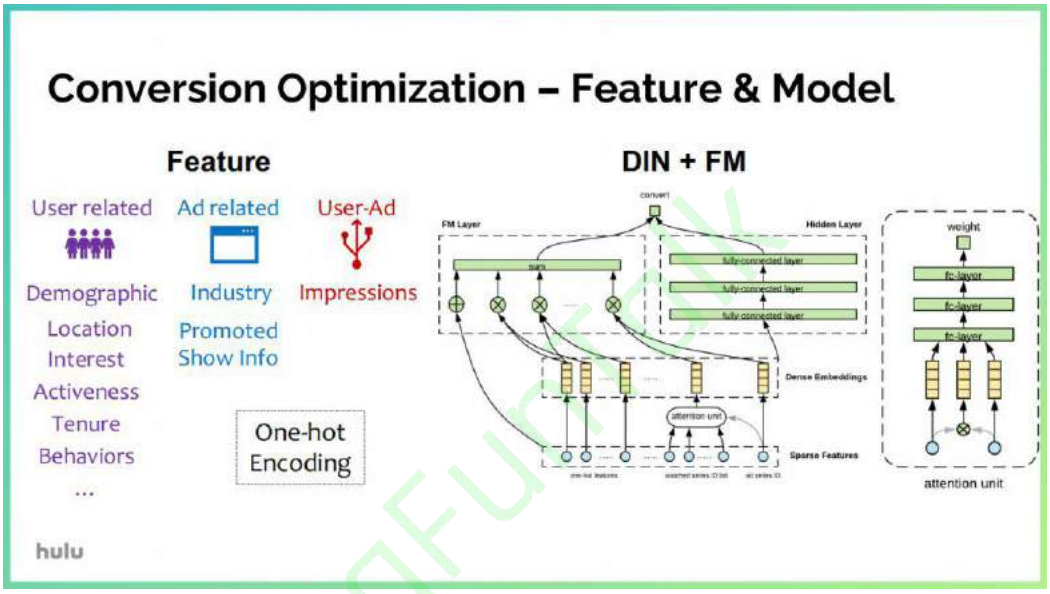
的场景进行少量的人工标注，对模型进行 Finetune 以及使用多模型融合决定最终的标签。

c. 转化率优化：



虽然品牌类的广告点击率没有那么重要，但是越来越多的广告商开始和 Hulu 谈，我把我的转化数据给你，你帮我计算下我这次投放的转化率是多少。这就带来了另外一个问题，广告商最终还是希望优化自己广告的转化率，所以我们最近开始了这方面的研究。目前开展的研究是基于 Hulu 自己的广告的转化率优化，Hulu 在自己的视频里面也会放一些推广新剧的广告内容，例如一个 15 秒的预告，希望这些广告能投放给正确的用户，得到比较高的转化率。这个问题被建模成为 CVR 预估的问题，根据投放数据以及转化数据，通过机器学习来建模用户看到广告后的 7 天内、30 天内的转化率，再投放的时候把正确的广告投放给正确的用户。

转化率优化中的特征 & 模型



特征主要分为三块：

User related：基本的用户属性，如位置，活跃度，留存时间等，以及一些重要的历史行为，比如曾经看过哪些内容，搜索过哪些内容，收藏过哪些内容。

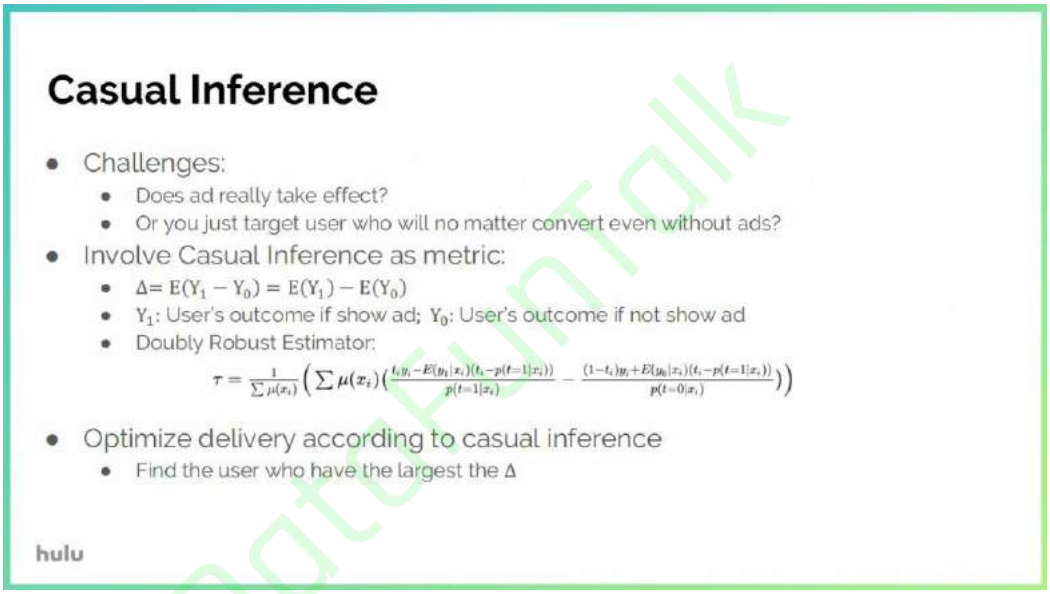
Ad related：广告本身的属性，如行业、剧目本身的信息等等。

User-Ad：用户这个广告看过多少次了，通过多次投放使用户加深印象。

模型方面我们目前使用了 DIN + FM 模型进行建模，同时也在探索更好的模型优化。

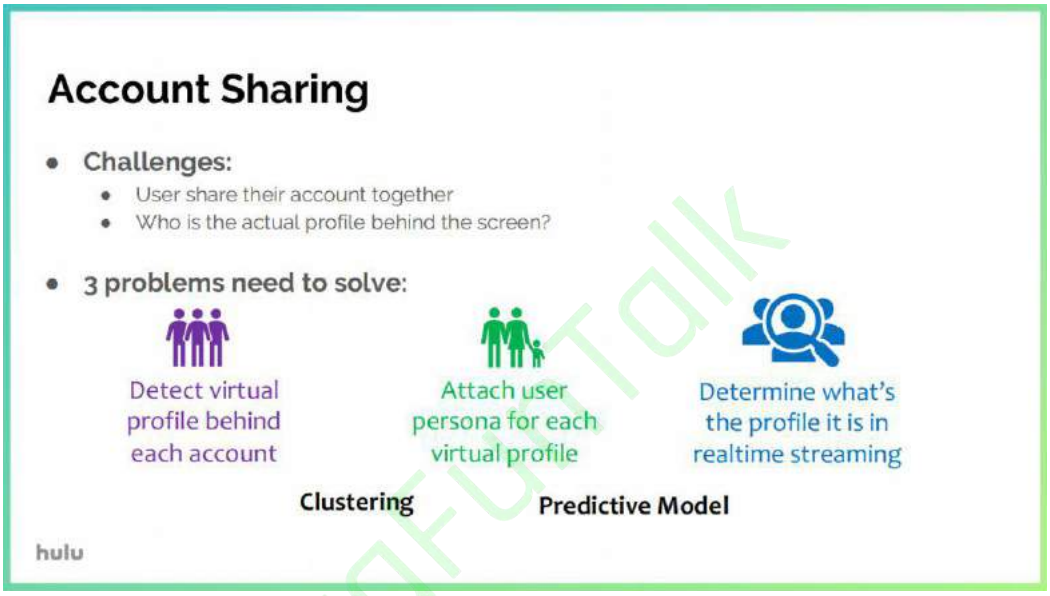
d. 广告定向中不可忽视的问题：因果推断和共享账号

因果推断



我们在评估广告转化率的时候，会考虑评估是否正确，广告是否有效，举个例子：假如有一群用户，不论你对不对他投放广告，他都会去买这辆车，如果把广告投放给这样的用户，最后评估广告的转化率的时候会评估的非常高，其实广告效果一般，因为你投不投他都会去买，其实是白白消耗了这次投放。这就需要引入 Casual Inference 作为评估指标，其核心思想是：评估这个用户看这个广告和不看这个广告的收益分别是怎么样，最后将两者之间的差作为评估指标，使用 Doubly Robust Estimator 进行建模。除此，更加重要的问题是怎么根据 Casual Inference 作为目标去进行转化率的优化，怎么找到这些广告增益最大的用户，然后把广告投放给这些用户。

共享账号



接下来是用户共享账号的问题，例如：一个账号，孩子晚上 8 点左右看动画片，妈妈晚上 10 点左右看综艺节目，爸爸周末看体育视频，妈妈使用该账号的时候，发现推荐的都是动画片，找不到自己喜欢看的内容。这个在推荐以及广告定向中都是非常重要的问题，例如有的广告商，希望这个体育类广告只推荐给家庭中的爸爸，如果是小孩和妈妈看是没有效果的。

这个问题规划到三个方面来解决：

① 探测这个账号背后有多少个虚拟用户

② 给每个虚拟用户打上标签，预测他的行为模式

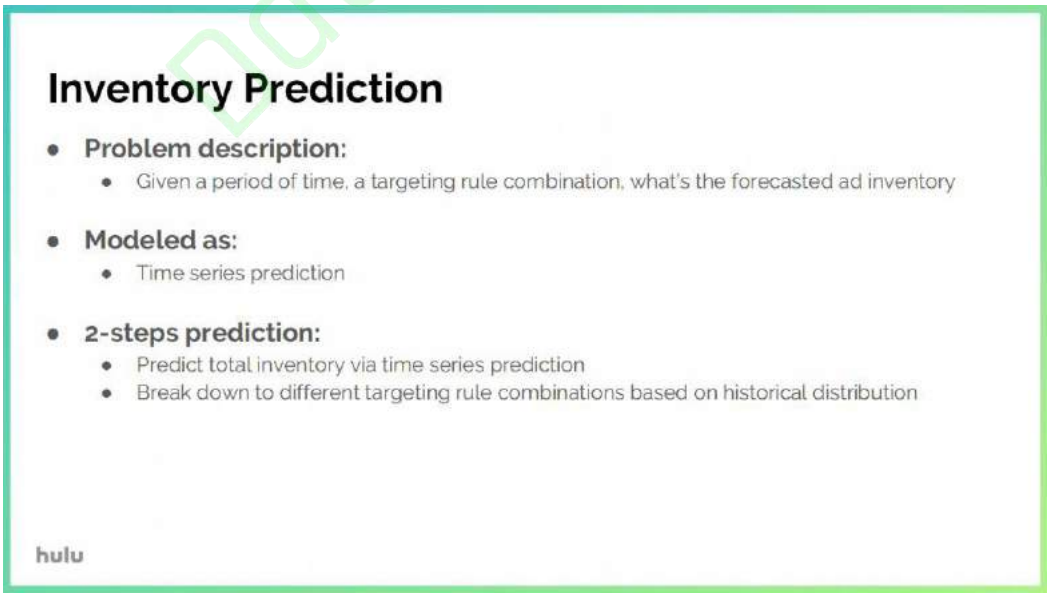
③ 用户来到 app 的时候预估是哪个虚拟用户，推荐对应的广告和内容

这个问题核心是一些聚类的算法和预估模型的问题，这也是我们目前正在解决的问题。

2. Inventory Prediction 流量预估

流量预估在担保式广告中是非常重要的环节，广告在售卖之前都要检查是否有足够多的库存余量卖给广告主，以保证售出的广告都能够保量完成投放。

问题简单描述为：给定未来的一段时间，一些定向条件的组合，预估有多少广告流量库存。



这个问题可以建模成时序预估的问题，通常有以下两个步骤：

第一步：用时序预估模型来预估总的流量是多少；

第二步：根据历史分布把流量分配到不同的排列组合，不同的维度上。

Time Series Model

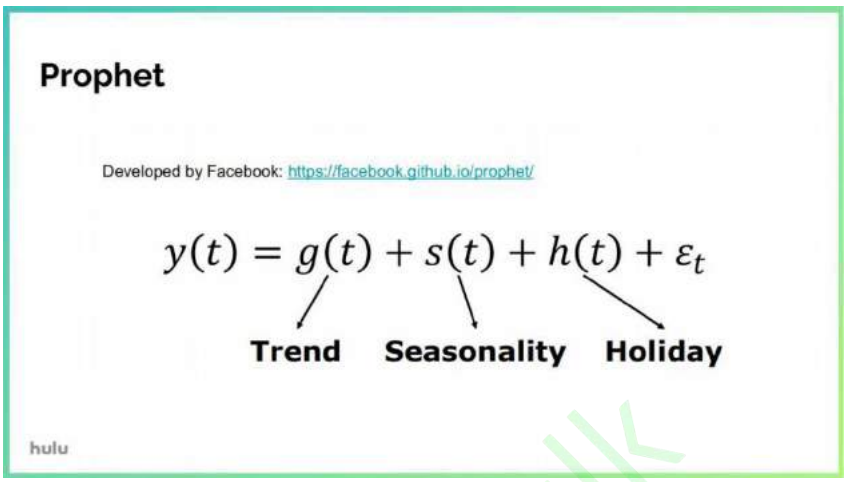
我们目前尝试过以下三类模型：

第一个是 ARIMA（Autoregressive Integrated Moving Average model），稳定时序预估中最常用的模型，第二个是来自 Facebook 的 Prophet 模型，第三个是 LSTM。

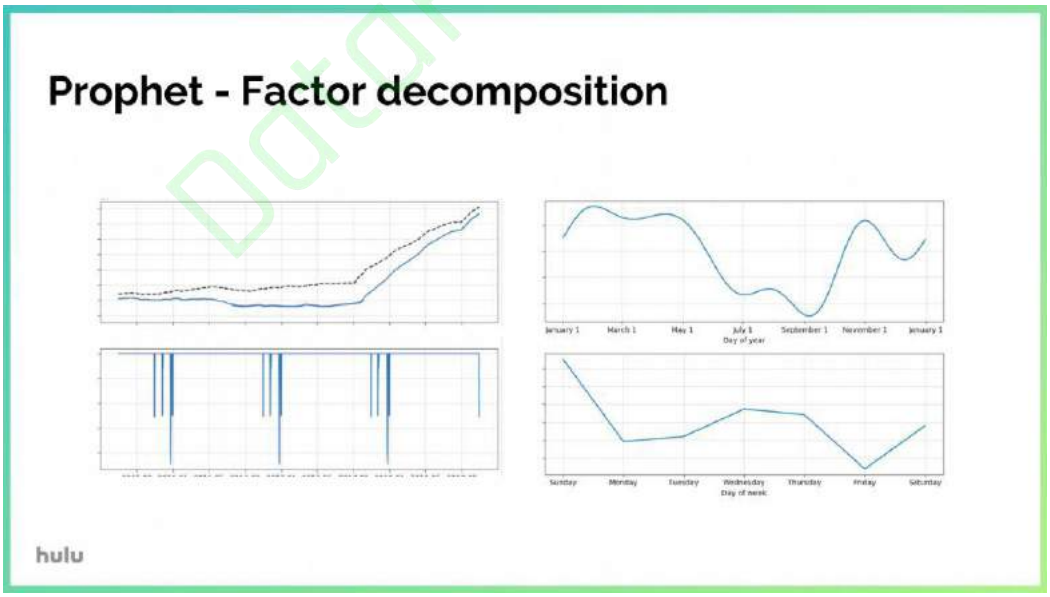
ARIMA 作为最为直接简单的稳定时序预估模型，在我们的实验中，在短期时序的预估准确率很不错，但是对长期的预估相对来说就不那么稳定了。原因在于，以下一些因素都会影响到长期的广告流量时序数据：

- 用户增长的因素
- 季节性的因素
- 每周的波动
- 节假日和特殊事件

Prophe 模型是专门为具有这几种特性的时序数据而设计的时序预估模型。



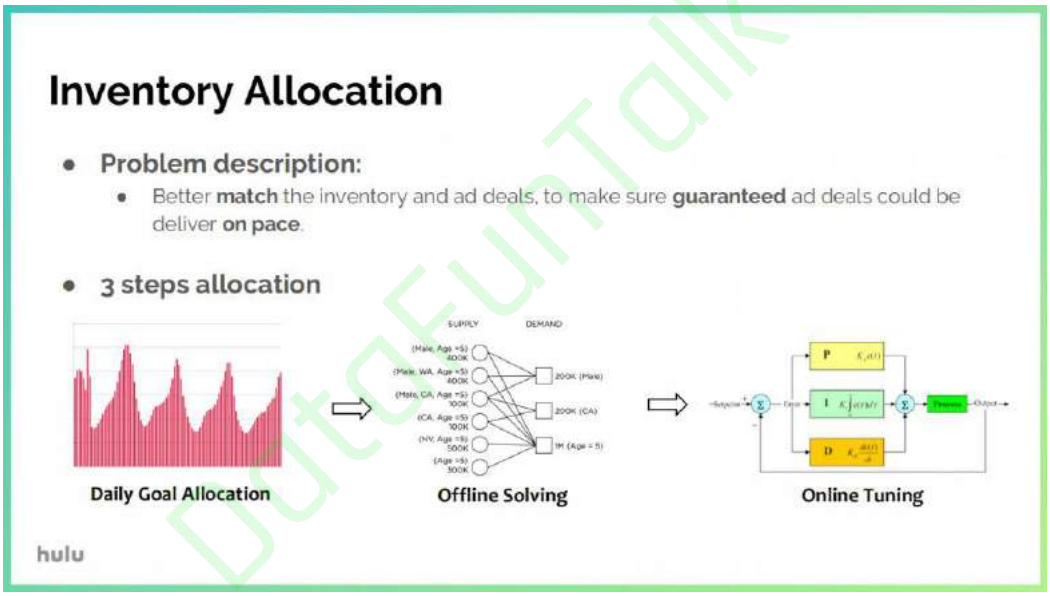
Prophet 把时序数据拆成了三部分：趋势部分，周期性和季节性部分，节假日部分，每一部分会单独做预测模型，最后合到一起得到最终的预测结果。



例如，上图为 Prophet 模型产生的几种因素的分解：左上角为趋势，左下角为节假日，右边为季度性，分别为每个月和每周的变化，可以观察到淡季和旺季的变化。目前，我们采用的是 Prophet 模型，在 Hulu 的广告流量预估问题上表现稳健。

3. Inventory Allocation 流量匹配

流量匹配在担保式广告的投放环节发挥着重要的作用，保证所有广告订单都可以按时按量并且节奏均匀的投放完成。



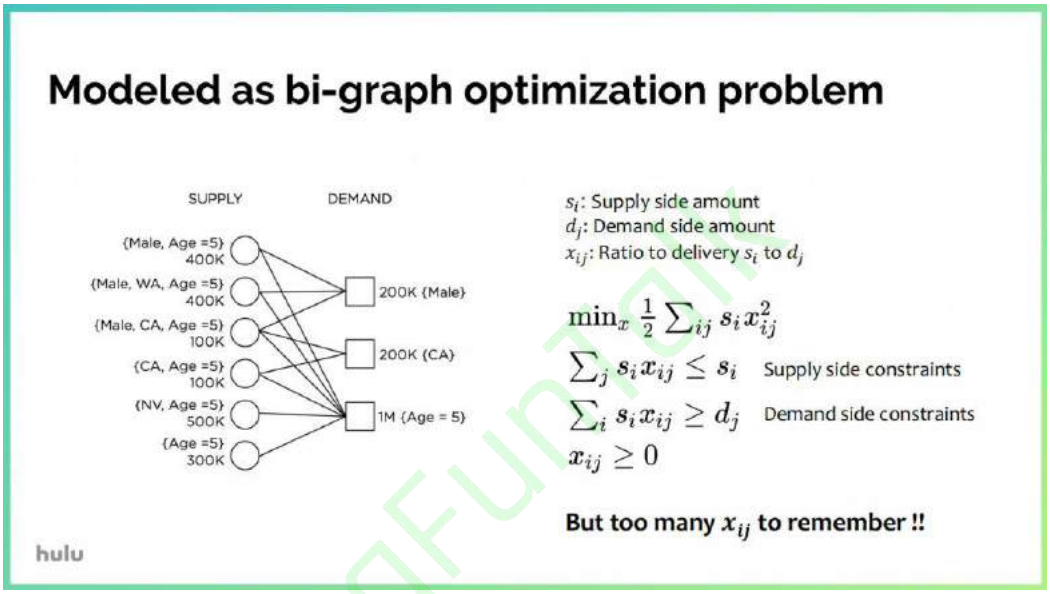
流量匹配算法会分为以下几个步骤：

第一步，把广告订单的投放目标拆分到每个小时，这个拆分会根据我们预估的流量和紧俏程度来进行分配；

第二步，进行 Offline Solving，把问题建模成一个二步图匹配算法，然后使用优化算法进行压缩解的求解；

第三步，进行线上调控，采用 PID（Proportion Integration Differentiation）控制，根据真实的流量实时进行调控。

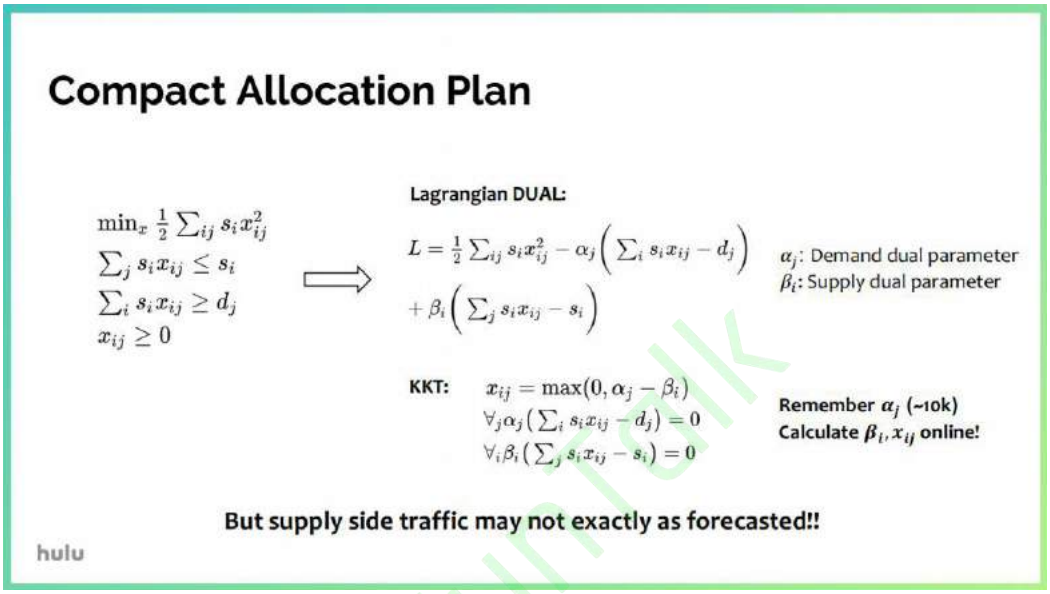
第一个步骤比较简单，下面简单介绍一下第二、三步中具体用到的算法。



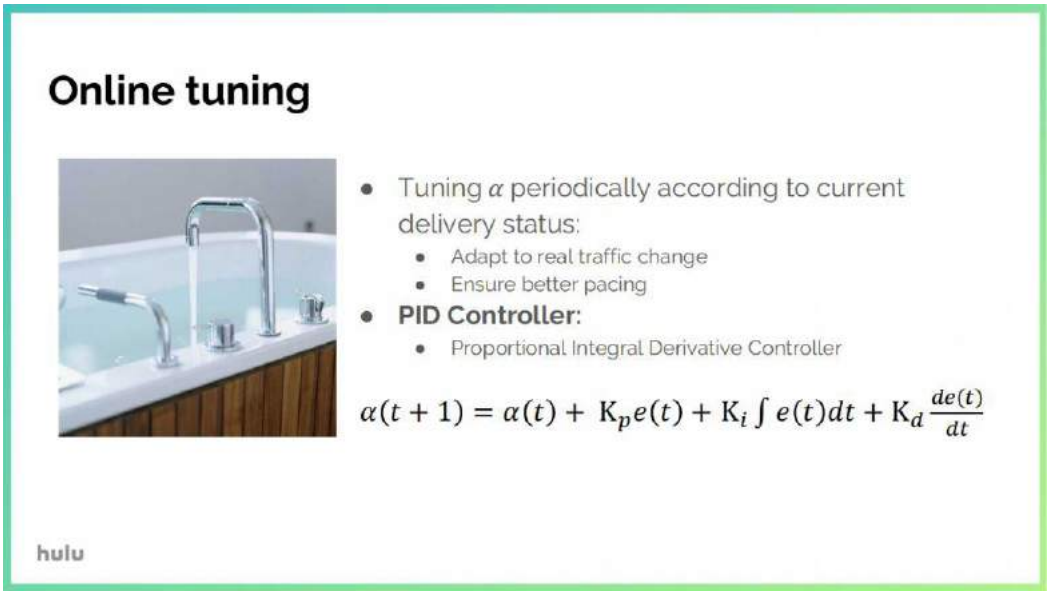
在离线匹配中，我们把问题建模成一个二部图匹配问题。

在广告中最主要有 SUPPLY 与 DEMAND 端的两个限制条件：上图中左边 SUPPLY 表示用户带来的流量，流量上会有一些标签，表示用户及广告位的属性，右边 DEMAND 表示广告商的订单，订单上也会有些属性，比如投给男性，或者某个地区的；SUPPLY 与 DEMAND 之间的连线表示这个广告位可以投放的广告。SUPPLY 端的限制条件为最多只有这么多的库存可以使用，DEMAND 端的限制条件为需要将每个广告订单都保量投放完成。

之后可以根据二部图建模优化问题，求解目标是每个边的权重，表示该流量以多大的比例分配给某个广告订单。



首先会将原始优化问题转化为拉格朗日对偶问题，求解得到对偶变量 α_j 和 β_j，表示的分别是广告的对偶变量和流量的对偶变量。根据最优解满足 KKT 条件，我们可以只将相对数量级较低的广告对偶变量 α_j 记下来，提高存储效率，线上再根据 KKT 条件计算 β_j 和 x_{ij}。α_j 可以被看做各个订单重要程度影响因子，作为线上选择订单的权重。



但是这里还是存在一个问题，这里全部都是用的预估的流量和订单来进行匹配，但实际上会发生一些变化。所以需要第三步，线上实时调控。第一可以保证适应真实的线上流量， 第二也可以进一步保证投放节奏的稳定性。一般使用 PID（Proportion Integration Differentiation）控制技术，这是常用于机械、电气系统中的一种应用非常广泛的自动控制技术，分为比例调节、积分调节、微分调节这 3 项的调节，根据线上广告真实投放的快慢来调控接下来应该投放的速度。

以上就是今天的所有内容了，由于篇幅有限，计算广告以及视频广告系统中很多涉及到的算法没有办法一一覆盖，也有很多更有挑战的算法问题亟待研究员和算法工程师们来一一解决。本篇文章作为抛砖引玉，欢迎同行进行探讨指正。

DataFunTalk

微博广告策略工程架构体系演进

嘉宾：李铁牛　　编辑：崔媛媛

导读：本次分享的主题为微博广告策略工程架构体系演进，将介绍微博广告在从 0 到 1，从 1 到 N 的过程中，微博广告架构是如何支持策略、算法、模型迭代的，包括以下几部分：

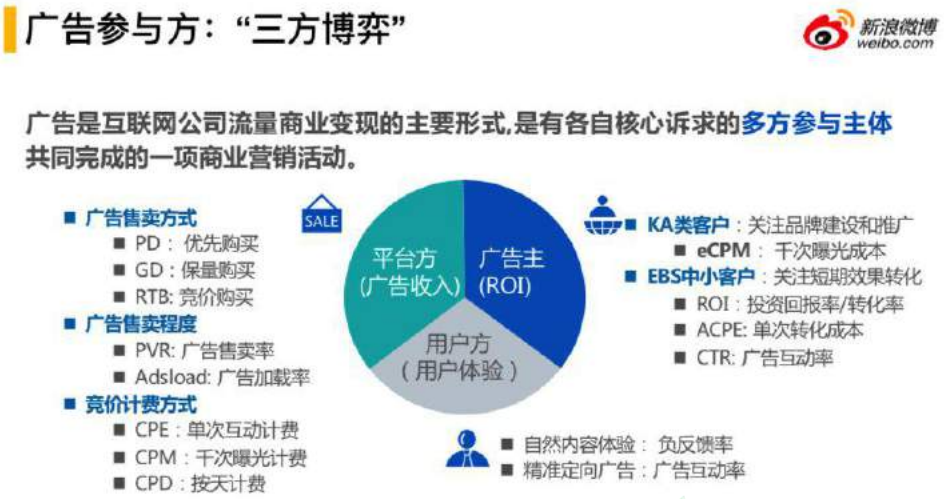
- 概述
- 微博广告策略工程架构体系演进
- 精益驱动思想工具：“两翼计划”

概述

1. 广告样式与场景

上图是微博广告目前商业场景流，“一屏四大流”。“一屏”指打开微博的 Fashion，“四大流”指占据微博商业化的主体，包括关系信息流、热门流、评论流和热搜流。右图为广告投放的后台。

2. 广告参与方

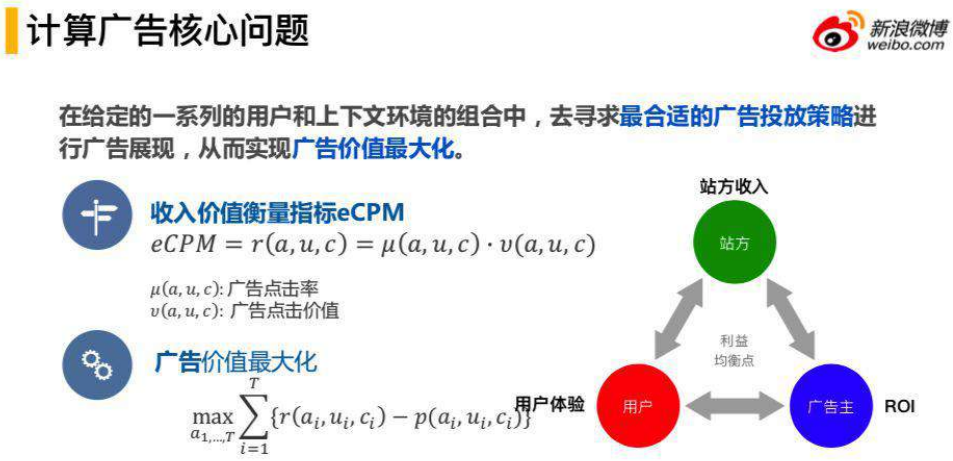


如上图，做计算广告首先面临这些概念，根据不同的广告主的规模和对公司的重要程度分为 KA 类和中小类。KA 类往往是进行广量式的购买。中小类，常规的客户会进行竞价。

常见的计费方式有：CPE，CPM，CPD。

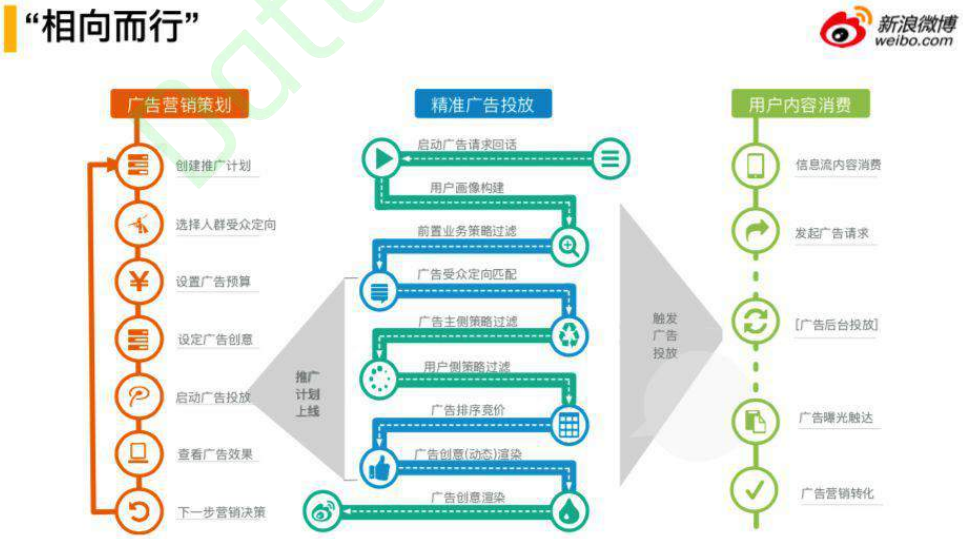
目前在互联网市场在大规模推广 OCPX，OCPX 作为一种需要很高的技术含量，也是一种很好的降低广告主投放风险的售卖方式。

3. 计算广告核心问题



这是广告涉及到的三方：平台（站方）、用户、广告主，在计算广告设计时的核心问题是如何追求三方之间均衡的、整体的利益最大化。

4. 广告投放流程



以上是一般的广告的投放流程。这是从广告主的视角、平台的视角以及从用户的视角来公共完成三方“相向而行”的活动：

广告营销策划的流程：创建推广计划 -> 选择人群受众定向 -> 设置广告预算 -> 设定广告创意 -> 启动广告投放 -> 查看广告效果 -> 下一步营销决策。

精准广告投放：针对广告库存请求，会对用户进行精准用户画像刻画，然后做广告的召回，对广告进行粗排和精排，挑选广告，根据不同平台进行广告创意渲染工作，最后展示给用户。

用户内容消费，比较简单请看图中流程。

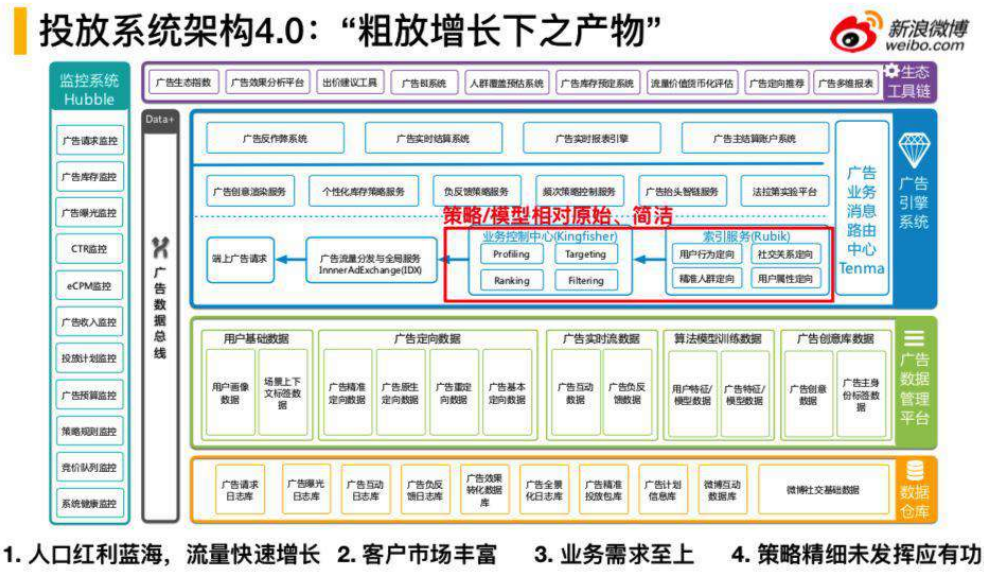
微博广告策略工程架构体系演进

1. 微博广告工程架构发展史

微博商业化进程不断的发展，支撑商业化的工程架构会随着具体的业务需求做改变。

刚开始做的是非信息流的广告，按照传统的方式会试投一些 banner 在微博，目前 banner 在微博的移动端已经没有了。从 1.0 版本简单弹窗式的广告系统，到 2.0 版本以粉丝通为代表的产品线陆续的孵化出来，这时候微博开始进行信息流广告的研发。微博是国内信息流广告的第一家，在探索中孵化出一系列的广告产品矩阵，为了产品的快速上线，复制了大量的广告系统，为了改变这一状况，随着 17 年底超级粉丝通的上线，对广告系统进行了整体重构，从此微博广告工程架构进入了 4.0 时代。

2. 投放系统架构 4.0



上图是 17 年的工程架构图，是随着微博广告产品线的探索演化出的工程架构 4.0 版本，当时处于流量的蓝海阶段，所以通过配合广告客户和广告预算的不断增加，不断提升广告系统的稳定性、高可用性、高并发来不断的实现广告收入的增长。因此我们也对系统进行了分层化的梳理，图中蓝色的区域是线上广告投放系统最核心的链路。通过广告请求，流量统一接入，接入包括微博多个产品矩阵，广告的请求会对多个产品矩阵进行请求分发，通过统一流量价格评估，对用户的请求进行响应。整体上这样一个引擎结构是为了不断的满足产品提出的面对客户的需求来进行的架构设定。

其基本流程是广告请求，广告库存接入，到总体流量的分发，请求用户的画像，经过竞价服务，会进行广告触发，会请求在线索引服务，形成比较完善的行为定向体系，包括：

1. 用户的行为定向。在微博上的行为定向，比如，对话题的互动，热门微博的互动，热搜人群的互动等。

2. 社交关系定向。比如关注的大 V 信息，如果在某个大 V 信息下用户的群体能理解为自然形成的社会群体，那么这个大 V 信息下面的粉丝群是可以进行选择投放的。

3. 精准人群定向。是由平台方或第三方的数据加工，或广告主根据一次投放效果进行的召回，或已有的客户信息形成的用户粒度聚合后的数据包，这个就是精准用户的集合。

4. 用户属性定向。包括用户画像，年龄、地域等。

以上是整体的在线投放流程，但是投放流程仅仅有以上这些是不够的，还包括个性化库存策略，广告的负反馈的策略，智能频次控制策略等，以及配套的 A/B Test 系统，这样就形成了广告投放的在线服务群。

由于流量来自于微博站方，所以微博广告请求是无需流量反作弊的。存在的反作弊主要是针对互动回传，也就是说广告投后的后链路数据的回传会有大规模的反作弊策略，当然也包括社交互动。然后会有一个实时的结算中心—结算系统，提供给广告主需要的报表，以及与广告主密切相关的账户系统，总体上形成了投后链路在线服务群。

之下是属于近线的数据访问，按照数据分类：用户基础数据、广告定向数据、广告实时流数据、算法模型训练数据、广告创意库数据来制定线上这种实时访问的需求。

最下面属于线下的数据仓库。线上投放完的数据会进入数据仓库落地。

这就是广告数据总线，实现数据流的方式一般通过 kafka 机制等实现，然后汇聚到数据仓库里，对数据进行分门别类。

图中最左边广告的监控系统，会从系统的各个层面对系统进行业务运行状况的监控以及服务稳定性的监控、可用性的监控。这些就是业务层面完整的一个工具链。原来多个产品线就逐渐聚合到这样的一个系统当中。

DataFunTalk

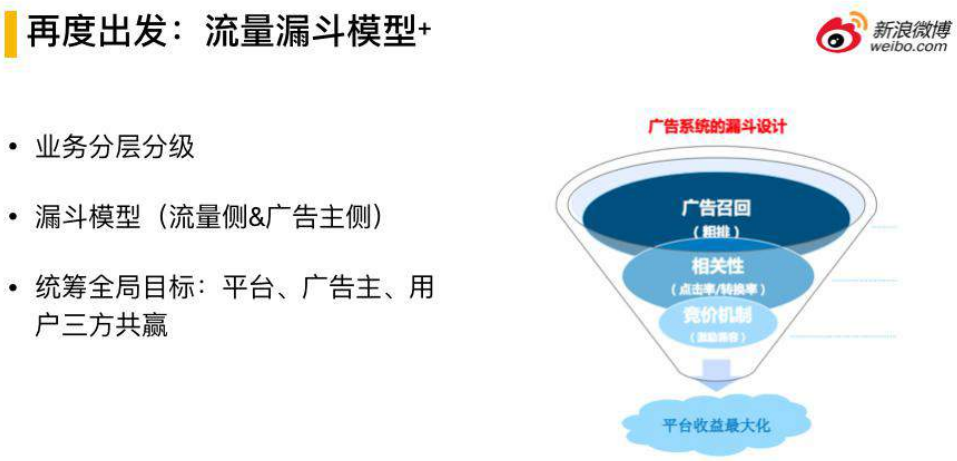
4.0 时代的架构在整体上是为了“粗放式增长”而设定的工程架构体系。这种粗放增长的客观现实是，广告预算供应的持续提升和微博不断供给的流量变现规模，以及不断增加广告主的数量、预算和规模，来实现广告数的增长。这时对系统最大的考验是系统的高可用性，以及做业务需求时，对研发效率的保证，这样的架构就是“粗放增长之下的产物”。

这个架构体系是存在一些问题的（即红框中的）：相对来说对策略模型是相当淡化的，也就是从功能架构层面对算法模型的迭代是比较简单的。比如 A/B Test 使用的是非常原始的 A/B Test，在这种在人口红利的情况下能够快速支撑广告业务的增长，但是随着人口红利的消失，已经不能很好的支撑广告业务的增长，这时系统对策略模型的支持显得异常重要。

3. 系统如何支持广告增长的转型

如何支撑广告增长的转型，会通过粗放型增长（扩流、扩广告主、扩预算）转型到精细化的增长，不断的提升投放效果来促进收入的增长。这时对应的系统需要进行转型，在系统不断完善的基础上，实现策略模型的良好驱动。在这种情况下随着算法不断引入新的深度学习模型，整体的工程架构也不断的深耕细作，从原来的业务划分方式（Target、Filter、Rank）转型成面向算法策略（召回、模型、机制、排序）的划分方式。

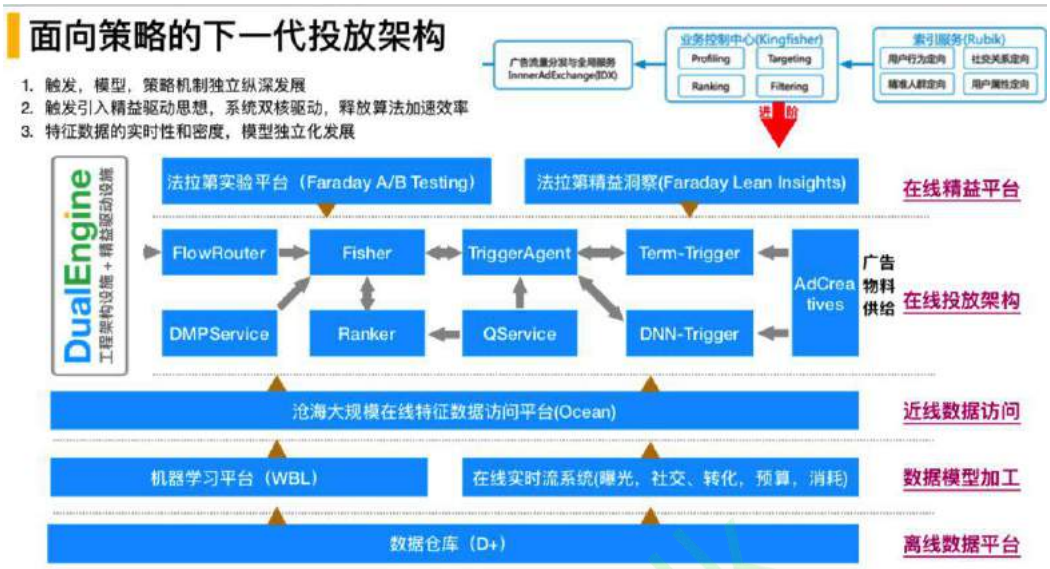
4. 流量漏斗模型+



广告系统常用的模型：流量漏斗模型。对其重新进行思考和定义：开始是对广告进行召回，完成基于精准的最大概率展现，到相关性的挑选，再到以模型为核心的竞价排序机制。

本文不会从算法角度去讲解如何召回和相关性的机制等，主要是介绍工程是如何支持算法模型迭代的。

5. 面向策略的下一代投放架构



在架构系统 4.0 中的基础上对在线投放引擎进行业务分级，以满足新的流量漏斗模型。有以下关键点：

① 触发、模型、策略机制向独立纵深发展

系统在支持触发、模型、策略迭代上能够实现很好的各自的独立纵深发展，能够做到各自的快速迭代，互不影响。

② 引入精益驱动思想，系统双核驱动，释放算法迭代效率

在做整体的精细化转型的时候，系统需要不断的进行尝试，而尝试要有一个很好的尝试平台，所以引入了精益驱动的思想。在线精益平台包括：法拉第

实验平台和法拉第精益洞察，这是一个比较好的促进业务模型迭代的工具链，更加注重数据的实时性和数据的密度。

系统架构总体上分为：在线精益工具平台、在线投放系统、近线数据访问、数据模型加工（曝光机器学习平台和在线实时流的机制）和离线数据平台

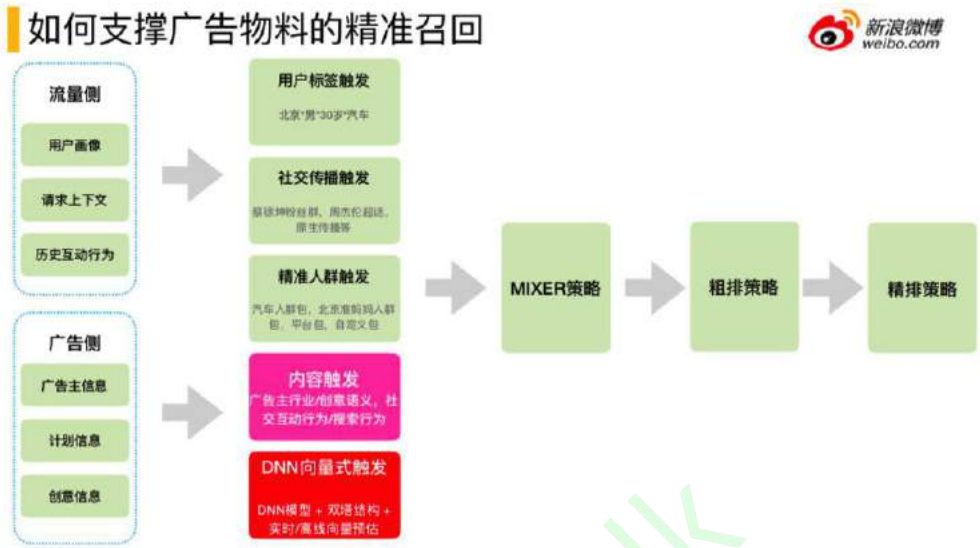
③ 特征数据的实时性和密度，模型独立化发展

重点说下在线投放服务，在服务中，会有流量接入，会有流量的触发，会有触发机制，包括多路触发，通过多路触发体制后，会有机制策略，包括模型预估服务，模型预估服务是聚合服务，会进行粗排、对数据进行裁剪，会在大规模分布式预估服务中完成，Ranker 也会基于预估进行精排。

特别说明一下，为什么进行粗排和精排，粗排我的理解是为了精排的性能考虑的，因为精排会涉及到大规模的精细计算，性能有可能会扛不住，所以需要粗排，而且在保证效果的情况下，为了性能的保证会有多级粗排。

最上面是法拉第实验平台和法拉第精益洞察。整体上会形成双核引擎：一个良好的工程架构体系和精益驱动的工具平台。

6. 如何支持广告物料的精准召回



从微博广告的召回机制来看会有用户标签触发、社交传播触发、精准人群触发、内容触发、DNN 向量式触发，经过 5 路触发，会进行 MIXER 广告召回级的汇总，汇总后会有粗排策略、精排策略。

这里用到的信息，包括流量侧和广告侧。

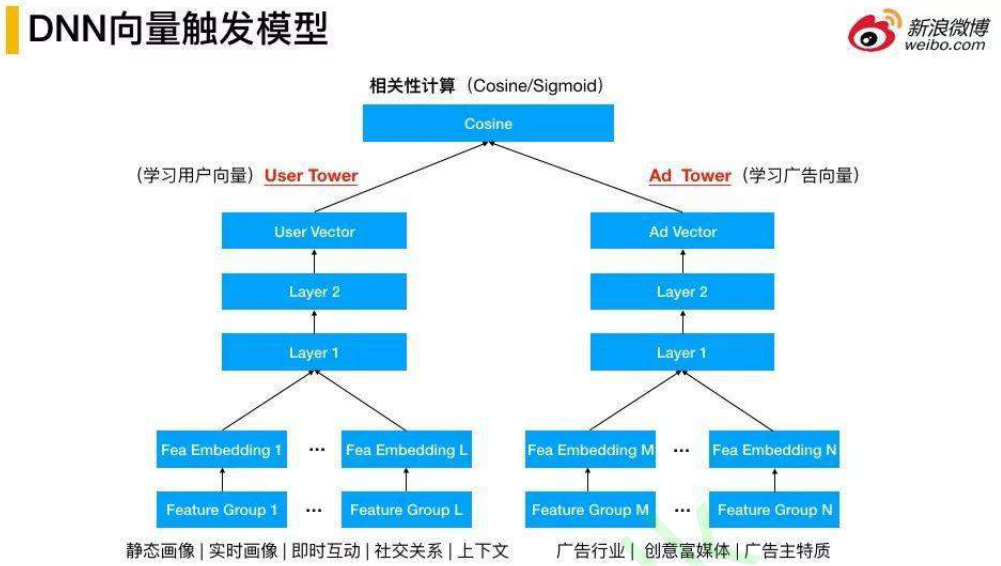
流量策：

用户画像、请求上下文、历史互动行为

广告侧：

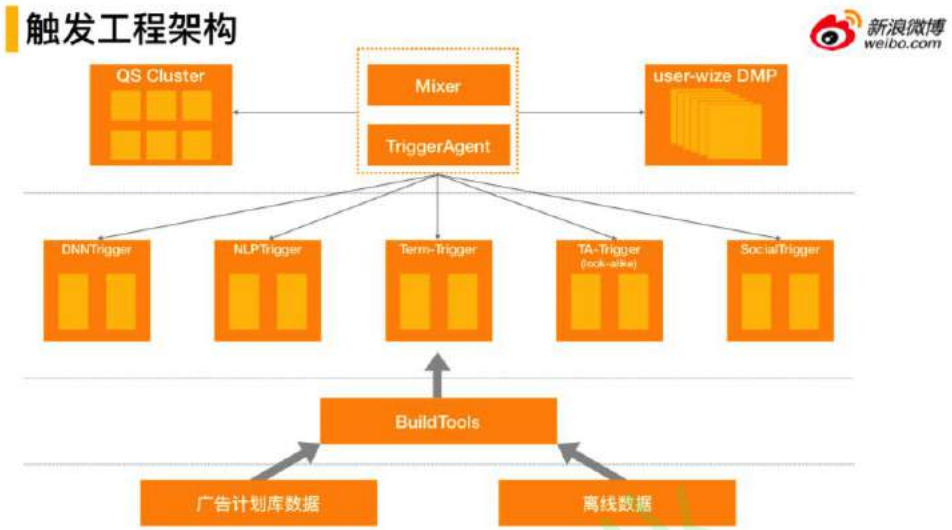
广告主信息、计划信息、创意信息

7. DNN 向量触发模型



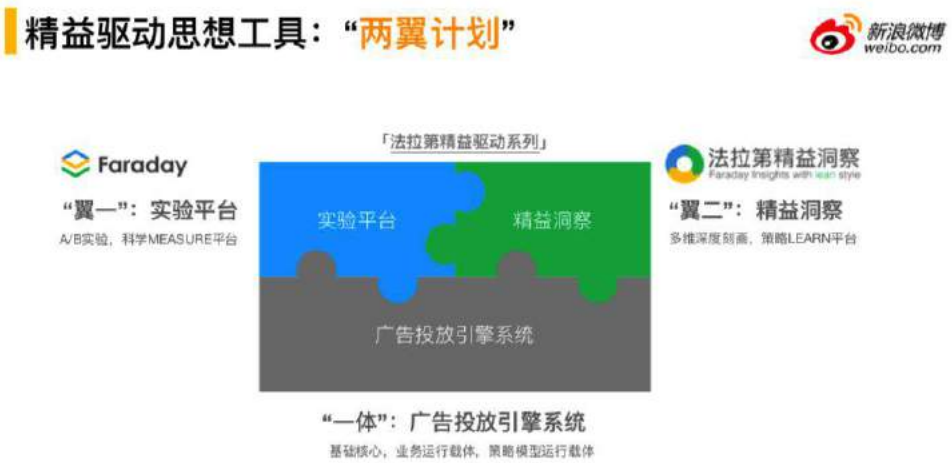
这里介绍下深度向量触发模型，用到的是双塔模型，包括用户侧和广告侧。用户侧根据用户信息进行训练生成用户侧的向量，用到了三层的神经网络，广告侧也是一样，整体来说训练都是采用离线的方式完成的，接下来会做实时的向量预估。进而在双塔汇合点进行相关性的判定，使用了简单的 cos 和 sigmoid 进行相关性的判定。

8. 触发工程架构



触发工程体系在如何更好的支持触发角度来说，研发了对应的服务体系。请求在请求召回时会触发 Agent 进行五路召回，包括双塔的召回、内容定向的召回、用户画像的召回、精准人群的召回、微博社交关系的召回，召回之后进行 Mixer，会结合质量预估服务进行裁剪。然后，线上会有实时的广告计划库数据和离线数据会对线上的五路触发分别根据需求进行访问，这样会形成一个整体的广告触发工程架构。

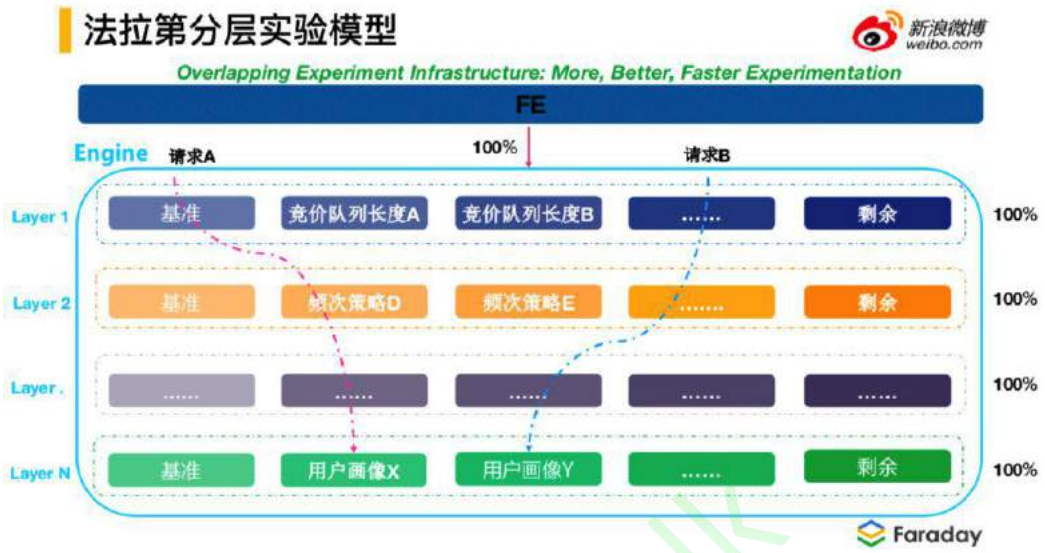
精益渠道思想工具：“两翼计划”



随着超级粉丝通在 17 年上线，考虑到微博广告从粗放式增长到精细化增长的转型，需要有一个比较好的实验平台和一种线上运行洞察的方式和思想，这样就构成了精益驱动思想的来源。包括两部分：一是进行线上策略的实验和调控，一个是用于线上策略运行的精益洞察，与在线投放系统一起，构成一体两翼策略工程架构体系。

1. 法拉第实验平台

① 法拉第分层实验模型

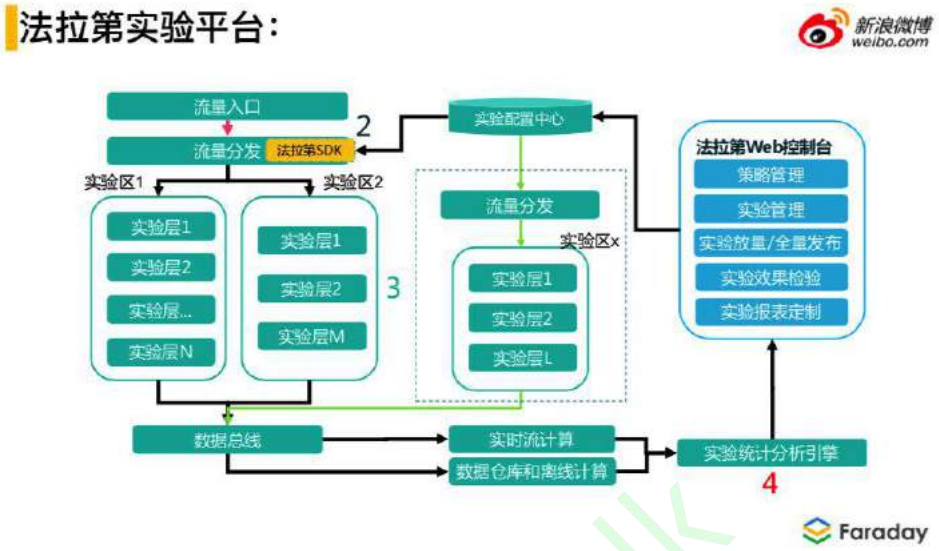


法拉第实验采用正交分层模型，这种模型在互联网行业一般都会使用，包括百度的爱迪生、阿里的特斯拉等。微博法拉第模型思想来自于 Google 流量正交分解模型的经典论文，为多层的独立实验提供了理论基础，当然实验也结合了微博广告的实际情况，包括刚开始解决从无到有的问题，对论文中的模型进行了简化。最开始是没有域的概念，整体上每一层都会使用 hash 函数。

② 实验分桶

虽然每个实验层共享 hash 函数，但是 hash 函数参数不同，参数包括流量标识和实验 id 标识，呈现出的是不同层的流量分桶划分是正交的。另外，也引入 Google 论文中提到的分配条件，场景应用很经典，例如做一个实验，会考虑流量的重用或者实验的一些特征，包括实验的地域、性别等，这样的话，使用圈定或限定下的流量而不是使用全部的流量；如果使用全部流量的话会造成实验效果不显著，实验效果被稀释，容易引起不置信。这样可以使流量重复复用，并且能很好的观测到策略所产生的实验效果。这就是流量分桶类型和进行流量圈定的分配条件。

③ 法拉第实验平台



这个是法拉第实验平台的整体架构图。采用全程自动化的机制。通过法拉第的 Web 入口，进行实验信息的记录，在线上流量入口处进行实验的下发和解析，根据实验的数据信息，进行线上策略的调控，会对实验命中数据进行解析，进行实验的埋点，然后进入实时的分析引擎，统计实验效果。

实验的下发和解析有两种方式：

一种是在流量的总入口处统一进行实验的下发和解析，一步到位，然后请求信息一块下发，随着请求链路走，最后返回，命中那些策略会有对应的标识。如果实验数比较少时，解析无压力，实验相关性解析比较小，实验消耗带宽不大，此时是合适的。但是随着实验规模的增大，由于现在的广告系统是分布式的系统，如果完整的实验信息一直随着请求下发的话带宽消耗会非常严重，造成返回结果超时，可用性下降，实验时间就会变得很长。因此出现了另一种方式，由对应的服务分别对对应策略的实验情况进行解析，而其他策

略的实验情况则不需要解析，这样就只获取自己感兴趣的信息，避免了信息的冗余。

为什么刚开始没有进行这样的设计，因为系统刚开始是解决从无到有的问题而不是一到多的问题。第一步是实验平台的创建，采用第一种方式比较简单，可以很快的将实验下发。

④ 广告侧 A/B 实验

常用的 A/B 实验是关于流量侧的分桶实验，根据流量的不同配比进行不同实验的对比。但是这种流量侧的分桶实验不能满足广告的一些需求。在实际的广告系统中，会出现针对某些广告行业或者广告主，进行尝试性的策略，关注的实验效果也是被关注的广告行业或者广告主的效果。如果用流量侧实验的话也不是不行，但是在进行实验效果分析时候，数据分析需要具体到广告

主粒度，会给统计分析引擎带来极大的挑战；如果不这么做，仍然只看整体的效果，实验分析效果必定是被稀释的，无法准确的反映出策略的意义。这样就需要进行广告侧实验的机制设计：

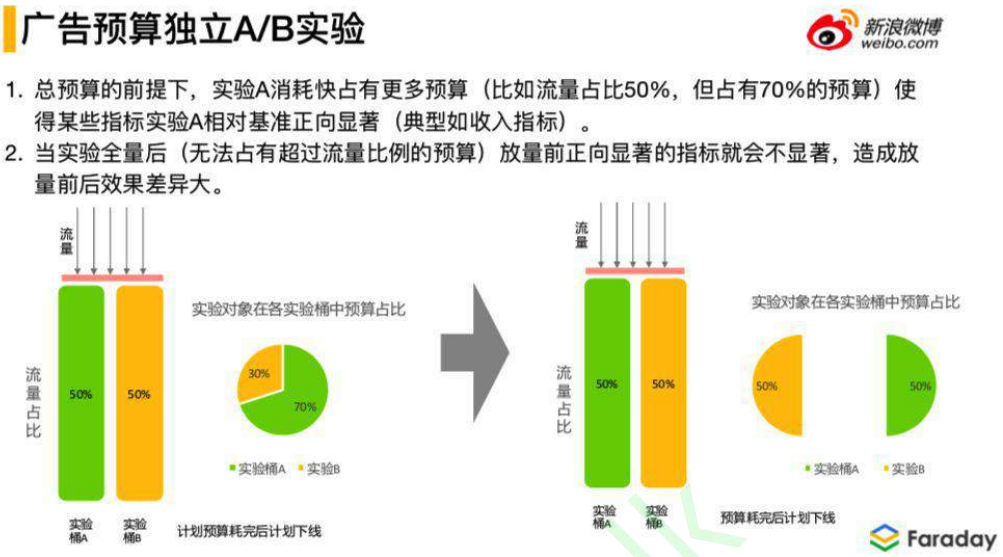
首先会进行实验对象的圈定，包括：广告主 、广告类别、广告计划、广告创意等。

然后进行同质或非同质实验：

同质实验（即对一个推广计划的受众进行随机的划分，一分为二，在一分为二的基础上作用不同的策略进行对比），一个计划，可能会创建两个类似于“伪计划”的概念，然后对伪计划同时进行系统投放。进入两个人群的划分当中，然后进行效果对比。

非同质实验，这个是尝试性的圈定一波人去进行实验，这种往往是没有对照的是根据实验效果进行纵向的对比判定，也就是当前的策略实验效果与之前没有进行的实验的策略运行效果进行对比判断。

⑤ 广告预算独立 A/B 实验



广告实验平台在具体的广告实验中会遇到的问题，这种问题在非广告业务中是没有的。因为在广告领域涉及到 A、U、C 这三方，这三方是个有效的完整回路，一个简单的回路就考虑到计划的投放->花钱->广告下线，这里会涉及预算，预算没有了就会下线。这里会涉及个问题，如果在进行这种简单的流量分桶实验的话，一分为二，上不同的策略，这两个策略的结果对广告主预算的消耗速度是不一样的，带来的结果是，假如某个分桶（五五分），一段时间会发现 A 桶消耗的快，如果没有进行预算的独立，会把 B 桶的预算拉过去，具体表现就是广告主的钱，在 A 桶花费的更多，如果流量小的时候效果可能非常好，但是随着流量的不断的放大会发现策略的实验效果会变的很小。所以广告预算的独立性会在广告主侧实验的时候就考虑到，比如说计划的一分为二，预算平均分配，各自独立，互不影响。最后会考虑将预算独立实验应用在流量侧分桶，目前没有做但是经过论证是可行的。

⑥ 实验检验与效果评估

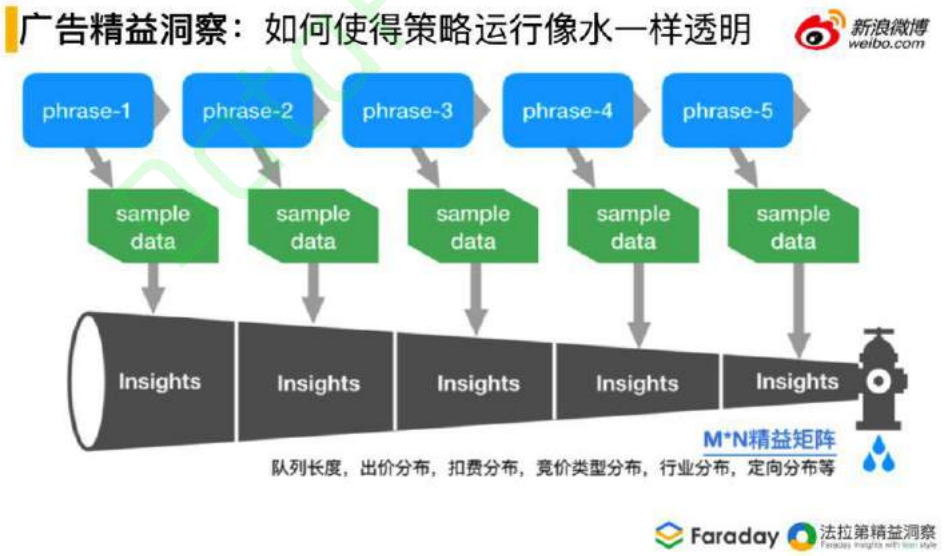
实验检验与效果评估，是实时的效果评估，能够做到 5 分钟（保守的）内的数据延迟，如果 1 分钟也可以，但是问题点在于涉及广告效果衡量时，比如 ctr=互动/曝光，互动可能会有延迟，会造成计算出的结果飘忽不定，所以为了让实验效果趋于稳定会 5 分钟计算一次。

对于跨多天的实验效果分析采取的机制是离线跨天+实时流分析引擎。由于实时流机制的效果不如离线的，这里又进行了优化，比如几个小时之前的，如果离线进行这种任务的调度都能跑出来的话，也会用离线的方式进行批量的处理来替换今天早些时候的实时流分析数据，实时流会倾向于用最近比如一个小时或两个小时的数据。在实验决策角度来说，实时流主要是看实验效果会不会带来严重的缺陷，以便进行实验的及时终止，离线的数据处理带来的实时报表主要作为实验效果的评定。

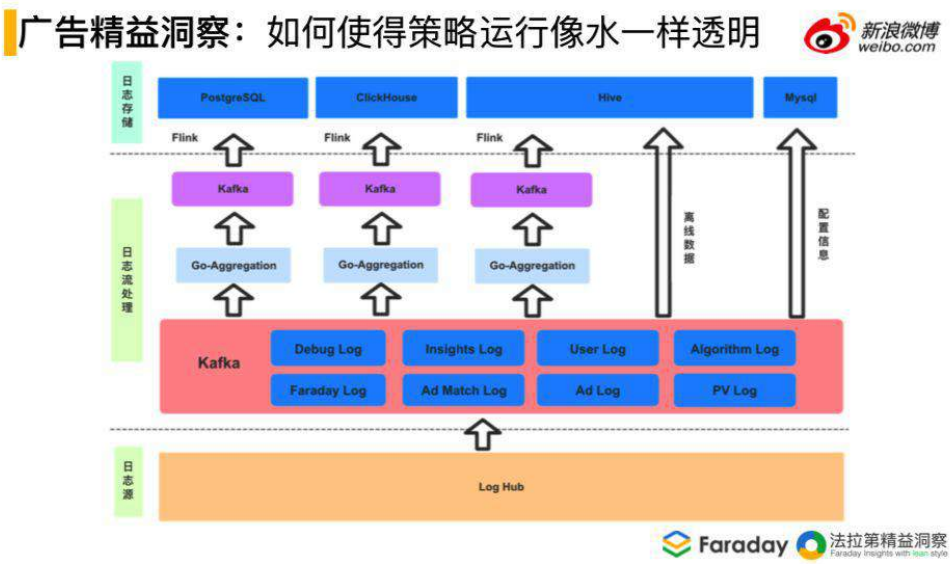
整体的实验效果会有可用性的监控（例如对流量分桶的监控，不同策略对流量层面的可用性是不一样的），保证两个实验分桶的可用性在同一个水平上的。

实验平台支持的业务指标将近 4000 万 ~ 1 亿的规模，而且支持自定义指标的方式。同时实验支持版本的跟踪，也支持 nodiff 的对照实验等。

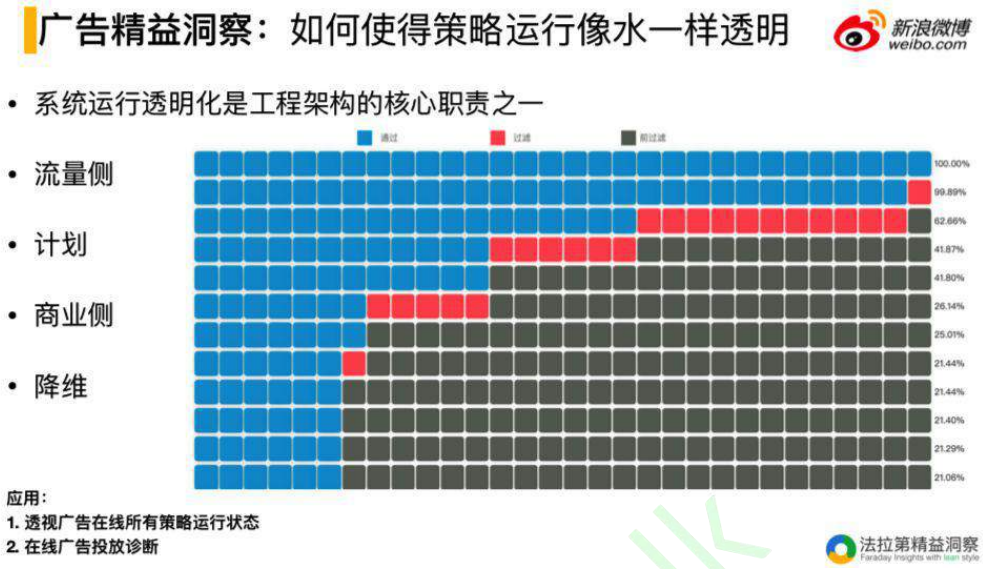
2. 广告精益洞察



精益洞察想解决的问题是：由于线上的策略很多，怎样让策略运行的时候，在系统之外能很好的看到系统的运行状态，使得策略运行的像水一样透明，后来提出了精益矩阵的描述方法，整体上的业务分为多层次的业务阶段，每个阶段都会有完整的洞察。



上图是系统的架构图。数据落地，进入到实时流机制，实时流机制中的数据类别是分类的，之间是通过 key 进行关联，将日志构建组件化，包括线上的调试日志，用户的信息日志、线上跑策略跑模型的日志、广告日志等等，再往上会进行日志处理存储 ,分为离线存储和在线存储 ,在线存储会用到 PG 和 clickHouse 这种列式存储，以便在线即时访问。



整体的呈现效果是：每一层是一个业务策略阶段；请求访问或者广告召回集的处理，整体看是一个漏斗，从广告数量来看是二维的，一个广告召回集的数量，随着策略的运行，数量在不断的减少。每个层面上都可以更细粒度看，例如不同的竞价类型占比的分布情况以及整体的出价水平都可以细粒度的查看，分辨出在不同策略层之间的表现差异，洞察策略的具体影响效果。

精益洞察提供了不同的视角：

流量侧：看总体的流量的漏斗运行情况；

用户粒度：实现 debug 功能，对单个用户的请求在策略漏斗中的表现情况；

计划粒度：商业侧，比如某广告主计划的推广运行情况都可以洞察到。

本次的分享就到这里，谢谢大家。

AI 在爱奇艺视频广告中的探索

嘉宾：刘祁跃　　编辑：李婷婷

导读：本次分享的主题为 AI 在爱奇艺视频广告中的探索。AI 可以对视频内容、广告素材进行理解和加工，并应用于视频广告，尤其在广告的点位挖掘和生成、点位分析、素材创作等方面已产生价值，进而有益于广告库存提升和收入增加。本次分享的主要内容为：

- 背景介绍
- 场景：生成/推荐点位
- 效果：辅助广告素材创作

背景介绍

1. 广告的目的

DataFunTalk

广告是在一定周期内，让尽可能多的人产生消费的一种方式。从周期来看，这个周期可长可短，长周期广告更多是品牌广告，目的是让客户记住该品牌，当产生消费需求的时候，使用该品牌的服务 ；短周期的则是效果广告，希望客户看到该广告时，当下就会做出一些消费的行为，比如点击、购买、下载、安装等等。在中间传递层，从广告的价值分析，向合适的人传达消费的价值，也就是让有需求的客户看到广告，并且让他感受到当前广告推荐的商品服务可以更好地满足他的消费需求。

2. 如何传达消费的价值

广告系统是一个非常复杂的系统，包括很多非常复杂的投放策略，各种各样的策略算法。它包含基本的两大功能：匹配需求和展示价值。

其中，我们如何匹配需求，比如从用户行为，人群特征等方面，了解用户喜欢做什么，从而投放匹配的广告，满足怎么比较好的传达广告的价值。这一阶段总的来讲就是通过匹配需求，把用户想看的广告实时推送到用户面前，带来广告主精准投放，并且能实现很好的展现广告的价值。

那么如何实现“展示价值”呢，我认为可能由这两个方面组成：场景和效果。场景就是我们经常听到的营销上的词语“场景化营销”，比如今天上海台风，雨伞被风吹翻折，在这个场景下，如果出现一个可以抵抗 10 级台风的雨伞广告，用户会很有动力买这把伞。

从视频理解相关的 AI 范畴来讲，第一是如何找到这些适合场景化营销的场景，即广告点位，第二是摸索怎么去帮助广告展现出一个更好的效果。

场景：生成/推荐点位

1. 场景化示意

创可贴：关键是找到一个适合出广告的场景，并且这个广告跟场景带有一定的延续性、相关性。现在已经可以规模化地为影视剧生成广告点位。

例子：用户吃饭买单时，出现支付宝广告：买单就用支付宝。

前情提要：自动从上一集筛选并拼接多个小片段，能够对上一集内容做摘要，称为前情提要，在前情提要上可以贴广告。它是无中生有去制造出来的，可以根据广告主的要求自动生成不同的前情提要，作为广告点位。

例子：找到男女主高甜的场景，出美颜饮料广告，喝这种饮料，女人可以变得更美丽，让用户产生这样的关联。

video in：将广告 in video，它是一种后期植入，更灵活的广告投放方式。一方面当一个剧拍得很好，后期上线后会不断引入更多客户，我们需要这种在后期可以上线的植入广告。另一方面前期植入需要客户、商务和片方导演、演员等频繁沟通协调，相对而言，后期植入的灵活性较大，也不影响前期拍摄。

例子：运动员在乒乓球场馆，他们身后有一个饮料瓶。这是后期植入的一个广告。

2. 视频广告如何实现场景化

刚刚的例子都显示出在一个具体的场景化中，一个商品具有什么样的效果，使得客户有身临其境的感觉，同时配合不同的说辞，很明显的感觉到广告的效果。所以核心的工作就是寻找和生成场景化的点位，这些点位同时是具有商业价值的点位，通俗来讲也就是具有消费需求的点位。从视频分析，我们会给视频打上非常多的标签。有的具有商业价值，有的具有创作价值。商业价值的点位提供给商业部门，实现变现；创作价值的点位提供给内容创作者，进行二次创作。商业价值点位不一定是最精彩的点位，但它一定是能跟我们的消费场景有一定的关系。比如聚餐想喝饮料，地铁上听歌，海滩上防晒，亲吻时联想玫瑰花等等这些可能的消费场景。

3. 视频理解

DataFunTalk

为了找到消费场景，需要进行视频分析，也就是让机器去理解视频的内容。抽象来说，一个视频内容就是一个三元组，一个对象在一个什么场景，发生了什么事情。

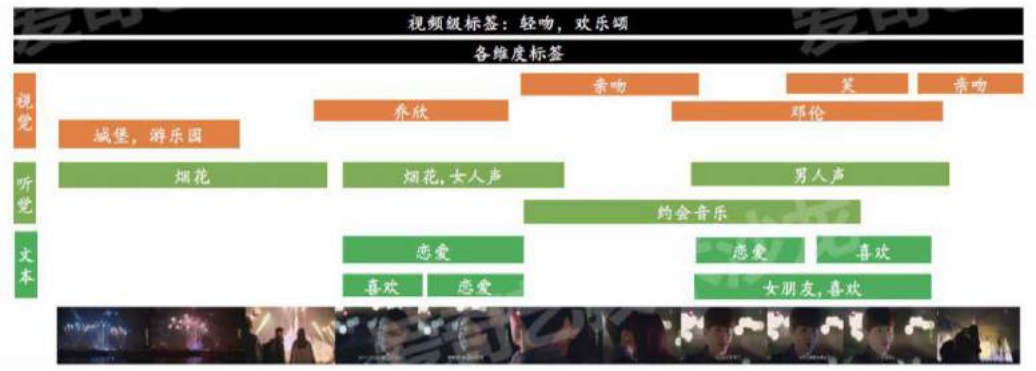
关于对象，这个对象可以是人，也可以是一个宠物，甚至是一个物体，对象的识别，我们可能去识别它的身份，穿着的装束，姿态等，加深我们对对象的理解；

关于事件，最基础的事件可能就是我们的行为，更抽象一点的事件，比如婚礼，战争等更高层的语义。在事件的识别中，既有视觉上直接的分类识别，也会利用到语音的信息，音频的信息，台词 OCR 识别产生的文本内容等；

场景的识别，即识别一种场景类型，比如这是海滩，会议室等，也有具体地标的识别，比如南锣鼓巷。包括它的调性，比如奢侈品，希望投放在一个高

大上的环境比如西餐厅，同时也包括背景音乐等等，通过这些大量的底层的算法，我们可以形成对视频的密集标签，这些标签从各个维度去理解视频。我们可以创造出很多具有商业价值的标签，通过类似创可贴这样的广告方式进行售卖，但是客户要求会越来越高，比如更关注男女主之间的互动，或者客户需要更抽象一些的概念时，底层的标签不能满足。这是因为底层标签是孤立的，未产生关系。

4. 视频图谱



怎么实现对视频更进一步的理解，以下面的例子为主，我们可以从视觉、听觉、文本打各式各样的标签，也可以从部分到整体可以打上各种标签；有了这些底层标签之后，我们需要能达到满足 2 个需求：

① 怎么去表达抽象的概念，比如浪漫；

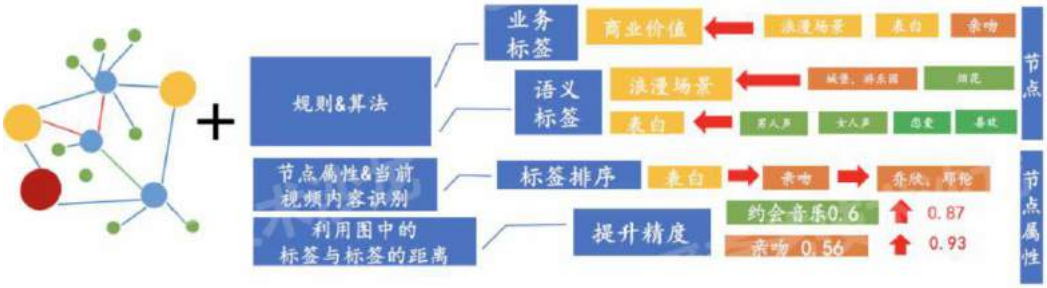
② 如何判断哪些是重要的标签，哪些是非重要的标签，以及标签的精度。

当我们生成这些独立的标签之后，我们应该利用标签之间的相关性，提高标签的精度。

当各个识别算法给出这样的结论：

从场景识别，识别出宫廷；从物体检测，识别出手机；从人物装扮中，识别出古装。

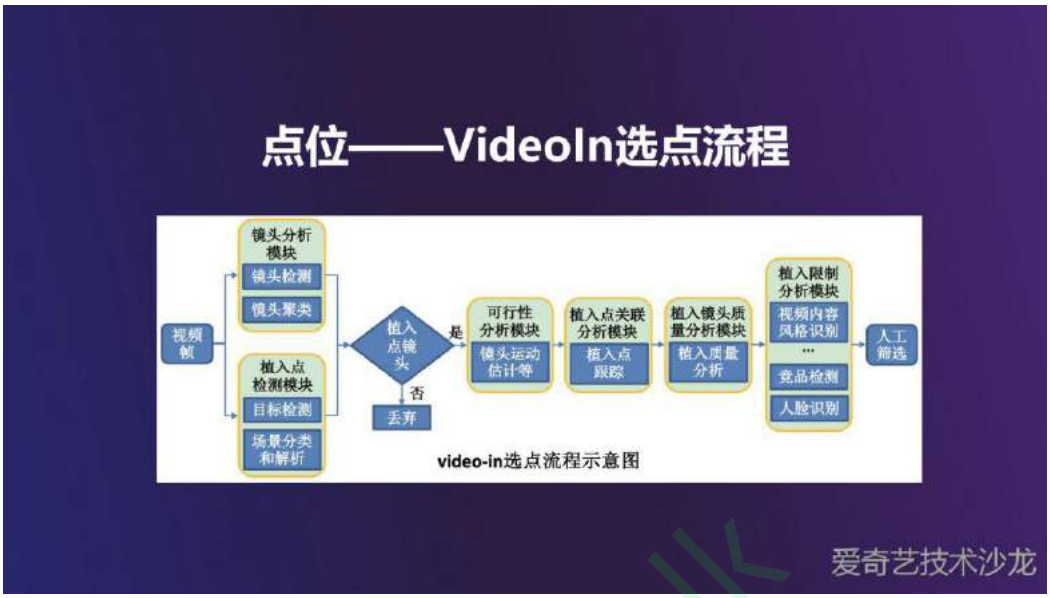
很明显物体检测出手机是有问题的，因为这些标签之间会很多的关系，比如同义词，上下级、相关性、互斥等，通过对标签之间的关系的分析，我们可以提高标签识别的精度，对标签排序，找到一些更多高层标签，特别是业务更需要的标签，比如具有商业价值和创作价值的标签。



我们还会有一些外部的知识，当我们每一个算法去分析视频的时候，我们的算法大部分是机器学习的方式，尤其是深度学习，我们搜集了大量的带标注的数据，算法能实现的是这些标注数据里面体现出的知识，如果这个算法的背后模型的网络结构设计的足够好，训练的方法足够好，它就越有可能去逼近这些标注的数据，虽然我们也会做一些非监督类的算法，但这些模型学到

的知识都是来自于训练的数据，这些训练数据都是一个领域范围内的数据，通过这些数据，我们会学习到哪些视频片段有什么样的行为，但是我们不知道发生这种行为的原因。比如我们识别出人物在跑步，流汗，我们很难得出人物想要喝水，想要补充体力等这些知识。这些知识不在我们训练数据里面，存在于外部数据，所以我们需要学习内、外部知识，对视频有更深入的理解，在这些深入理解的基础上，我们可以支持广告主更复杂的要求，它具有很多的节点以及节点间关系，从而我们称它为视频图谱。

5. VideoIn 选点流程



我们要做一个饮料的后期植入，我们需要什么样的点位？简单总结一下：

品牌方要求：空间足够大，不要被遮挡，镜头不要一直运动，不要运动模糊，具有场景化的需求，和明星同框等；

制片方要求：解决与艺人的同框问题，不能跟竞品广告同时出现；

技术方要求：技术层面是否可以植入等等。

6. 前情提要贴生产逻辑

对每一集剪辑出一个几十秒的片段集锦，它是一个兼顾内容价值和商业价值的一款产品。首先需要体现内容价值：

① **前情提要要精彩**，精彩是一个抽象的概念，可以拆为感官层面和情节层面；同时需要具有代表性，涵盖剧情关键的结点；

② **转场点识别**，主要分为场景/镜头和对话两方面，不能出现镜头/场景的突然切换，不能出现对话还未结束就被切割等；

③ **无效片段过滤**，比如一些回忆的镜头，一些主角缺失的镜头等；

④ **剪辑逻辑**，使得剪辑的视频非常丰富，也就是多样性，同时按照剧情发展的时间顺序。

7. 视频场景下的更多广告诉求

时效性：

新剧刚刚上映，是否可以马上产生创可贴或者前情提要的点位，选择 AI 来做，可以克服人去做的成本问题，可行性的问题（时效要求），不可控的因素等，可以交付一个更加具有鲁棒性的结果。

丰富性：

让人去做一件事情时，想要在有限时间内发现有价值的点位，会筛选出最大可能的点位，数量有限。但一个视频可自动产生的广告点位是非常多的，我

们在为广告主服务的时候，希望尽可能的满足客户需求，更关注广告点位的丰富性，这种丰富性可以提高广告投放的灵活性。

效果：辅助广告素材创作

创作方面的探索，怎么通过图片生成视频，怎么选封面图，怎么生成标题等方面的探索。

1. 场景化下的效果展示

DataFunTalk

可以在广告方面也做类似的探索，比如扫地机器人，如果出现老人，广告的术语为：有扫地机器人，避免老人弯腰；如果出现一个比较懒的宅男，广告的术语为：解放家务活等，广告内容会根据不同场景有不同的表现。

2. 视频广告的主要形式

DataFunTalk

视频广告的主要形式：

贴片类的：前贴、中贴和后贴；

浮层类的：创可贴、角标和前情提要贴

植入类的：前期植入、后期植入

3. 各类典型视频广告分析

我们会考虑在资源一定的条件下，评价各个广告形式实施的难度，规模化的程度。在实施难度方面：我们可以看到前贴对场景化、视频内容相关性要求最少；但植入对视频内容的要求非常多，需要大量的人工或机器做深入的分析；在规模化难度方面，植入很难规模化，但前贴是非常容易规模化的。

4. AI 辅助视频场景营销

在视频场景下的营销之后会是一个非常广泛的场景，之前我们看到的很多广告，它不一定有视频的场景，但是我们走在大街上看到广告牌其实也是一个视频化的场景。只是说我们生活中看到的广告没有剧情感，但是当我们看一个故事，看一个游戏的时候，它有剧情的，所以它能提供更多具有场景化的点位，所以视频平台，广告公司，内容创作机构都会一起来参与，那 AI 在里面，尤其是视频分析这一块 AI 的能力，主要会提供内容分析和素材的一个创作，在这样一个框架下，我们在视频场景做越来越多的探索。

OCPC 广告算法在凤凰新媒体的实践探索

分享：凤凰新媒体广告算法团队 编辑：陈道昌

导读：广告算法在优化点击率和竞价机制等提升平台收益的过程中，如果不能保证广告主转化的质和量就会影响到平台生态的稳定发展，因此在 CPC 的基础上，我们探索了 OCPC 模式，并在实际中成功应用。

本次分享的题目为 OCPC 广告算法在凤凰新媒体的实践探索，主要内容包括：

- 背景介绍
- CVR 预估
- 二价机制和智能出价
- OCPC 算法
- 技术架构

背景介绍

1. 凤羽简介

DataFunTalk

凤羽是凤凰卫视程序化广告变现优质品质曝光平台，汇聚了凤凰网、手机凤凰网、凤凰新闻客户端、凤凰视频客户端等多项业务，每天为凤凰网提供 20 亿次优质流量的曝光。

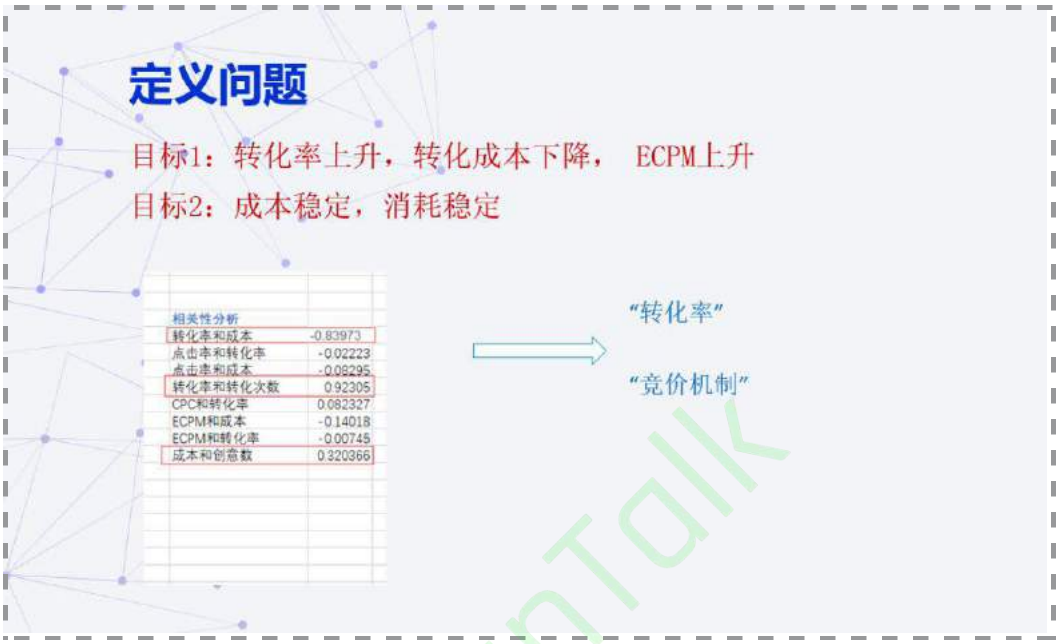
2. 什么是 OCPC

OCPC 是效果广告最近两年比较好的业务模式，广告主关心的还是成本 ROI，同时能跑量。

CPC 按点击付费，广告主基本都接受这种付费方式。本质上，广告主会根据凤羽的 CPC 消耗，核算他的转化成本。所以广告主为了优化这个成本，他可能调低 CPC，但是抢不到量；选择做人群、时间定向，基于经验论或者数据反馈；或者跑人群包等等。

相比传统的 CPC 业务效果广告，OCPC 本质上，是通过算法把优化工作替广告主做了。而且由于有更好的手段，所以效果更好。OCPC 还是按照点击付费 ,但是 OCPC 一般情况下 ,客户不再出点击价 ,出的是目标成本价 CPA。OCPC 一般分第一阶段和第二阶段。第一阶段用于数据的冷启动，第二阶段用于积累数据。

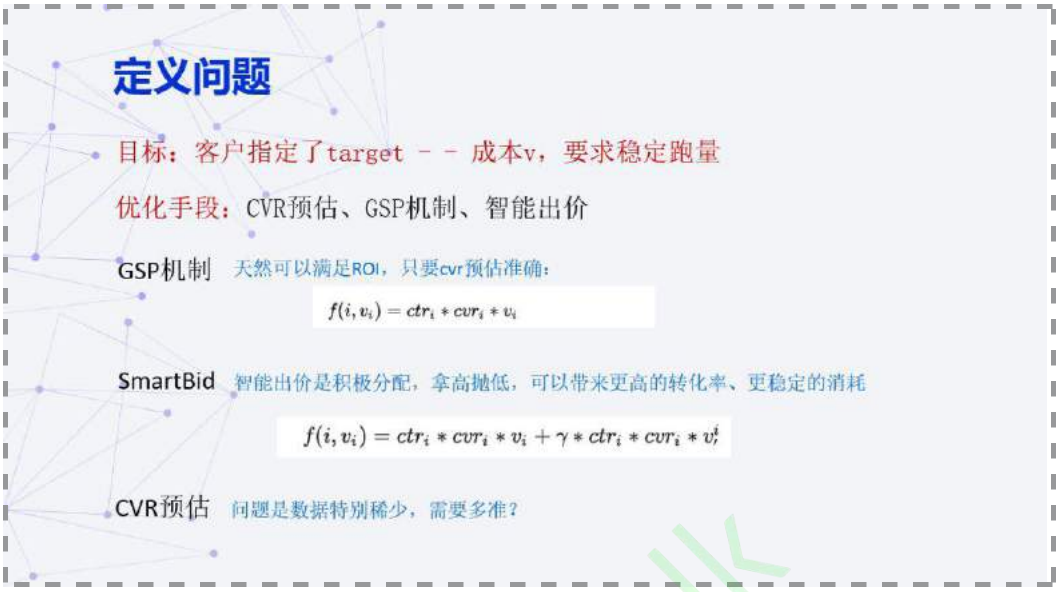
3. 定义问题



CPA 有两个目标：

目标 1：转化率上升，转化成本下降， ECPM 上升，从而提高客户转化率。

目标 2：成本稳定，消耗稳定，从竞价机制上提出优化点。

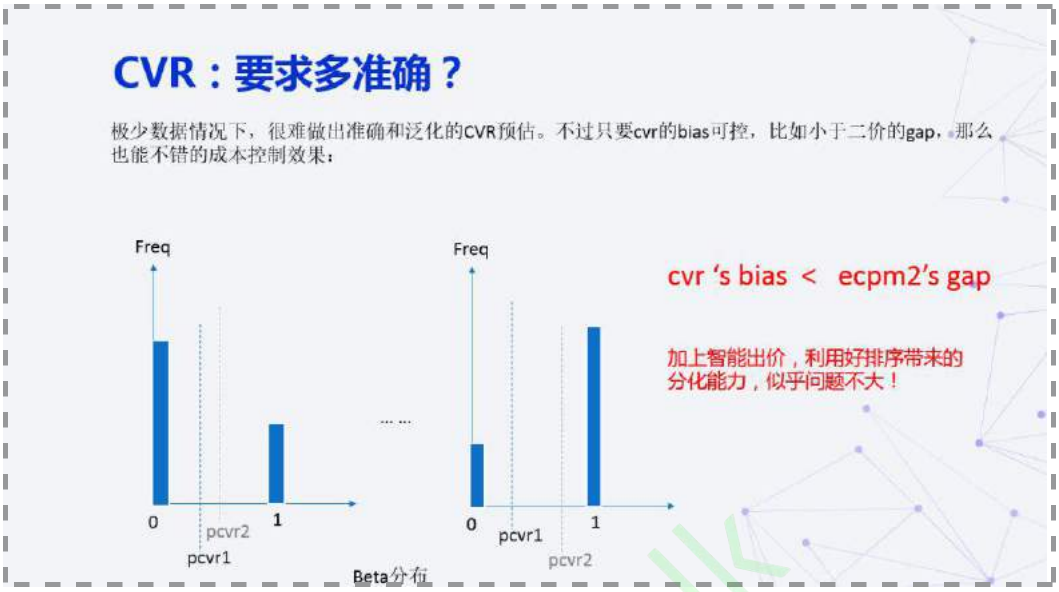


相对应的，我们的目标是在客户指定成本的情况下，每天相对稳定的跑量。

我们提出了三种优化手段：

- 二价机制，二价机制指的是出价格抢这次的流量，只需支付第二名的价格。
- 智能出价，相比二价机制的自然分配，智能出价是一种流量的更积极分配。会根据目标，进行智能的出价操作。
- CVR（ 转化率 ）预估，CVR 预估是整个阶段的基础，问题是 CVR 预估的样本非常稀少，需要做到多准确？

CVR 预估



由于在数据非常稀少的时候，很难做出准确和泛化的 CVR 预估，而 OCPC 可以完成。因为我们用到了二价机制，抢到的流量是第二名的价格，不过只要 CVR 的偏差可控，比如小于二价的 Gap，也能获得不错的成本控制。第二层面是 CVR 预估虽然有不准的成分，但我们可以通过智能出价来做经验控制，比如拿转化率高的部分，也可以把成本降下，从这个角度分析，CVR 预估不用特别准确。

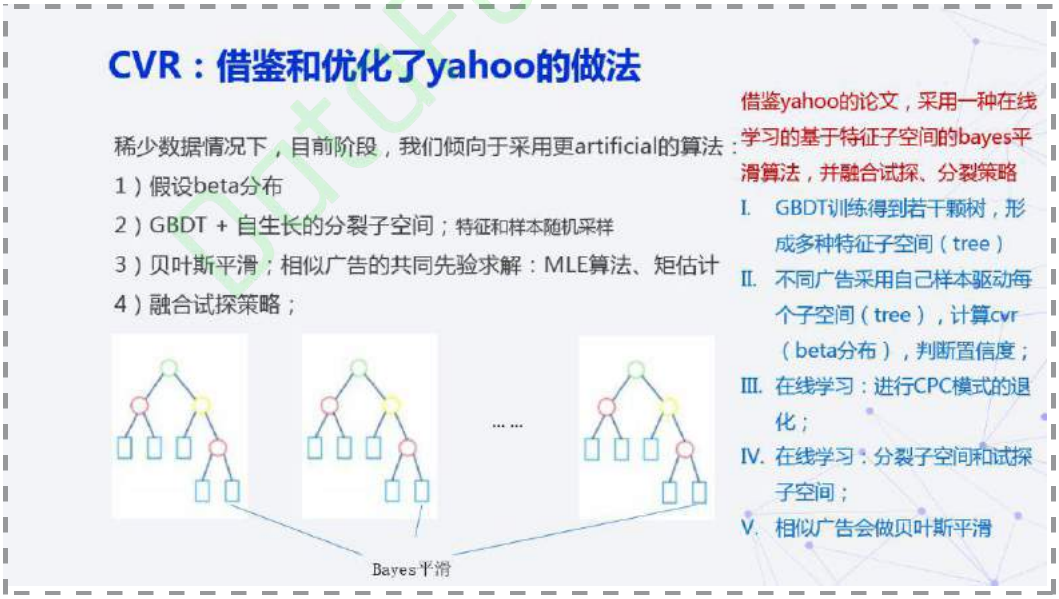
关于 CVR 数据稀疏的问题。新的客户在冷启动阶段的转化不到 100 个，使用不到 100 个的转化建立用户的转化率的预估，这个问题是比较困难的。其次，点击转化率与曝光转化率其实不一致，这是一个漏斗序列，先曝光再点击再转化。

如果用点击行为样本训练模型行预估点击转化率，在线上会出现数据不一致的情况，在数据样本比较小的时候，会有很大偏差。曝光转化率解决了样本不一致的问题，但是考虑到数据稀疏，建模后偏差过大，这种方法被弃用。

做过数据分析，发现点击率与转化率存在相关性，可以考虑将 CTR 与 CVR 一起联合训练联合建模，它们肯定有很多特征信息可以共用的。比如通过 weightedLR 做 loss 改造，在推荐等其它项目中有过不错的效果，后续可以一试。

最后，是构建一个模型还是多个模型？是分一个客户建模还是考虑同行业同目标一起建模？关键还是在于业务目标和数据情况：

OCPC 追求单个客户的完成率，要为单个客户做 ROI 达成和算法归因，同时各个客户之间的差异很大，这和全局最优化不一样。从这个角度出发，在考虑很多方法之后，决定为每个客户单独建模。所以会有很多模型，有很多工程上的版本控制，会考虑泛化与工程准确度的需求，我们会为相似的用户做一些先验的平滑，来提升准确度。后续 OCPC 的客户非常多的时候，可以考虑同一个行业，同一转化目标的类型一起建模，数据的稀少情况也能得到一定缓解。



我们目前的 CVR 预估的方法，借鉴 yahoo 的论文，采用一种在线学习的基于特征子空间的 bayes 平滑算法，并融合试探、分裂策略。步骤分为以下五步：

① GBDT 训练得到若干颗树，形成多种特征子空间（tree）；

② 不同广告采用自己样本驱动每个子空间（tree），计算 CVR（beta 分布），判断置信度；

③ 在线学习：进行 CPC 模式的退化；

④ 在线学习：分裂子空间和试探子空间；

⑤ 相似广告会做贝叶斯平滑，提升准确度提升泛化性能。

下面是关于算法的一些细节，简单的说用 GDBT 对样本和特征做随机采样，然后用随机采样出来的多对异构的多棵树，从里面提炼出各个特征子空间，然后对这些特征子空间，每一个广告、每一个客户，我会单独用自己样本探索这些特征子空间，在里面做置信度判断、CVR 预估、试探、分裂策略，来

保证 CVR 的准确性和泛化性能。整个算法在实现的过程中还是存在一些问题的，大概列出了以下几个问题：

① 为什么不采用 GBDT 的预估？

希望通过在线学习和置信度控制，来得到一个可以业务进化的有效模型和策略；但是，GBDT 仍然是一个有效的预估，可以考虑结合；

② GDBT 如何构建树？选多少颗树？

原生办法利用 LightGBM 特征的采样，第二种方法是人工做交叉，去选取特征，在实践中应该选多少棵树，也是看具体业务情况。我们的经验是一般用 16 颗树。

③ 如何多颗树的多个子空间给出的 CVR，组合出新的 CVR？有以下的几种做法：

-> 取平均值。

-> 参考每个子空间 CVR 的方差，给出加权平均（yahoo）。

-> 参考每个子空间 CVR 的方差，子空间颗粒度做加权平均，子颗粒太大，方差就大，偏差也会大。

-> 取 CVR 的最大和最小，最大值不推荐，训练模型是右偏的，本来是高估的，用最大值高估现象更严重，会高估用户成本，最小值想对安全，但是最小值可能跑不出量。

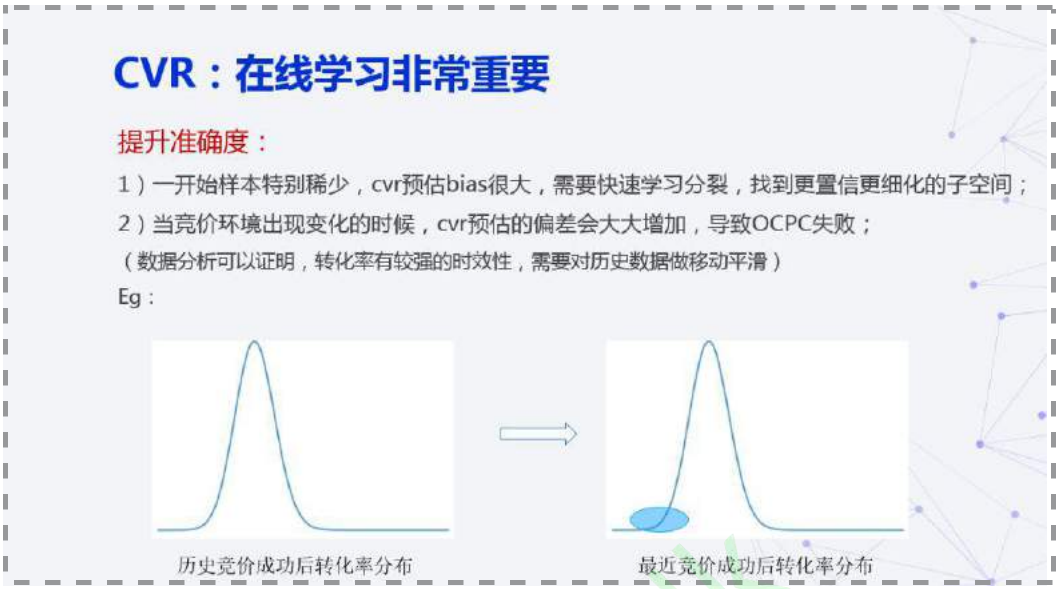
在后续的实践中，我们发现做 LR 集成还是一个更通用的模式。精度会高一些，同时可以将多个相似客户一起建模，提升一定的泛化能力，也减少了模型数量。

④ 多颗树中，有的子空间置信，有的不置信，如何选择试探、退化等策略？

-> 冷启动阶段数据太少，要退化到 CPC 模型，平滑了 OCPC 的第一阶段和第二阶段；

-> 相似广告的子空间数据要做贝叶斯平滑，做 MLE 求解；

-> 试探策略，计算置信上界，乐观估计，促使拿量；用置信上界。系统中有一个试探系数来控制。实践来看，当采用适当的试探系数时候，客户的成本和平台的 ECPM 能有一个双赢。所以这个试探系数可以作为一个策略学习部分后续优化。



可以通过在线学习，进行快速试探。在线学习的一个很大的优点是可以提升准确度：

一开始样本特别稀少，CVR 预估 bias 很大，需要快速学习分裂，找到更置信更细化的子空间。另外，当竞价环境出现变化的时候，CVR 预估的偏差会大大增加，导致 OCPC 失败，比如在节假日。

比如，在正常情况下，可以拿到左边完整的特征分布的流量。但是当竞价变得激烈之后，同样的模型和策略，我只能拿到转化率很低的那部分流量，实际 CVR 的偏差就会很大。这是很失败的一个案例，在线学习能大大提高 CVR 的准确度。

试探策略可以加快业务进化和 CVR 收敛。基于试探策略，将流量分配给不置信的子空间。它有两点好处，如下：

① 节省 OCPC 的试探弹药。将预算更多投放给可能有高转化率的子空间；能加快 CVR 预估的收敛速度，业务快速进化到第二阶段；

② 试探有一个刷新人群的效果；偏向于给新人群。

同时，实践中发现，试探能做为一个有效的策略，来影响最终的多目标达成，所以还是一个很好的智能调价 strategy。

CVR：试探策略

如果一个特征子空间不置信，我们倾向于给它足够多的试探；类似UCB策略，取威尔逊的置信上界：

$$\frac{1}{1+\frac{z^2}{n}}\left[\hat{p}+\frac{z^2}{2n}\pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z^2}{4n^2}}\right]$$

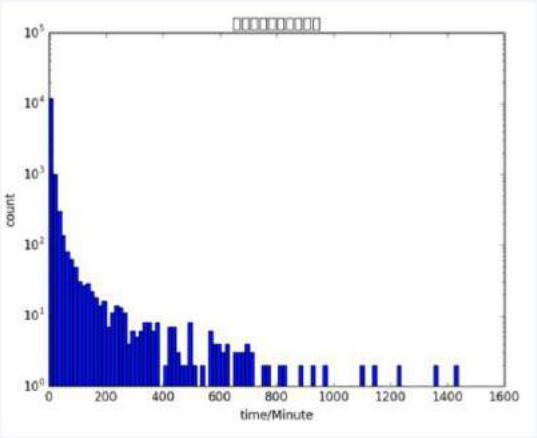
其中p是伯努利实验的成功率，n是样本数量或实验次数，z就是"z分数"或"标准分数"，是和指定的置信度相关的一个常数。

这样就可以让客户第一阶段能更快的收敛。

实际中，如果直接用置信上界去试探，由于太高的CVR值，可能出现跑量太快的异常情况。所以cvr预估值也不能太高，需要做一个上界的约束。

试探策略我们采用的是推荐常用的 UCB，为了避免样本过少，所有做了平滑，取威尔逊的置信上界。这样如果特征空间不置信，我们倾向于给它足够多的试探，就可以让客户第一阶段能更快的收敛。实际中，如果直接用置信上界去试探，由于太高的 CVR 值，可能出现跑量太快的异常情况。所以 CVR 预估值也不能太高，需要做一个上界的约束。

在线学习的问题：样本延迟没法做有效反馈



问题：不同目标、不同商品的转化周期不一样（有的能延迟几天）

做一下指数平滑，实现相对实时的反馈：

$$\alpha_{new} = \alpha_{\Delta} + \gamma * \alpha_{old}$$

$$\beta_{new} = \beta_{\Delta} + \gamma * \beta_{old}$$

后<1,t>天要再做下全天的修正。

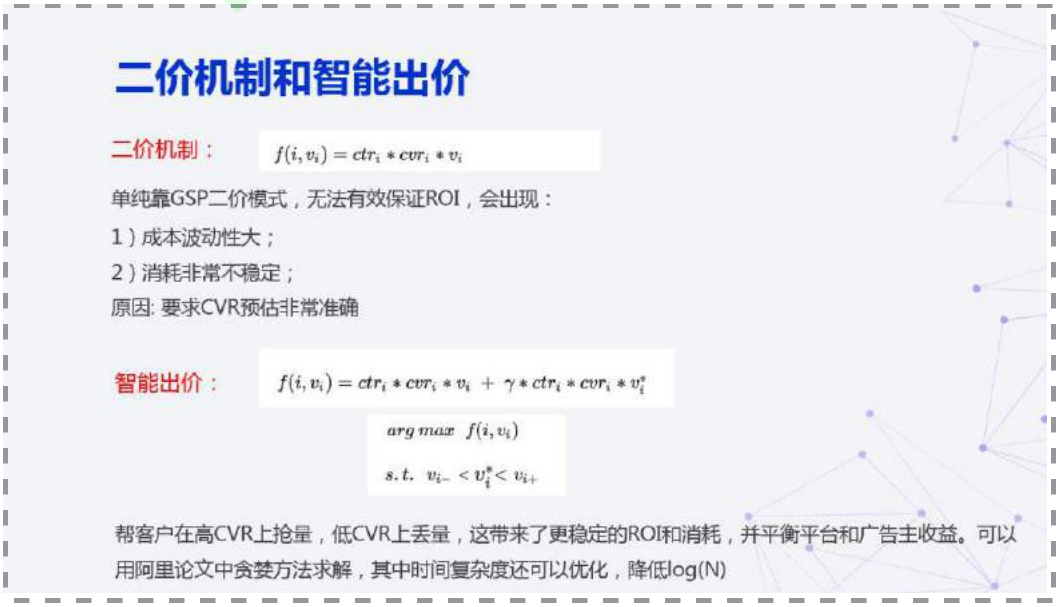
求解延迟概率分布，对实时样本做在线权重修正

归因的问题：比如3天前点击了广告并下载了APP，但是今天再点后才完成激活

在线学习有一个很明显的问题，转化其实是有很强的样本的延迟，不同的转化目标，不同商品，他的转化周期不一样。比如说 app 的下载转化是很快的，app 的激活可能要延迟好几天，我们在做在线学习的时候需要快速收敛，我们会以小时为单位对数据进行校验，然后开始做实时的返回学习。但是我们会对历史数据做指数平滑，同时我们会每一天、每两天、每三天为窗口的这段时间的数据重新，再重新做下全天的求证，样本延迟带来的偏差。

同时可以求解客户转化的概率的延迟分布，针对一个小时内的样本做权重上的求证。样本延迟还会带来一个问题，就是转化归因的问题，把转化归因到某一次点击上。比如三天某一个用户点击了广告并下载了 app，但是今天才实行了激活，这样的激活的转化是应该归因于三天前的点击，还是现在这一次的点击，这其实也需要自己取舍。

二价机制和智能出价



二价机制：在实际的过程中，OCPC 客户成本的波动性相当大，同时消耗非常不稳定，根本原因是 CVR 预估做不了太准确（如果可以做的准确就有更简单的解决方案了）。我们会采用智能出价，在二价基础上会更积极，能帮客户在高 CVR 上抢量，低 CVR 上丢量。智能出价带来了更稳定的成本 ROI 和客户消耗，并平衡平台和广告主收益。

智能出价：在多个客户情况下，智能出价变成一个组合优化的问题。可以参考阿里论文中贪婪方法求解，其中论文中的时间复杂度还可以优化，可以使时间复杂度降低 log(N) 倍。

智能出价是一种很有效的一种手段，但是核心还是 CVR 预估要准确，并需要找到一个合理的 CVR 期望。

关于 E(cvr) 的求解，阿里和新浪据说会采取用户的多个竞品对应极限的转化率。最好有多个竞品，去掉多个竞品中的最大值与最低值，然后取平均比较准确。但是在实际业务开展的过程中，尤其是 OCPC 没有足够竞品的时候，我们采用相对低效的手段，我们取客户本身已经近似的广告最近若干次的 CVR 预估，去除最高值与最低值的 10%并取平均，这是一个低效的策略。

我们还在探索基于强化学习的动态调价。但是这个需要有很大的流量来做支撑，所以不太好建模。

OCPC 算法及架构

总结下 OCPC 算法策略的特点：

① 业务进化，包含试探和分裂的策略，我们并没有明显的 OCPC 第一阶段；

② 在线学习，可以有一个在线的校准（Calibration）效果，可以一定程度上降低偏差；

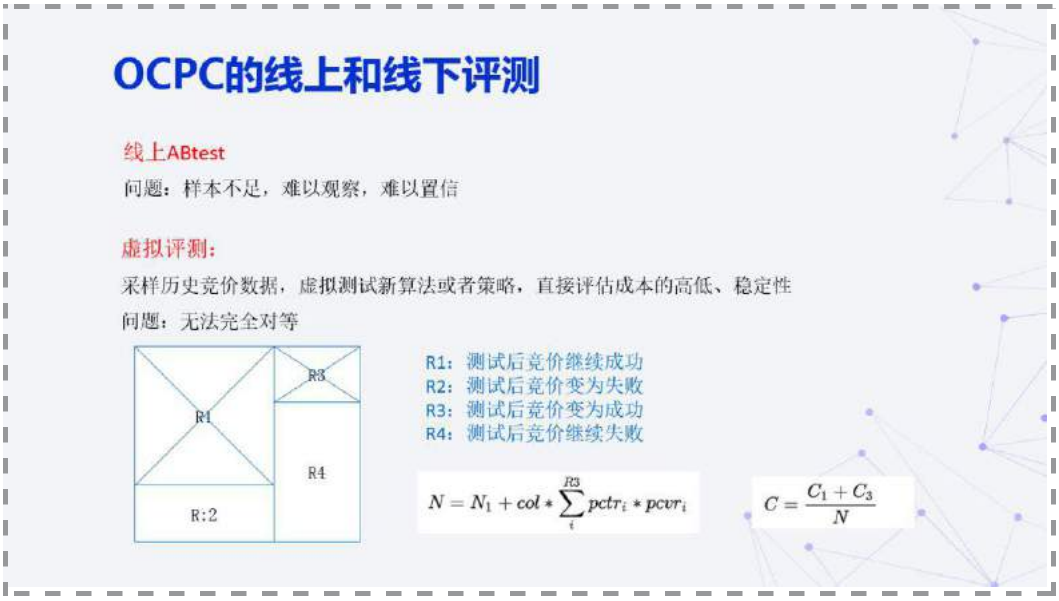
③ 在小样本的情况下，通过评估置信度，在准确和泛化中找到一个平衡，人工干预整个算法的进程；

④ 整个算法有试探、分裂等策略，在实践中有一个很快速、很明显的优化链路。

这是我们截取的客户的案例，经过短短的一周，用户的 CPC 的消耗和成本就能快速稳定下来，蓝色是 CPC 跑出来的曲线，黄色是 OCPC 跑出的曲线，

DataFunTalk

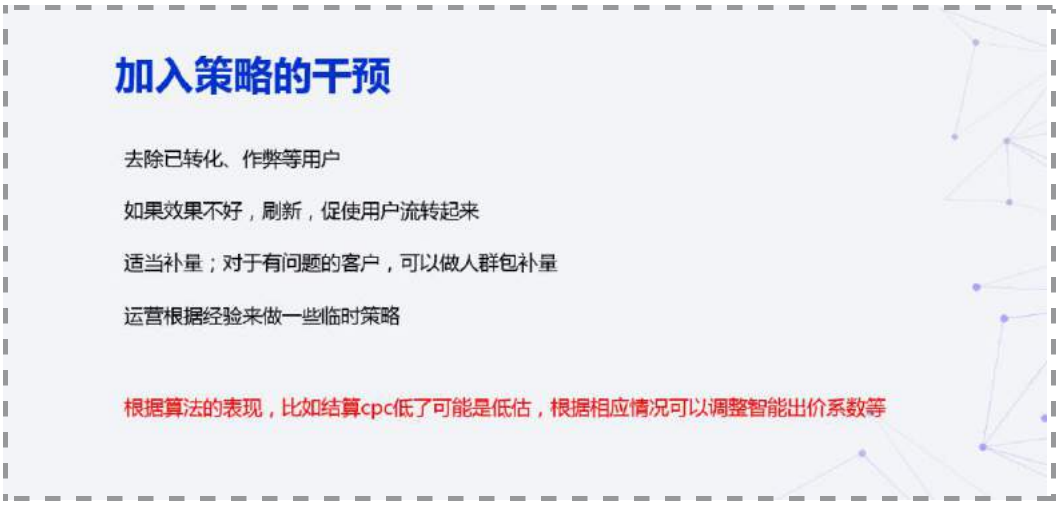
可以发现 OCPC 比 CPC 的成本要低 30%左右 ,同时 OCPC 的波动性也大大降低了，整个算法收敛的还是很快的 。



接下来谈谈 OCPC 其他的细节，除了对算法进行常规 AUC 评测、OE 评测和偏差评测之外，OCPC 还有很多的策略、方法和参数。如果都在线上开启 AB 测试，成本太高；另外，线上测试样本不足，用户的转化可能只有 10 几个转化，用 10 几个转化做 AB 测试，在统计上难以置信。

因此，我们采用虚拟测试的方法进行测试。我们会对用户一段时间内历史的竞价数据进行随机采样，把数据完全复原。然后，基于虚拟的竞价数据，针对我们新的方法和策略进行虚拟测试，观察采用新的算法，成本能否得到优化。但是会有一个问题，这种方法无法完全对等。比如客户的部分流量竞价没有成功，所以没有曝光，这部分流量的后续点击、转化概率其实都是未知的。

DataFunTalk



完成上述所有的手段后， 我们必须承认算法的不足，此时需要加入人工干预。干预手段包括：

① 对于已经转化的用户应该去掉，作弊的用户也去掉；

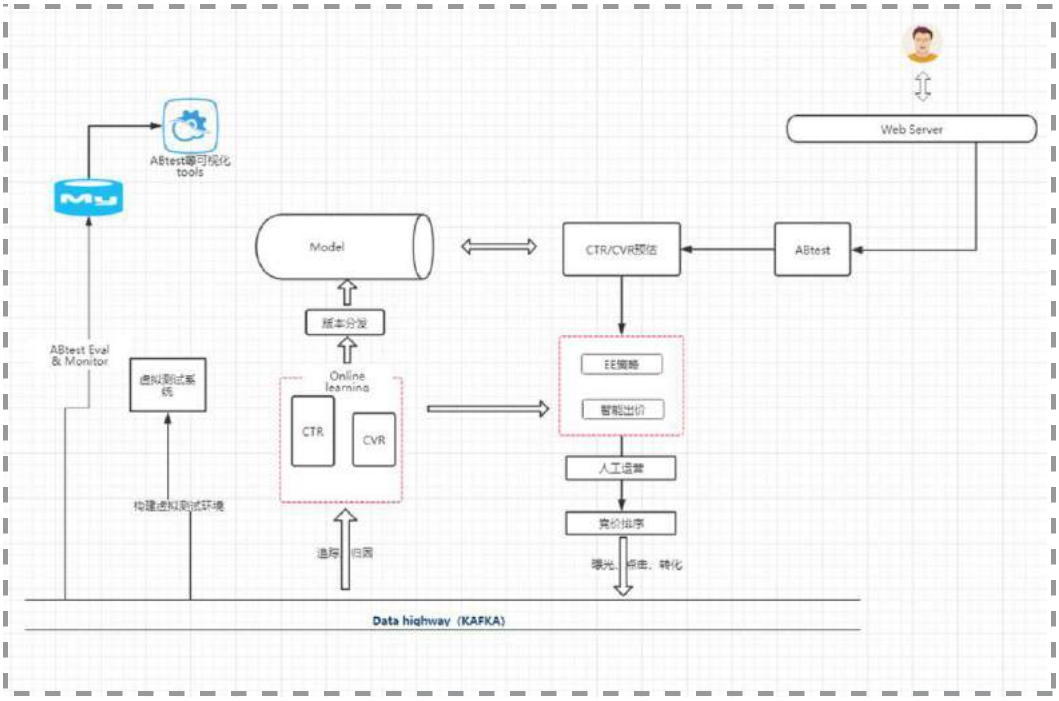
② 如果客户效果不好，可以采用刷新，促使用户流转起来；

③ 适当补量，对有问题的客户，可以通过人群补包补量，其他 APP 的流量引入；

④ 运营根据经验来做一些临时策略。

根据算法的表现，比如结算的 CPC 变低，可能是 CVR 预估偏低，这样的情况下可以做一定的修改。所以我们是用进化的算法+人工干预来保证每个客户的 RI。

DataFunTalk



最后是我们 OCPC 的技术架构落地。整个过程的关键是在线学习，融合了试探、分裂、净化的策略，在线追求好的工作链路，需要做些监控保证工作正常，可以构建虚拟侧环境，在线上引入 AB 测试，整个架构相对简单，预估部分与 CTR 工程相似，包括特征工程、模型。

未来工作

关于下一步的改善方案：

① 数据稀少，通过落地页的停留时长，来改善数据的稀疏性，停留时长与转化率有很重要的关系；

② 提升现有算法精度和泛化能力。考虑加入 GBDT 的预估值；采用特征和样本，提高 tree 的异构性；

DataFunTalk

③ 尝试采用强化学习的算法。对状态、动作空间都做一些限制，并引入确定的概率。这样在流量不大的场景下，是不是也可以求解。

阿里妈妈：品牌广告中的 NLP 算法实践

嘉宾：肖国锐　编辑：陈道昌

导读：本次分享的主题为阿里妈妈品牌广告中的 NLP 算法实践，主要内容包括：

1. 品牌广告业务模式与技术架构的简要介绍

2. NLP 算法在品牌搜索广告中的实践，以两个具体的算法问题展开：品牌意图识别和短文本相关性

品牌广告业务模式与技术架构

1. 阿里妈妈品牌广告业务概况

阿里妈妈品牌广告构建了丰富的产品矩阵，有强大的生态资源支撑，其中手淘是最大的资源方，阿里系的多个 APP 也是我们重要的资源提供方。品牌广告产品分为两个重要方向——品牌展示广告、品牌搜索广告。其中品牌展示包括超级风暴，整合了各个 APP 开屏页的广告；品牌特秀，主要指手淘内具有稀缺性的 banner 展示广告；以及优酷和 OTT 等视频广告。售卖模式上，品牌展示广告基本以定价保量模式售卖。品牌搜索广告包括两个重要产品，一是明星店铺，以 CPM 模式售卖，二是品牌专区，以 CPT 模式进行售卖。

2. 确定性诉求决定品牌广告业务模式与技术方案

大家可能比较了解效果广告的业务模式与技术架构，那品牌广告如何运作呢？它与效果广告差异非常大，核心差异来自于品牌广告要满足品牌广告主

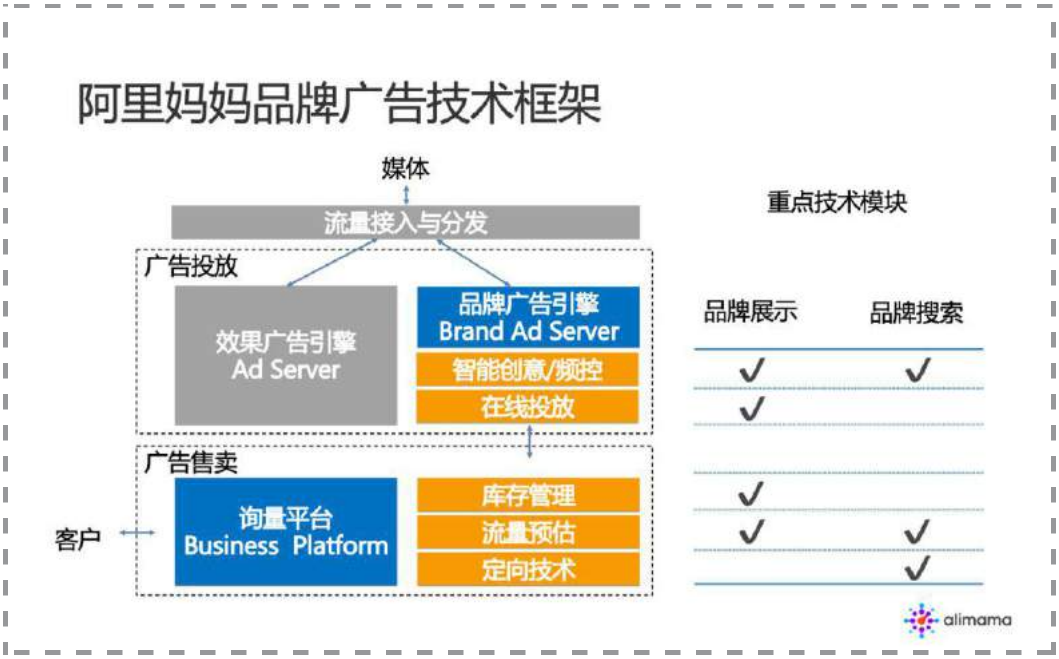
DataFunTalk

的最大诉求，即确定性。确定性体现在品牌广告全链路强烈的可控性——下单前，对明确的资源进行询量，事先锁定资源；下单后，签订合约，并以固定广告预算进行投放；最后，按合约拿到确定性的投放结果，比如触达多少UV、PV，N+ Reach、TA% 分布如何。

基于此，品牌广告业务和技术可以分为询量、下单、投放与评估四个阶段。询量阶段，和效果广告类似，需要实现定向体系。不同点是，需要对定向做流量和可用库存的预估。下单阶段，需要根据广告主确定要购买的定向、时段、预算，进行库存管理，比如库存的扣除或者释放，以及对订单进行报价。投放阶段，会面临多订单流量重叠的情况，要保证广告合约的确定性，技术上会涉及流量分配。在评估阶段，一般需要进行结案报告，如果是纯人工的方式效率会很低，因此我们需要有自动化的洞察、结案系统。最后，对投放的数据资产需要有沉淀的能力，实现持续化的消费者运营。

3. 阿里妈妈品牌广告技术架构

DataFunTalk



简要看一下阿里妈妈的品牌广告的技术框架。首先，从广告主侧视角看，广告主（客户）到达时，首先通过询量平台对要采购的流量、人群、时段和地域等各种定向进行询量询价，背后提供支撑的是库存管理、流量预估和定向技术。订单下单后，将构建广告索引，通过广告引擎实现在线投放，这里面涉及到的关键模块是在线投放、智能创意/频控。另外，从流量侧视角看，一个流量到来，会由统一的流量接入与分发层处理，根据流量类型分发到不同的引擎，效果广告或者品牌广告引擎。

这里要说的是，品牌广告内部的展示广告与搜索广告在重点技术模块上存在差异。品牌展示广告在整个链路上的重要模块是库存管理与流量预估，以及在线阶段的流量分配。这些是业内研究十分深入的方向，有很多技术资料，

这里不做展开。品牌搜索广告，除了以上常规问题，其特殊问题还包括定向技术、智能创意模块，今天会重点分享。

4. 品牌搜索广告产品形态

我们认识一下品牌搜索广告，左图是业内一种产品形态，右图是阿里品牌专区的一种形态。从技术视角看一下，品牌搜索广告需要解决什么样的问题。

首先，需要判别触发词是有品牌意图的 query。上面的两个例子比如“奔驰”、“BALLY 男包”，我们去理解它的品牌意图是非常容易的。对比来看，我们去判断“鲨鱼女包”的品牌意图可能就有点不确定了，甚至部分女生也不太了解鲨鱼是一个包包品牌。再比如“苹果 8 手机膜”，苹果是手机品牌，但是这样的流量不是品牌广告想要的，因为这个 query 的背后的意图不是买苹果品牌的商品，而是要买某品牌的苹果手机膜。

另外一个问题是广告匹配约束。品牌广告比效果广告有更强的广告相关性要求，它体现在两个层面。第一，流量与广告主需要品牌匹配，广告主可以将品专作为品牌专属的营销阵地。第二个层面是品牌效果层面的约束，广告主希望广告互动率要尽量的高，对于搜索广告需要解决 query 与广告创意内容的语义匹配。

NLP 算法在品牌搜索广告中的实践

1. 品牌搜索广告中的 NLP 任务

我们对上述问题进行提炼，包含两个技术模块，定向技术与智能创意技术。背后支撑三个重要业务能力：

第一是品牌流量的挖掘能力，扩充广告流量库存；

第二是品牌流量分配，解决广告主与流量的品牌匹配问题；

第三是智能创意匹配，解决的是广告内容与流量的匹配问题。

进一步，算法上对应两个 NLP 任务：

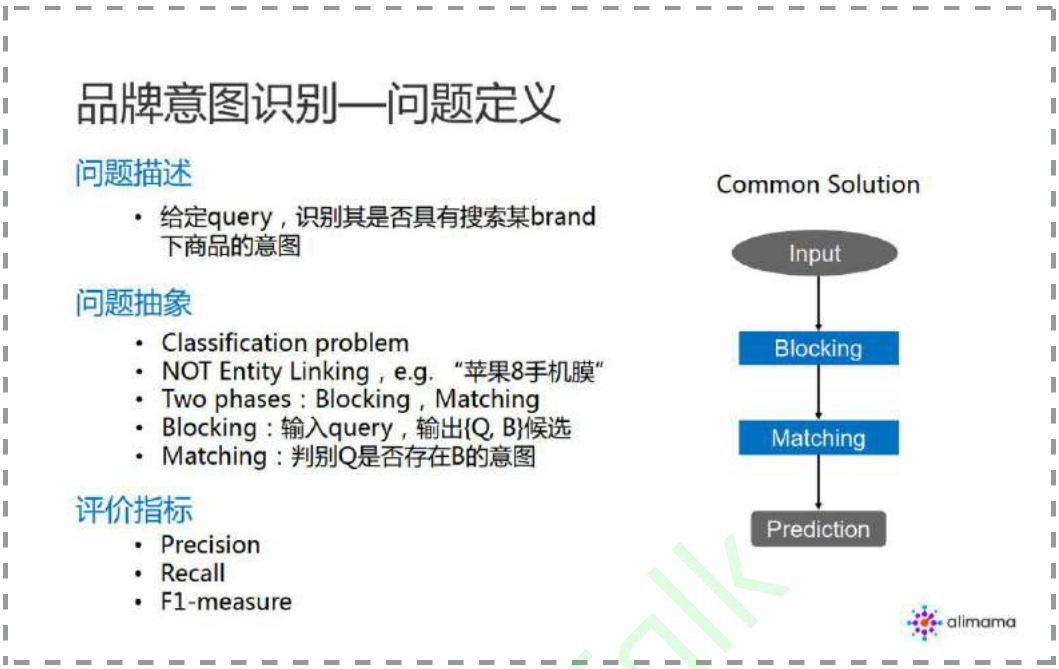
1. 品牌意图识别；

2. item 到 query 的短文本相关性。

下面重点介绍这两个任务的算法实践。

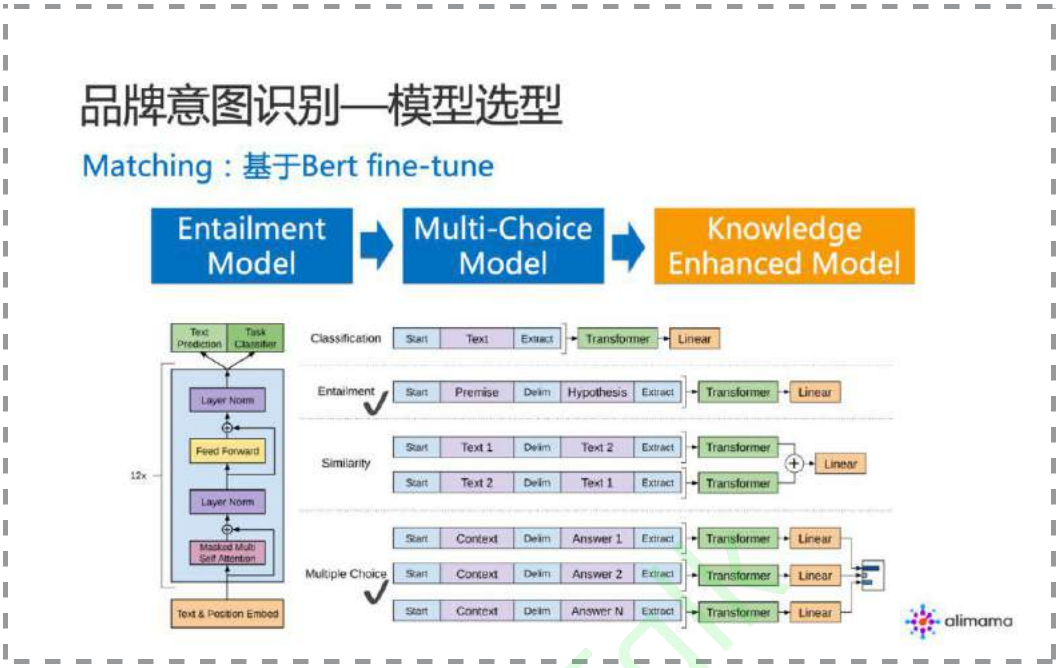
2. 品牌意图识别

① 问题定义



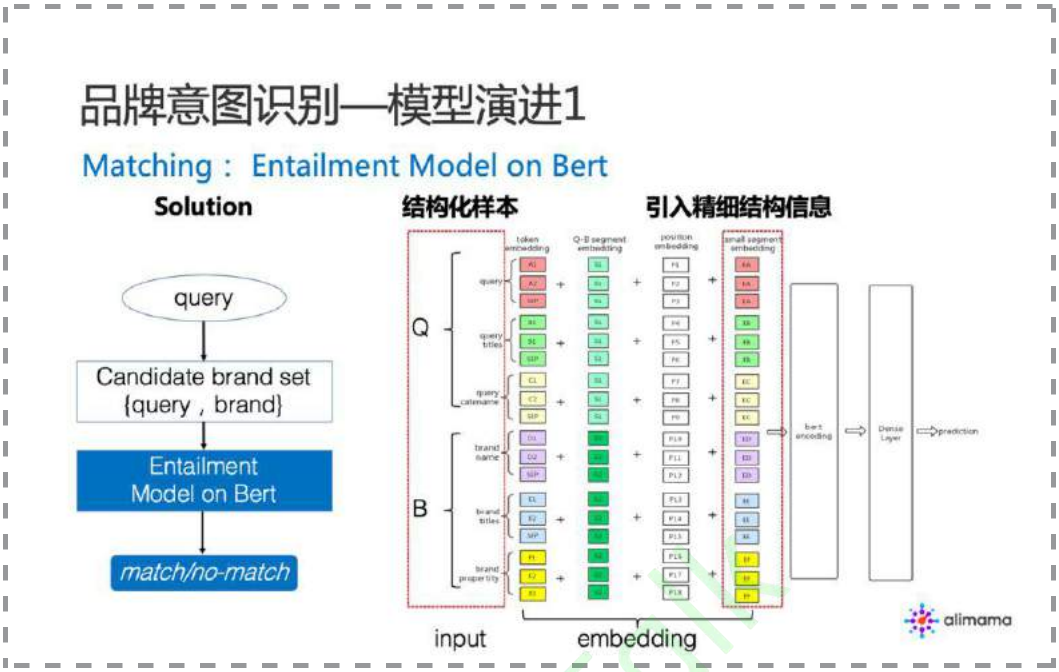
品牌意图识别问题描述为，给定一个 query 去识别其是否具有搜索某 brand 下商品的意图。首先，该问题看起来与 entity linking 相似，但实质不同。如刚刚已经看到了一个例子“苹果 8 手机膜”，entity linking 要识别苹果是一个品牌 mention。但这并不是该任务要解决的问题，而是要识别这个 query 是否有搜索“苹果”品牌的强意图。可以将其看作 classification 问题。我们将解决方案分两个步骤 :第一步是 blocking ,对给定 query Q 圈选候选品牌 brand B，形成一对一的候选 < Q , B > ；第二步是 matching，识别 < Q , B > 是否有相关意图。其中，blocking 可以采用一些启发式的规则，产生 Q 和 B 的候选集。我们将重点介绍 matching 的模型演进。

② 模型选型



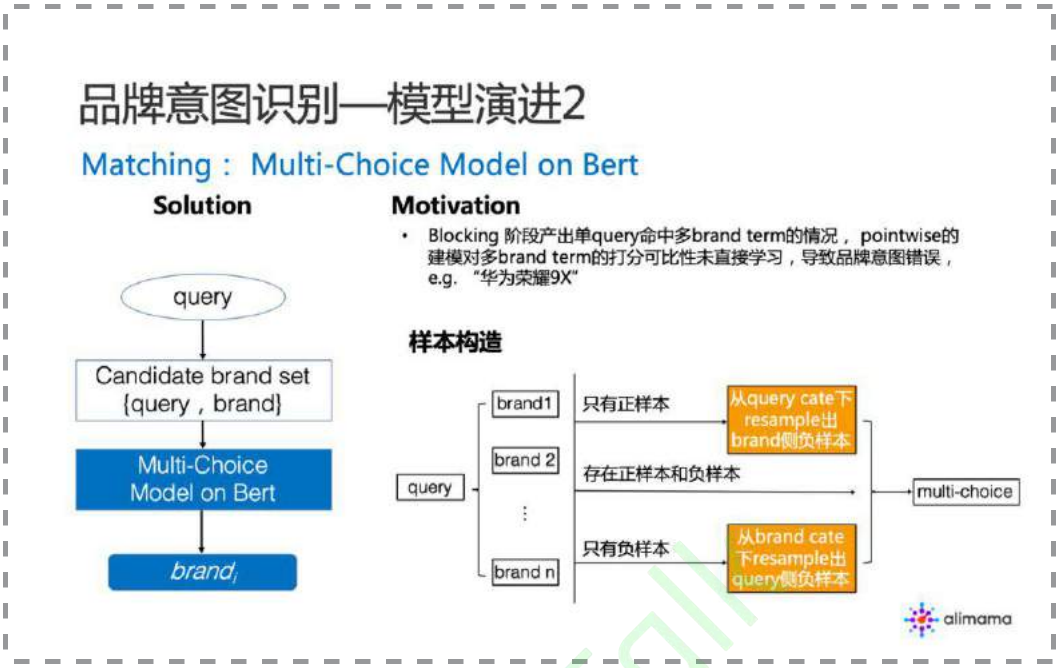
我们对品牌意图识别 matching 模型的演进分为三个阶段，已经完成的是前两个阶段，正在进行第三个阶段。前两个阶段主体思想是基于 Bert 进行 fine-tune。Bert 针对不同的 NLP 任务都有典型的建模方式，在该任务中，可以应用的有两个——Entailment task 与 Multi-Choice task。Entailment 任务中，Q 侧信息作为 Premise，B 侧信息则作为 Hypothesis。Multi-Choice 任务中，B 和 Q 分别作为 Context 和 Answer。

③ 模型演进



我们详细看一下 Entailment 模型实践，思路是对候选 < Q，B > 经过 Entailment Model 做一个预测，输出 match or no-match。其中，我们重点做了样本特征和模型结构上的一些设计。首先，样本特征上为了引入结构

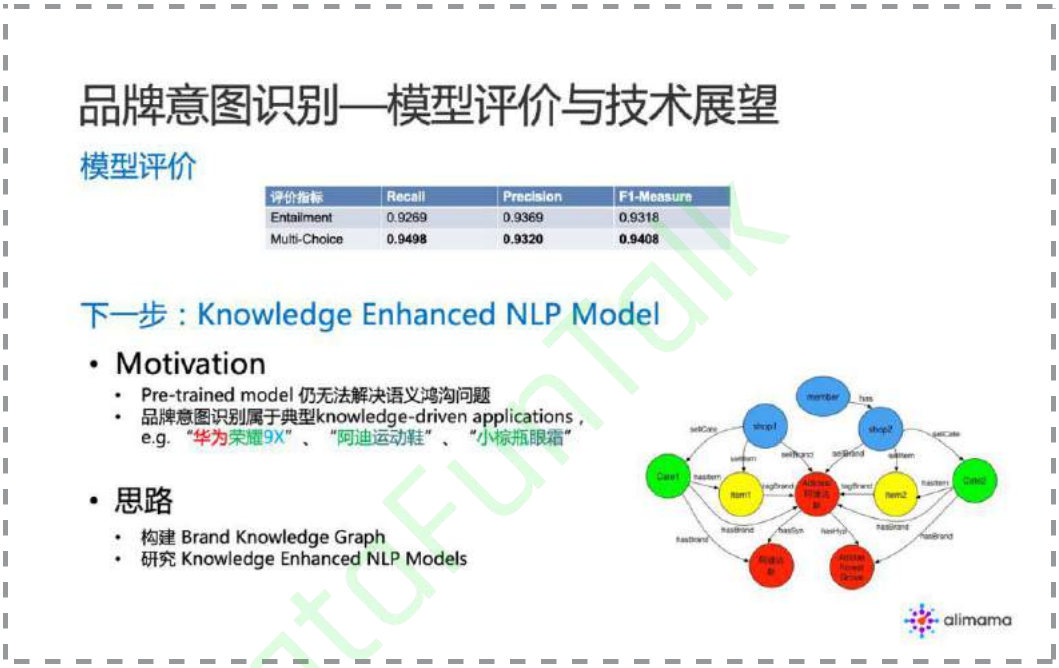
化的外部信息，我们将 Q 和 B 两个 segment 进行了三个 small segment 切分。在 Q 侧，small segment 分别是 query 本身、query click 的商品标题、query 的类目预测信息。B 侧也是一样的，分为候选 B 的标准品牌名称、品牌所属真实商品标题、品牌经营的类目信息。这是模型样本输入。对应地，我们在模型结构上针对 small segment 加入了一层 embedding，以更好地捕捉样本设计中的结构化信息。后续则正常输入到 Bert 进行 encoding，经过 n 层 dense layer 输出预测结果。



模型演进的第二阶段是 Multi-Choice 模型。Motivation 是 blocking 步骤可能会产出单 query 命中多个 brand term 的情况，Entailment 任务属于 pointwise 形式的建模，对多 brand term 的打分可比性未直接的做约束和学习，可能导致品牌意图识别错误。例如，“华为荣耀 9X”匹配到两个

候选品牌“华为”和“荣耀”，在具体业务场景中需要精确的识别其品牌意图为“荣耀”,而不能是华为。通过上一个部分介绍,我们可以了解到基于 Bert 进行 Multi-Choice 模型结构;在特征层面,Q 和 B 仍然采用一样的 small segment 特征。但在样本构造上,我们进行了一些工作。生成 <Q,B> 时,B 可能只有一个候选 brand term,这时需要提取多个候选的 <Q,B>,构成 Multi-Choice 的大样本。如果 Q 只有一个候选正样例 B,则在 query 的类目下 resample 出 N 个候选负样例 B;反过来,对于 Q 只有候选负样例 B 的情况,我们则对 B resample 一些正样例 Q。如上,从样本到模型构建了完整的 Multi-Choice 训练和预测流程。

④ 模型评价与技术展望



模型评价结果反映了 Multi-Choice 相对 Entailment 有一定效果提升。但是基于 Bert 预训练的语言模型无法解决语义鸿沟问题，尤其是当我们面临一些 knowledge-driven 的问题时，如果不引入 knowledge，模型效果很难有本质的提升。我们再看几个例子：

1. 刚刚已经提到了“华为荣耀 9X”；

2. “阿迪运动鞋”，人很容易感知阿迪是一个品牌，但是机器无法识别，这时最好建立“阿迪”与“阿迪达斯”标准品牌的同义关系，同时构建“阿迪达斯”可以经营“运动鞋”类目的知识；

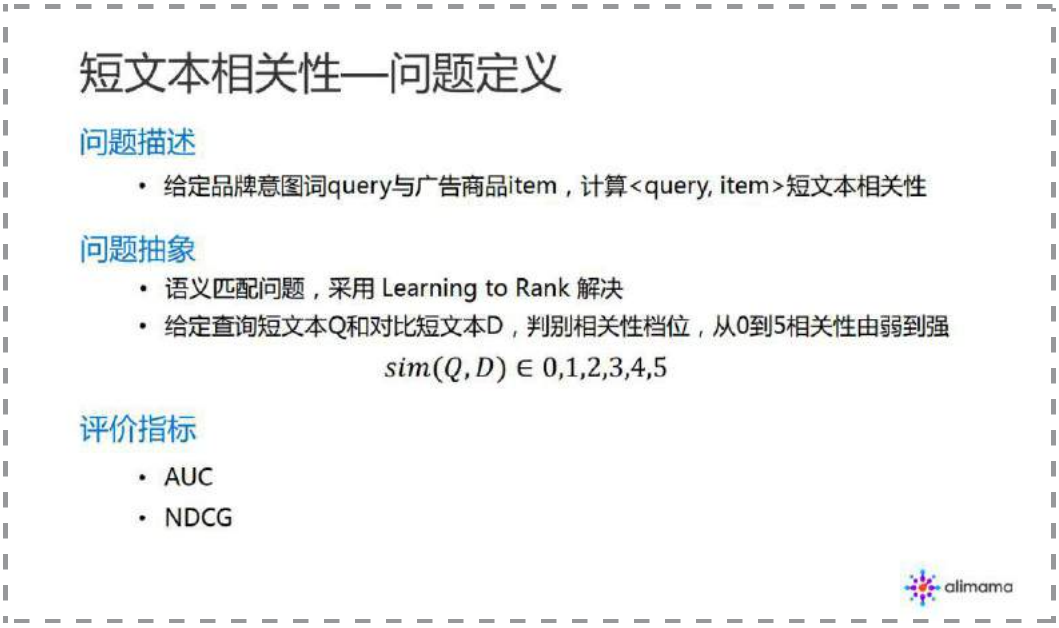
3. “小棕瓶眼霜”没有直接的品牌 mention，但要理解“小棕瓶”是“雅诗兰黛”强专属度的产品别称，同时“雅诗兰黛”在经营“眼霜”类目。

以上知识的需求，驱动我们构建营销领域的品牌知识图谱，然后基于 knowledge graph 进行 NLP 建模。目前业内关于 KG 与 NLP 任务结合的研究已经变得非常火热，值得大家关注。

3. 短文本相关性

① 问题定义

DataFunTalk



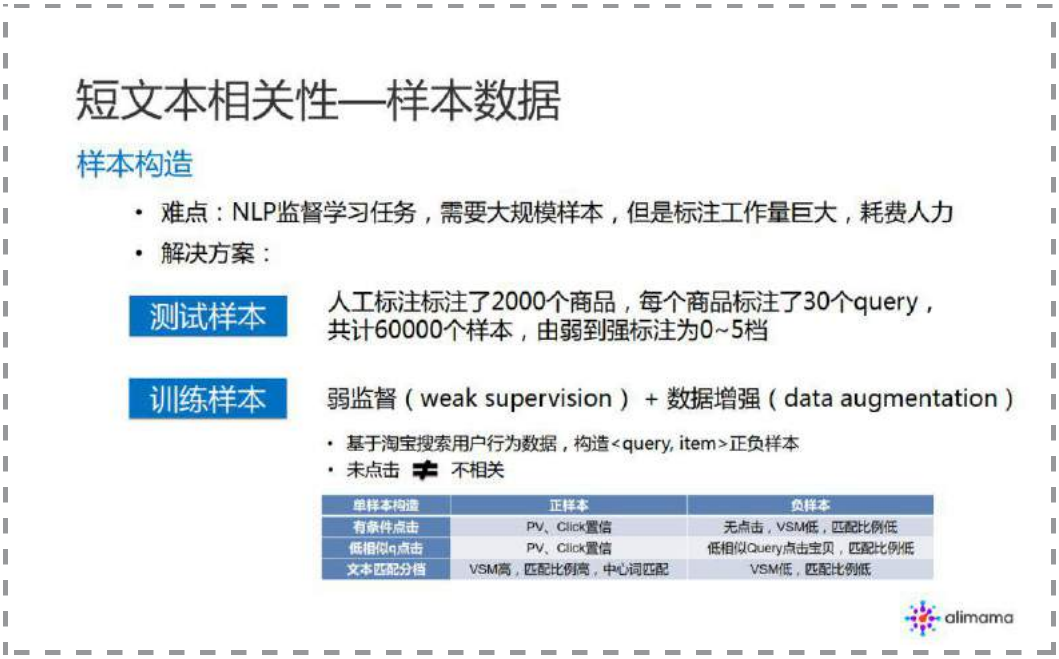
第二个算法任务是短文本相关性。问题描述是，给定一个品牌意图词 query 与广告商品 item，计算 < query，item > 短文本的相关性。可以把它理解为语义匹配的问题，采用 learning to rank 解决。进一步的形式化，给定查询的短文本 Q 和对应短文本 D，判别相关性档位，相关性从 0 到 5 相关性由弱到强。

$$\mathrm{Sim(Q,D)} \in \{0,1,2,3,4,5\}$$

对于这类问题，我们的评估指标有 AUC 和 NDCG 等。

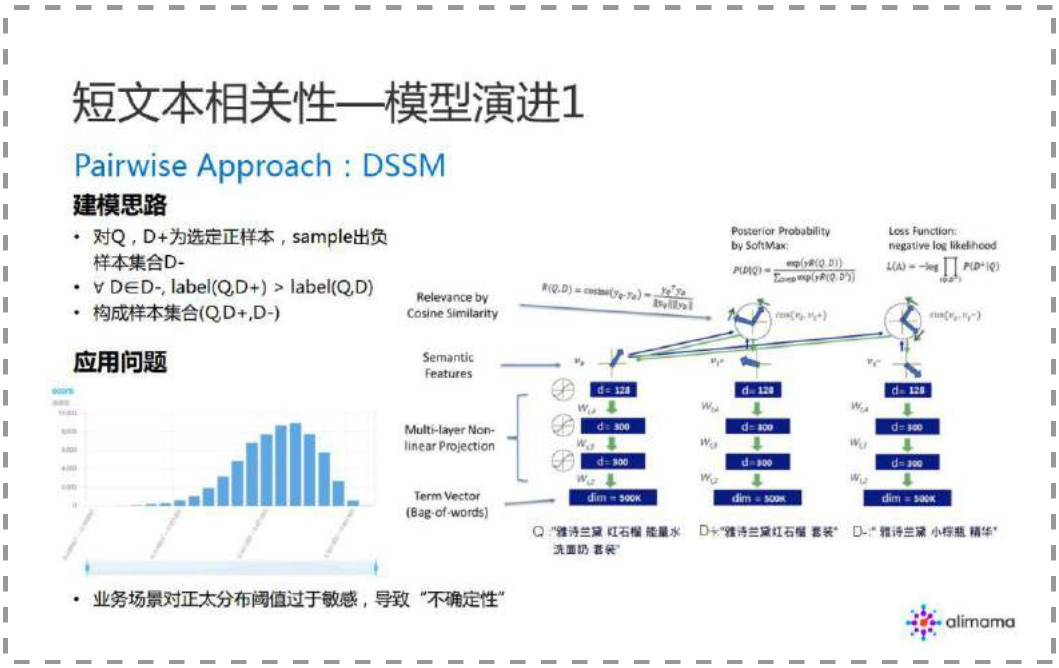
② 样本数据

DataFunTalk

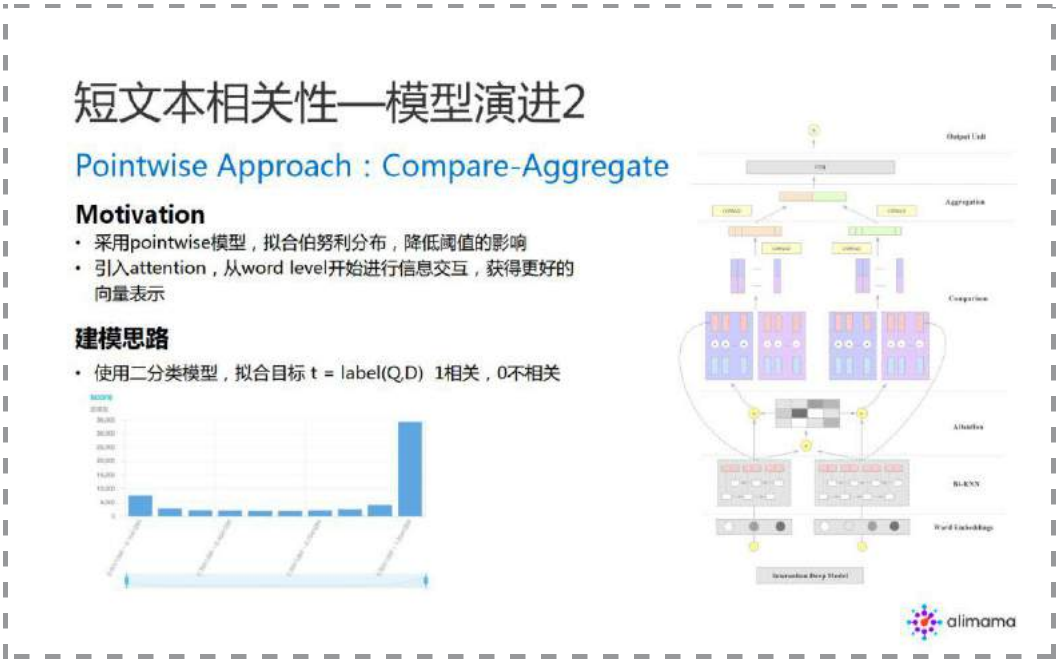


NLP 监督学习有一个很大的难点，即需要大规模有监督样本，但是其标注工作量巨大，将耗费大量人力和时间，一般公司难以承受。我们的解决思路是，先对测试样本进行人工标注，得到 6 万个样本，标注为 0-5 档层级。但是对于模型训练，采用 6 万规模的样本远远不够。我们的想法是把 learning to rank 的问题简化为一个二分类问题，判断相关还是不相关，而训练样本生成采用弱监督+数据增强的方法。弱监督的一种形式是，采用的数据标注不一定完全正确，但能一定程度上反应数据的相关性。淘宝用户的搜索点击行为就可以构建这样的弱监督样本。正样本非常明确，即 query 点击的商品 item 作为正样本。但未点击的 item 并不等价于不相关，我们需要人为的构建负样本。我们在负样本选取上做了大量工作，例如 query 无点击且 VSM 低的 item，再如与当前 query 低相似度的 query 点击的 item。另外，我们也做了大量的数据增强的工作，不在此展开。

③ 模型演进

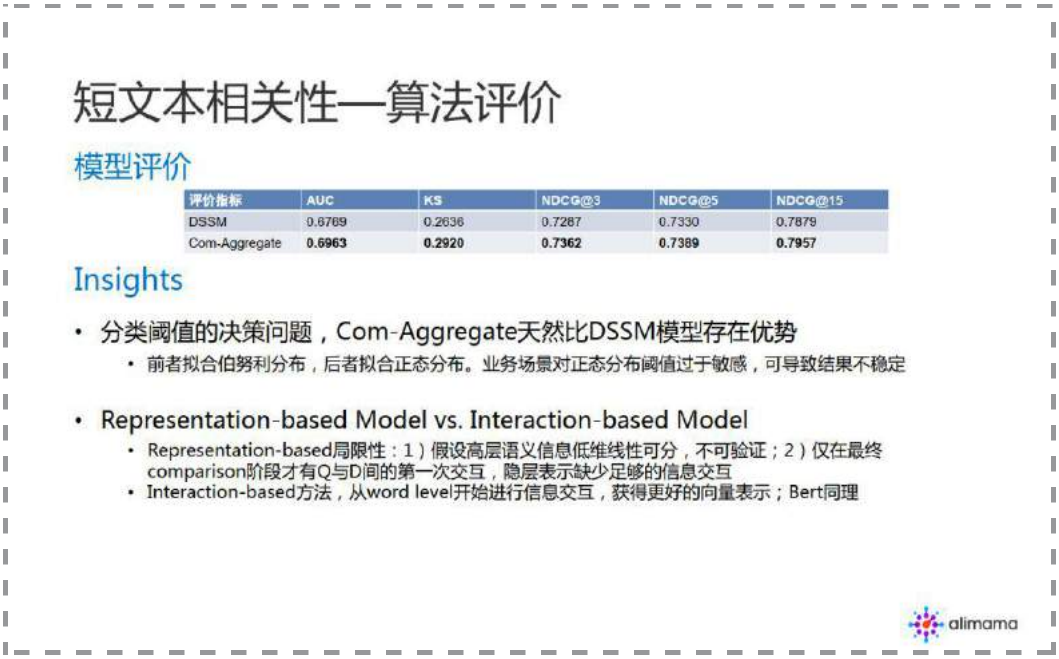


构建完样本，考虑如何建模。我们最初的建模思路是采用经典的 DSSM 模型：对于 query Q，其正负样本 item title 用 D 代表，可以构建样本集合 < Q，D+，D- >，作为 DSSM 的输入样本。我们发现对于 DSSM 模型结果，其算法评测指标比较理想。但在应用层面出现了问题，体现在候选 < Q，D > 的预测得分分布是偏正太分布。前面讲到，品牌广告业务需要满足广告主的确定性，广告主对流量规模、广告效果的稳定性高度关注。业务中需要决策相关性阈值进行 query 分配截断，正太分布下的阈值选择会导致不同天的 query 集合变动非常大，进而带来流量规模与广告效果的不确定性。



为了解决这个问题，我们做了模型升级。我们注意到 DSSM 拟合的是相似度偏序关系，即对任意 x+, x-: P(x+)>P(x-)；那么对于样本 x，从最大熵的角度看 distribution P(x) 可能更趋向正太分布。我们希望模型直接拟合伯努利分布，即 x+:p(x+)=1, x-:p(x-)=0，这将使得 P(x) 的分布呈现哑铃状。我们考虑从 DSSM 的 pairwise 模型转到 pointwise 的二分类模型，以此来降低阈值选择的影响。另外，希望在新模型上进一步优化效果（DSSM 模型基于高层语义表征计算相关性，缺少底层的信息交互 ）：新模型采用 Compare-Aggregate 架构，在 word-level 就引入了 attention 信息交互。结果表明，新模型使得样本预测得分呈现了哑铃状的理想状态，阈值截断影响大大降低。

④ 算法评价



从模型评价上看，Compare-Aggregate 在各个指标上全面超越了 DSSM。这里有两点 Insights：

1. 面对阈值决策问题时，为了大大降低阈值对结果稳定性的影响，模型选择上建议考虑拟合伯努利分布，避免拟合正太分布。

2. DSSM 这样的 representation-based model 存在两点局限性：

- 模型假设了高层语义表征能够线性可分，但该假设无法验证。
- 模型结构上只在最终的 comparison 阶段才有 Q 和 D 的第一次交互，隐层表示缺少足够的信息交互；而 Compare-Aggregation 模型本质是 interaction-based model，从 word level 就开始进行信息交互，获得了更好的向量表示。

DataFunTalk

我们认为基于 Bert 上做 fine-tuning 的部分任务也是同理。

总结

最后总结和分享一下我们在具体的业务场景进行 NLP 实践的感受：

- NLP 监督学习任务中样本数据获取是难点，可以考虑弱监督与数据增强。在模型方面，深度学习越来越工具化，可以针对实际问题在模型结构上做些思考和创新。
- 针对业务场景发现问题、定义问题，是应用算法工程师的核心竞争力；解决问题是应用算法工程师的基础门槛。
- 建议算法实践以履带式推进，这点与学术界有所不同，工业落地需要小步快跑，低成本试错，稳步迭代。

DataFunTalk

爱奇艺效果广告的个性化探索与实践

嘉宾：王晖　编辑：王浩

导读：随着大数据和人工智能的技术进步，数字营销行业也在不断变化。得益于雄厚技术实力，爱奇艺效果广告得到蓬勃发展，实现了海量触达、定向精准、效果显著的广告投放，为广告主提供跨平台、多终端的网络推广服务。本次我将分享爱奇艺效果广告个性化方面的思考与实践，聚焦广告排序算法，希望能为大家带来一些启发。

包括以下三部分：

- 爱奇艺效果广告算法 - 背景&架构
- 点击&转化率预估 - 核心问题
- 智能出价 - 博弈&共赢

爱奇艺效果广告算法-背景&架构

1. 广告资源位

DataFunTalk

如上图所示，爱奇艺主要的资源位可以用双引擎来概括，即信息流和框内：

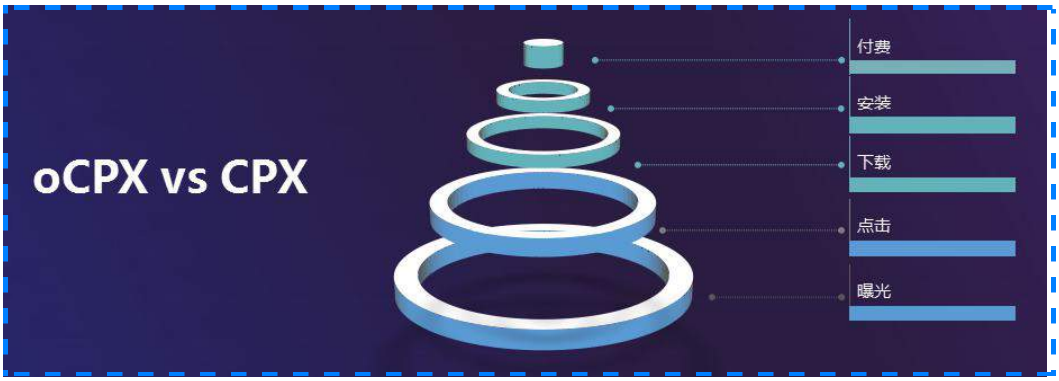
信息流：信息流广告位由准信息流和纯信息流共同构成的。其中，打开 App 显示的推荐页，由于是非原生信息，但是以信息流形式推送，所以我们称之为准信息流广告位。同时，爱奇艺平台提供了很多原生频道，比如热点频道，在这些频道下插入的广告称为纯信息流。

框内：对于框内资源，效果广告主要投放在框内中插的位置上，同时也会做一些前贴和后贴的框内广告投放。

其他：除上述两大引擎，还有猜你喜欢和视频关联位等资源位。

2. 计费方式

oCPX（optimized CPX）业务是 oCPC（optimized Cost Per Click）和 oCPM（optimized Cost Per Mille）两种计费方式的统称，是从 CPX 业务演进而来，这里的 CPX 代表的是以点击计费和真实观看计费。



CPX 时代广告主投放广告是以点击作为计费方式，但广告主的真实诉求是点击后的转化行为。例如上图，一个 App 的广告，用户点击后的下载、安装、付费，这些转化行为才是广告主真正关注的。广告主为了提升从点击到转化的概率，会做很多离线优化，例如定向 20-30 岁女性或定向喜欢《亲爱的热爱的》电视剧的用户。但这样做可能会产生很多问题，比如定向粒度过细，目标人群少，导致广告放量效果不佳。

为解决上述问题，oCPX 的业务诞生了，优化目标从点击变成了后续的转化行为，广告主只需在投放广告时设置转化目标以及这个目标下的出价，其他的事情由算法模型来解决。

3. 算法难点

oCPX 业务对算法挑战难题主要为上图三点：

- 转化样本非常高维稀疏，样本维度高且正样本量极少。
- 2oCPX 业务复杂，比如需要支持安装转化、付费转化、关注公众号转化等类型。
- 平台流量大，对计算规模时效要求高。

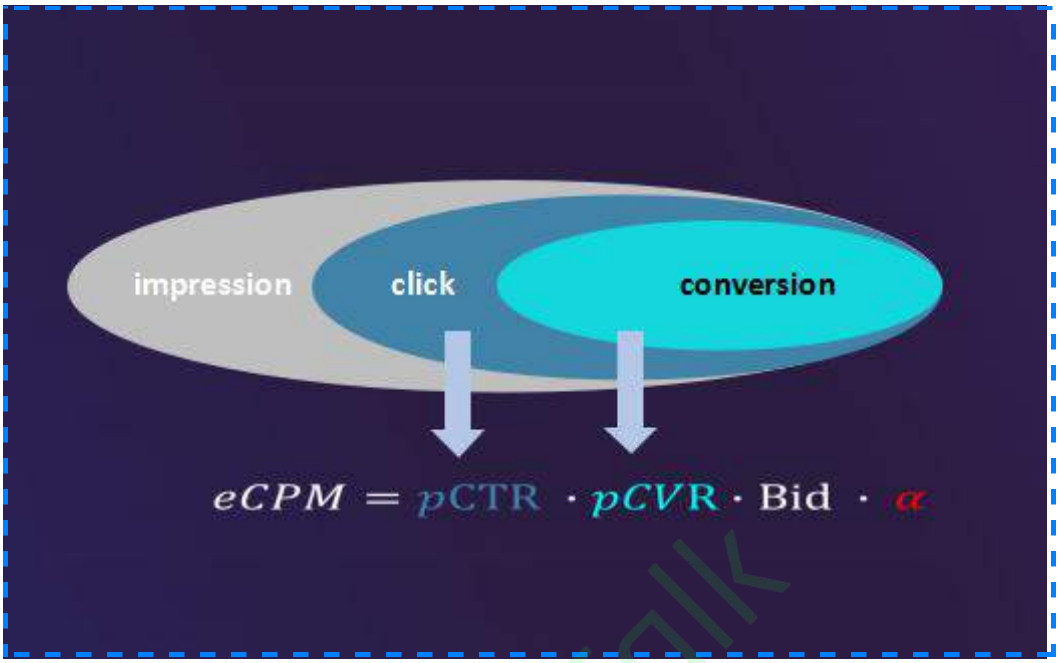
4. 个性化广告推荐流程

召回：对候选广告进行召回，比如人群定向等。

粗排：使用轻量级但保证精度的模型对广告做初选，避免精排阶段服务压力过大。粗排阶段还有冷启动和随机探索等功能模块，主要用于解决 E&E 问题，避免马太效应（ 投放好广告越来越好，差的越来越差 ）

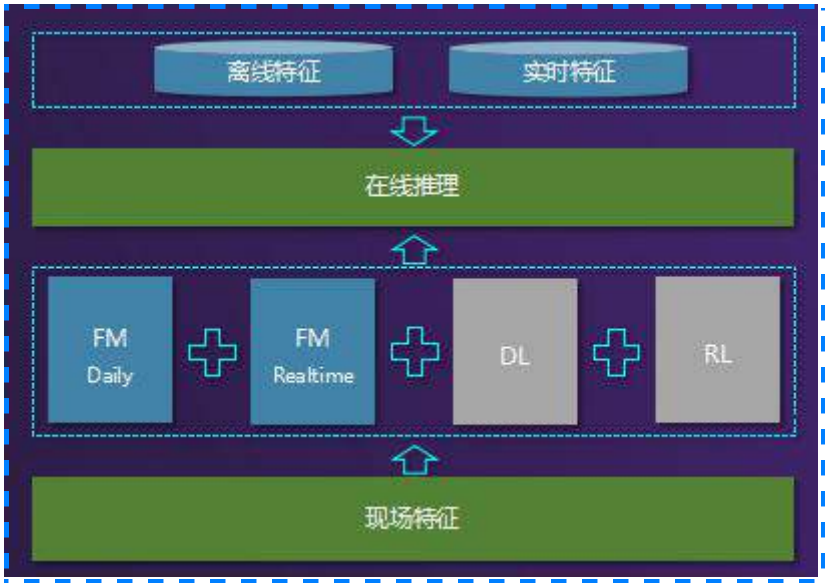
精排：使用高精度模型对点击率、转化率和智能出价进行预估。另外精排阶段还支持预算平滑功能。还有预算平摊到全时段等。

5. 排序逻辑



粗、精排的排序逻辑依照 eCPM（effective Cost Per Mille），也就是千次展示平台的预期收益来排序，其计算公式如上图所示，eCPM=点击率×转化率×出价×智能出价因子。

6. 线上训练推理流程



如上图自下而上：

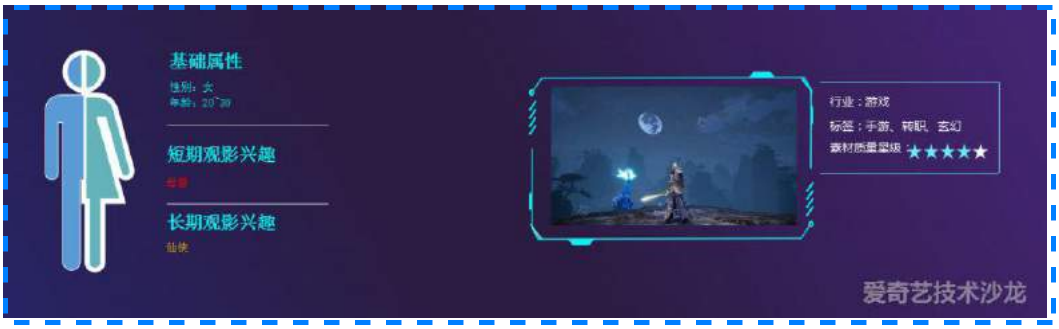
1. 现场特征，相较离线特征，现场特征能真实还原线上环境。

2. 第二步进入模型训练，会有 FM 分天模型、FM 在线学习、深度学习和强化学习等。

3. 完成模型离线训练后推送至线上，结合离线、实时特征进行在线推理。

点击&转化率预估-核心问题

1. 特征工程



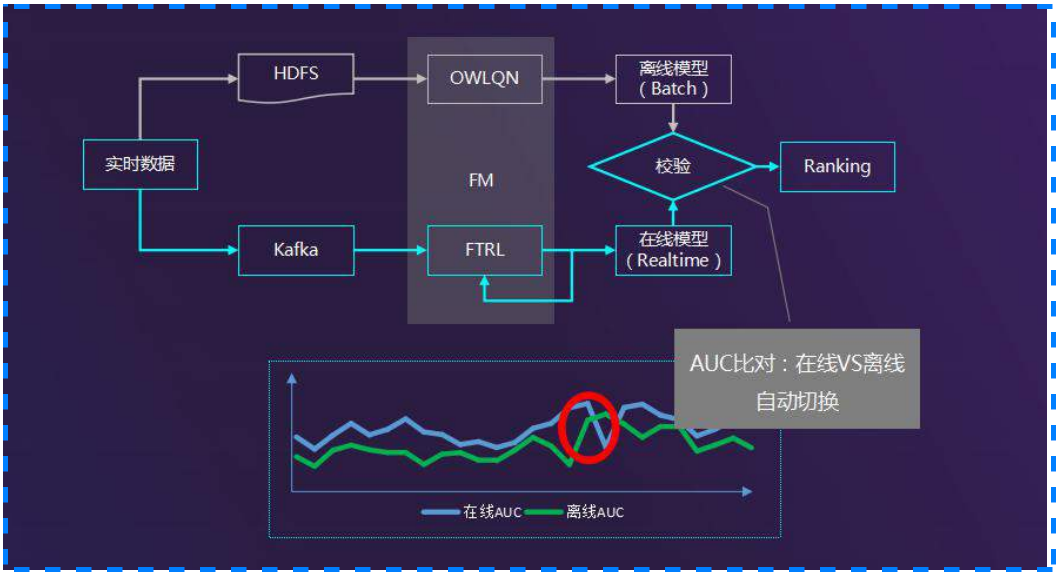
从周期来划分特征可分为实时、短期和长期。

实时特征主要考虑**场景**特征，即上下文特征，比如用户在看到广告之前看了哪些视频；**时间**信息也是非常重要的实时特征，比如周末和工作日分时段的点击转化率表现就有很大差异；此外还有**反馈**特征，包括正反馈及负反馈，主要针对用户看过哪些行业的广告及后续转化行为。

短期特征包含用户观影兴趣、搜索行为、社交行为和商业兴趣等。其中短期观影兴趣可能会发生变化，比如上图，如果有位女性用户怀孕了，她的观影兴趣可能会从长期的仙侠类视频短期切换到母婴类视频。

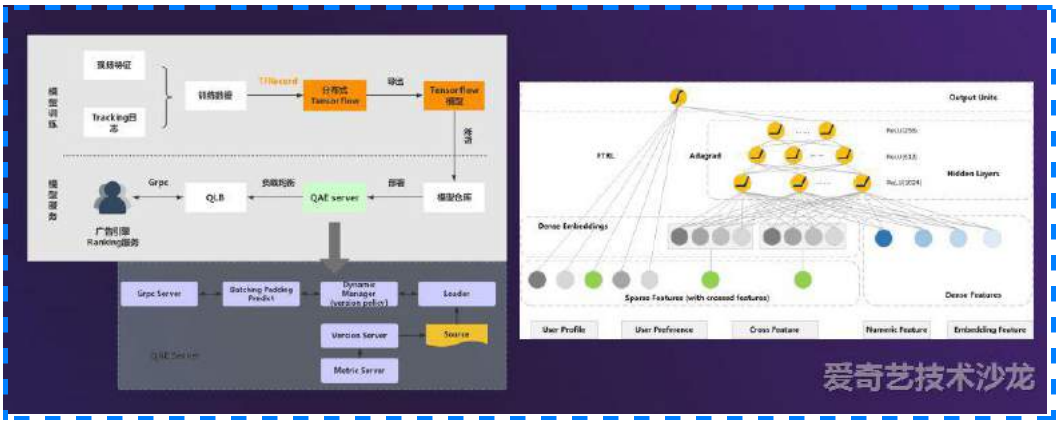
长期特征有人口基础属性(性别年龄等)、观影兴趣、商业兴趣、广告基础属性(所处行业等)和素材质量等。这里的观影兴趣，依然举上图例子，这位女性用户可能喜欢参与仙侠类的讨论，喜欢出现在仙侠剧中的偶像等等，所以针对这位用户打得长期观影兴趣的标签是仙侠。

2. 在线学习



天下武功唯快不破，时效性对模型来说是非常重要的。当前线上使用的基于 FTRL 的 FM 模型，实时消费线上 Kafka 流的数据，分钟级进行模型更新。由于线上模型是分钟级更新，线上要有一套监控体系，一旦线上指标出现异动，比如上图举例的 AUC 低于离线分天训练的备用的模型，会进行线上模型与备用离线模型自动切换。

3. 深度学习



为了提高模型准确率和泛化能力，上线了深度学习，当前采用的是 Wide&Deep 的模型，并进行了离线模型调优和线上性能优化，以满足广告场景下对高 QPS 以及高稳定性需求。

4. 高维稀疏

在转化率预估场景下，会遇到数据**高维稀疏问题**。举个例子，在做偏差校准的时候，会出现很多分桶里面一个正样本都没有，进而导致这些分桶最终的预估转化率结果为零。为解决这个问题，采用的方法是动态分桶大小，根据当前广告下历史正样本数量，动态决定分桶宽度以及分布。

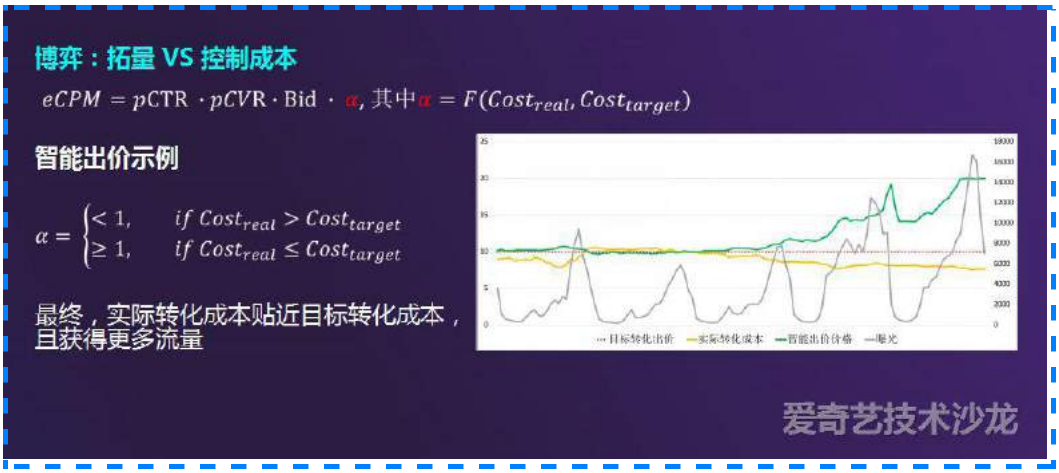
同时，转化率预估场景有**滞后性问题**，例如以 APP 作为推广目标的广告主设置转化目标为付费，但从用户点击广告到最终付费可能会经历很长的时间。传统的做法是设立时间窗口对这条样本进行跟踪，窗口内完成转化目标即打上正样本标签，反之则当作负样本看待，但很明显这么做有很多问题。为解决这个问题采用了一种替代方案，当点击产生时，我们将样本当作负样本来看，当转化行为完成后，这个样本就视为正样本，当然这里会进行适当的权重调整。

此外，还有**转化率阶跃的问题**，即转化数据集中到来，正样本突然增多，转化率预估可能从万分之几直接到百分之几。这样就会造成转化数据到来前广告放量不足，到来后爆量问题。为了解决这个问题引入非目标的转化数据作

为正样本使用，比如转化目标是付费，我们可以引入下载、激活等也作为加权降权后的正样本来消费。

智能出价-博弈&共赢

1. 成本控制



智能出价的基本功能是进行成本控制，保证广告的实际成本在目标成本附近。并在此基础上尽可能多拓量。所以这里存在成本和放量的博弈。所使用的方法是在 eCPM 上乘一个智能出价因子，这个因子是实际成本与目标成本的函数输出，比如上图实际成本大于目标成本时，因子小于 1，用更廉价的流量来拉低成本，反之大于 1，提高竞争能力获取更多流量。

2. 流量优选

上面介绍了基础版本的智能出价方案，但这样会产生一些问题，比如放量过程中的订单，可能以高价格获得了低质的流量而导致效果不佳，反之可能错过高质量流量。解决方法是做流量粒度下的智能出价 ，即考虑成本的同时也考虑流量质量，就是上图 g 函数，考虑成本参数的同时也加入流量质量参数。

360 展示广告召回系统的演进

嘉宾：王华呈　编辑：杨辉之

导读：随着展示广告业务数据量的日益增长，360 展示广告召回系统也随之也进行不断升级改进。本次介绍主要从召回系统演进的角度详细阐述工程实践中的算法应用、技术难点以及解决方案。主要分成三块：第一个是 360 展示广告的大致介绍，第二个是整体架构介绍，第三个就是今天的主题召回模块的演进脉络。

1. 展示广告介绍

1.1 展示广告业务介绍

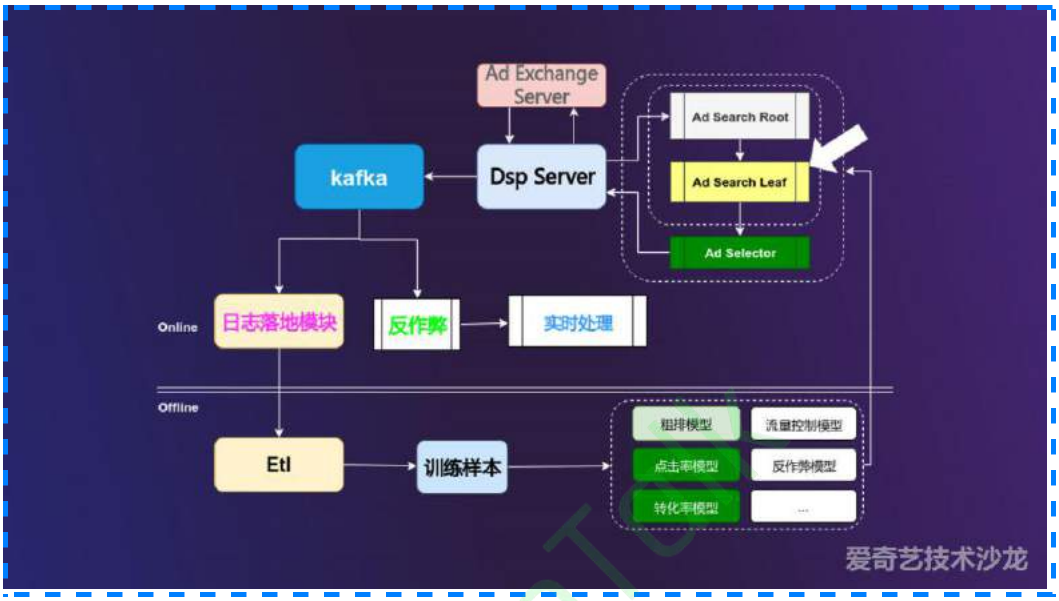
展示广告主要是在 RTB 程序化购买框架下运行的，首先有媒体方，比如新浪、搜狐等市面上的各大媒体，我们都已经接入。媒体在一次曝光产生之前会把

这次曝光发送给 Ad Exchange（ADX）模块进行拍卖，360 的 ADX 平台叫 360 Max。ADX 平台将该曝光发送给多个竞价平台 DSP，DSP 来决定是否对该次曝光进行竞价，360 的 DSP 平台叫 360 点睛 DSP。广告主们会针对自己的需求设置广告投放策略，DSP 平台会从广告主设置的广告投放库中根据该次曝光的特征匹配出合适的创意（广告）候选集返回给 ADX。一般要求 DSP 的响应时间为 100ms 左右，除去网络传输耗时，留给 DSP 模型的时间只有几十毫秒，其中间各个模块的时间就更少。ADX 从接收到的所有候选集中选择出价最高的广告进行曝光。

1.2 常见展示广告

目前在 360 场景下主要有这几种广告类型，第一种类型是广告侧边栏，包含具体投放广告的详情页。第二种类型是整体的开屏广告，主要是品牌广告。第三种类型是在信息流场景下，将广告嵌入新闻上下文中且与文章内容形式保持一致。

2. 展示广告整体架构介绍



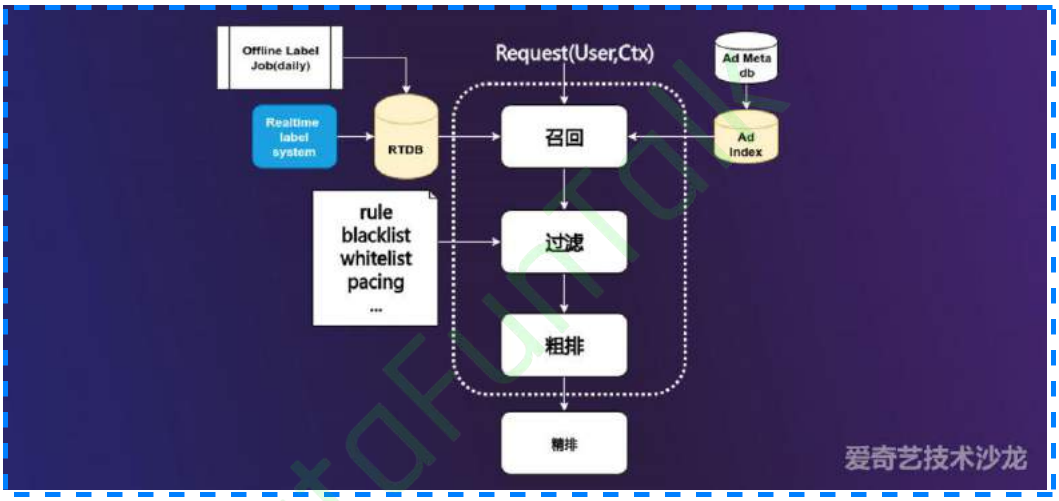
整体架构流程：

流量从 ADX 发送给 DSP 端，DSP 拿到流量之后交给检索召回模块，我们叫 Ad Search 模块，Ad Search 模块分为两个部分，一个是 Ad Search Root 模块，对流量进行识别，判断其为哪种类型的流量，比如 keyword 流量、信息流流量、banner 流量，然后交给不同级别的 Ad Search Leaf 模块对广告进行召回，选出相匹配的广告初步候选集，该模块就是我们今天的主题。然后将选出的广告初步候选集交给 Ad Selector 模块进行精排 ,即对广告的 CTR 或者 CVR 预估 , 进行打分 , 选出 top K 个广告返回给 DSP Server。同时 DSP Server 会将点击、曝光、后续的日志写入 kafka，并进行日志落地，日志落地后会进入后续的 ETL 离线流程，用于后续粗排、点击率等模型生成训练样

本使用，另一部实时数据日志数据流会经过反作弊模块后过一遍 ctr 预估的 online learning 和实时曝光反馈的 online feedback。

3. 召回模块的演进脉络

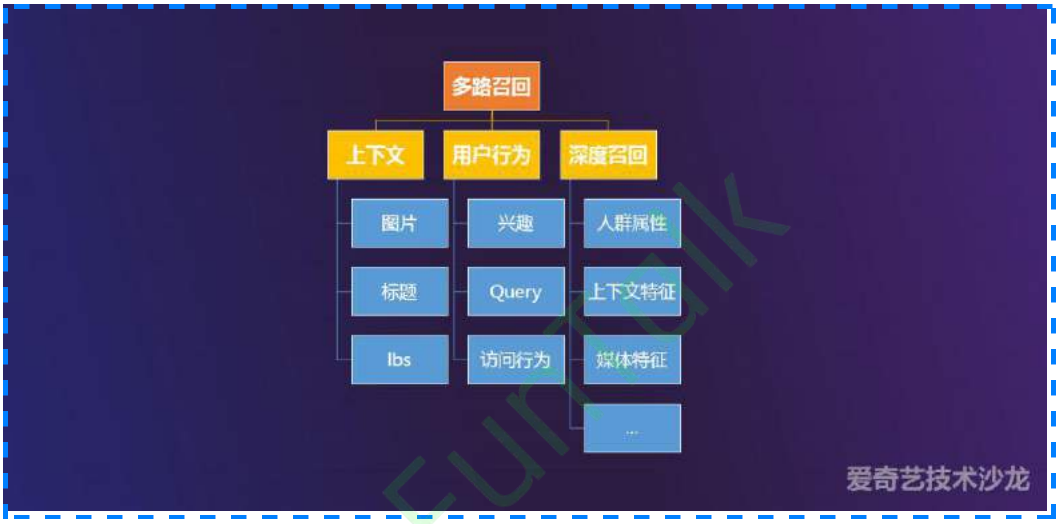
3.1 检索召回模块



Ad Search Leaf 召回模块展开来主要分三个阶段，召回、过滤和粗排。召回模块接收到 DSP 给流量请求中包含了用户信息和上下文信息。广告主设置的广告投放存在 Ad Meta db 中，然后建成相关的广告索引（Ad index），投放更新后才会实时更新这个索引。RTDB 主要存储用户标签数据，标签数据一部分是通过请求中带的用户信息和上下文信息进行用户标签的实时更新，另一部分是通过线下离线全量更新。召回模块就是结合这两端选出初步的广告候选集，然后进入过滤模块（正排模块），过滤方法主要包括基于规则、黑白名单、广告主预算 pacing 过滤。最后进入粗排模块，对初选的广告候选集

按评价函数模型进行打分，但没有精排模块那么复杂，相对比较简单，比如最开始基于核心特征的 LR 模型，后续升级过基于特征交叉的 FFM 模型。

3.2 召回通路



接下来讲一下在我们检索召回中一些通路，我们是进行多路召回的，其中有三种类型，第一种是上下文、第二种是用户行为、第三种是当前正在做的深度召回。

上下文召回有这几种类型，第一种是基于图片的，在内容页场景下，一个明星穿了哪件衣服，根据这件衣服投放哪件商品，做法是将图片向量化，计算广告商品与图片向量的相似度进行召回。第二种是基于标题的，主要是基于文本 NLP 相关模型进行召回。第三种是基于 lbs 的，广告主自身设定某个标签区域进行投放，在该区域内进行标签匹配召回，属于布尔召回。

用户行为召回有这几种类型，第一种是基于兴趣的，基于用户历史行为建立用户画像，打上兴趣标签，进行布尔召回。第二种是基于 Query 的，利用用户的 query 历史行为 ,进行 NLP 相关模型进行召回 ,与基于标题的方式类似。第三种是基于访问行为，利用广告主回传的用户商品行为，采用 Item CF、ALS、Neural MF 等模型进行召回。

最后一种召回通路是深度召回，主要是把 user profile、媒体特性、上下文特性等特征结合起来进入深度模型中进行召回，接下来会具体介绍下这个是如何做的。

3.3 基于文本的召回

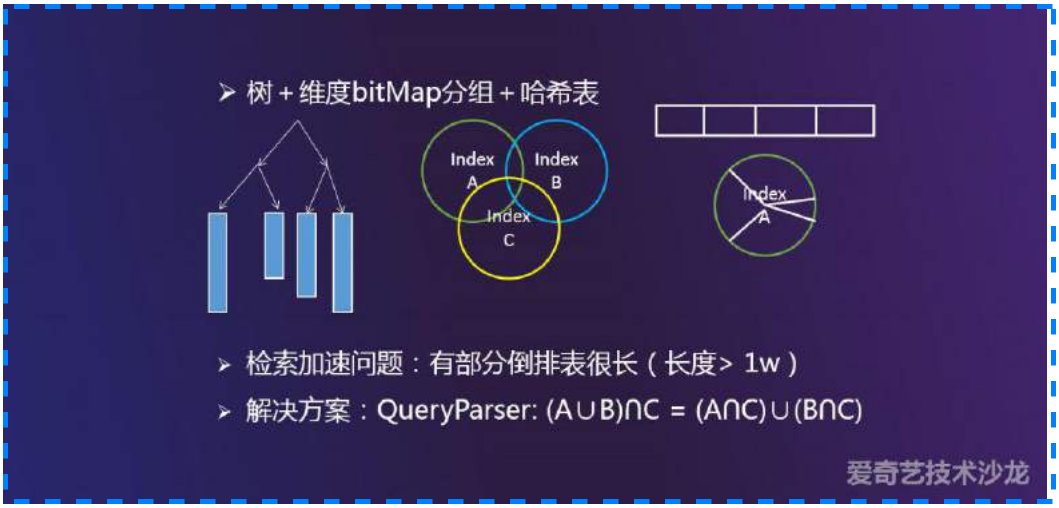
上面讲的标题和 query 召回主要是文本召回，包括几种，第一种是精准匹配，有完全匹配和基于 ngram 的 TF-IDF 提取核心词匹配 ,属于比较简单的方式。第二种是模糊匹配，有 word2vec 和 DSSM 语义化模型 ,将文本语义向量化，

然后按向量化检索召回。第三种是广泛匹配，是把语义化向量聚类成多个标签，然后按标签召回。

3.4 召回模块演进

我们的召回模块演进主要是分成三个阶段，第一种是布尔召回，刚才介绍的信息标签类型和 lbs 都是基于这种。第二种是向量检索召回，利用深度学习模型将广告、用户等信息都映射为向量，然后进行向量检索召回。第三种是我们现在在做的基于深度树的匹配。三个阶段是由浅入深的过程，接下来我们依次介绍这三个阶段。

3.5 布尔召回

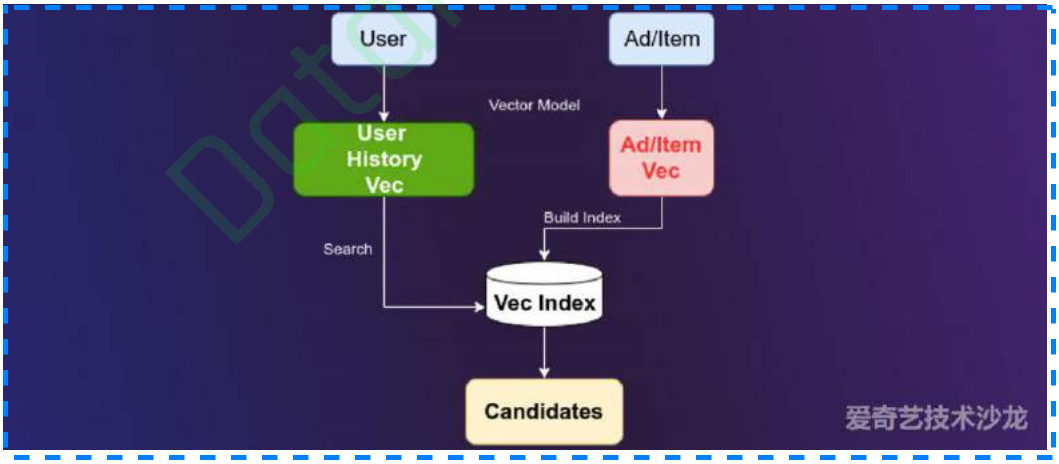


布尔召回在早期检索中都会用到的，我们的方式是基于树+维度 bitMap 分组+哈希表。广告主设置定向组合，比如访问某些网站的人群、有特定兴趣的人群等其他各种定向组合。而请求中会带有用户标签，问题就变成了怎么找到与用户标签匹配的定向组合广告？布尔召回本质就是基于倒排索引的布尔运算，所以关键在于倒排索引怎么构建？构建的方式有两个层级，第一层索引是将广告主的投放配置进行分解分组，每一个组为一个 conjunction，一个广告投放会对应多个 conjunction，这样就可以建立 conjunction 到广告投放的倒排索引。第二层索引在于根据用户标签找到对应的 conjunction，我们的标签是一个 int64 类型的整数，高八位的某些 bit 位会做一些标识，来区分这个标签属于哪种类型，拿到用户的标签做位运算，找到 bitMap 分组的 conjunction list。然后到基于每个 conjunction 到第一层取出对应的广告主集合，最后计算每个集合的交并，得到最终召回的广告候选集。

在实际使用中布尔召回有一些问题，比如倒排表有部分 conjunction 对应的广告集合很长（ >1w ），检索性能很差，我们的实际解决方案是在布尔表达式中做一个转换，比如将先并后交的布尔运算改为先交后并的布尔运算，能极大优化检索的性能。

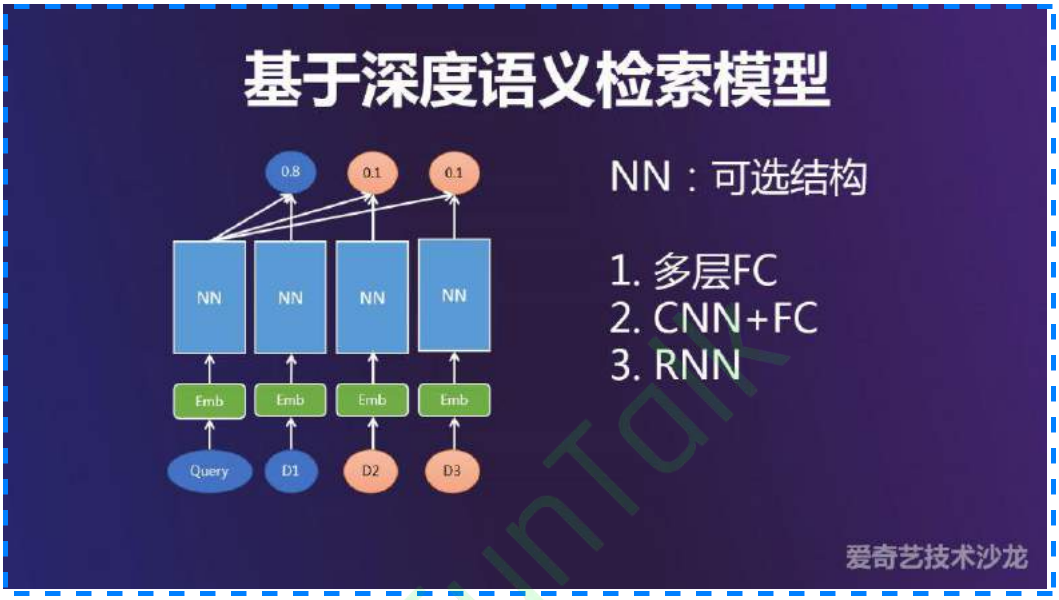
3.6 向量化召回

我们在实际向量化召回中有两种类型，第一种是类似于早期 YouTube DNN，把用户变成一个向量 ,把 item 也变成一个向量 ,最后做一个向量相似度检索。第二种是用户相关历史内容变成一个向量，再进行向量相似度检索。两种类型都是基于两个向量的相似度进行检索，得到 item 候选集。



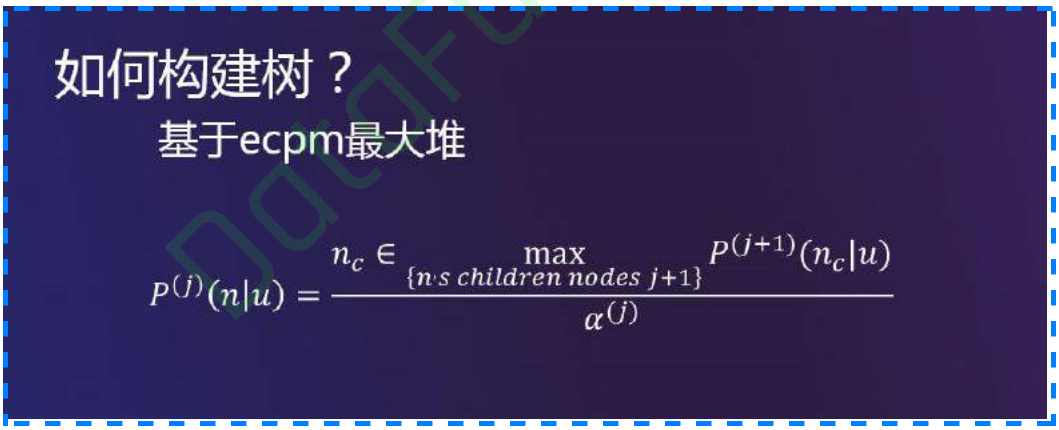
实际中我们选择的是微软提出的基于深度语义检索模型（DSSM），该模型的输入是一组相关的 Query 和 Document，以及两个与 Query 不相关的 Documents，然后分别 dense 为 embedding 向量，再经过 NN 网络，再做 softmax 打分，得到三个分数，训练目标就是是相关的 Document 分数尽可

能高，不相关的 Documents 尽可能低。中间的 NN 层可以有多个选择，可以为多层 FC，也可以类似 textcnn 那样为 CNN+FC，或者为 RNN。



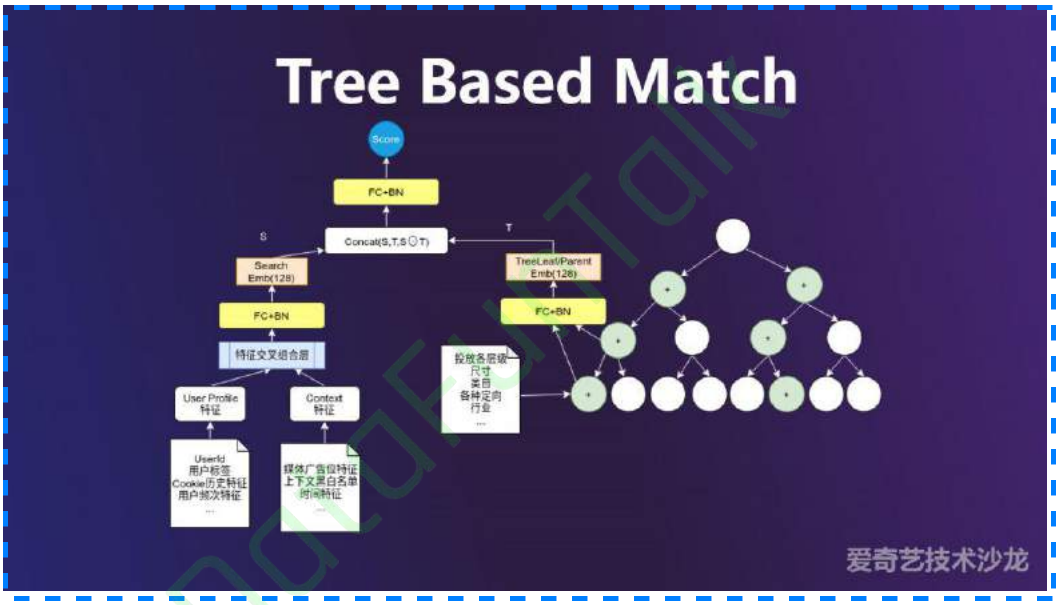
在通过模型向量化之后就涉及到向量索引该怎么选择？下面介绍下常见的几种。第一种为 LSH（局部敏感哈希）索引，方法就是基于多个哈希函数进行分桶，比较耗内存。第二种为 faiss 提出的 IVF Flat，方法是构建由聚类中心向量构成的倒排表，针对输入向量，先找到聚类中心向量，再基于 KNN 方式进行检索，这种类型是保留精度的，效果跟直接暴力检索非常接近，但性能有很大的提升。第三种为 IVF PQ，也是基于倒排，与 IVF Flat 区别在于在向量上做了有损压缩或 PC 降维，好处在于省内存，但是带来了实际检索精度的丢失。这三种可以根据实际业务场景进行选择。

3.7 深度树召回



上面讲的布尔召回和向量化召回都有一定的缺点，比如布尔召回的有些 conjunction 的候选列表非常长，容易遇到性能瓶颈。向量检索会局限于模型的向量表达是否准确，而且向量空间有限。这里参考了阿里妈妈提出的深度树匹配（TDM）模型，深度树匹配能解决全量检索的性能问题和模型精度问题。如何构建一个深度检索树呢？在广告场景下选取的是基于 ecpm（千次曝

光预期收益）最大堆方式，最大堆树下当前层最优 Top K 孩子节点的父亲必然属于上层父节点的最优 Top K ,这样就可以从根节点逐层递归向下挑选 Top K 直至叶子层，同时极大提高检索速度。为了简便起见，我们构建树的形式是二叉树，则检索 Top K 的时间复杂度是 2KlogN，N 为 item 个数，且随着 N 的增长，相比暴力全量检索的性能提升更为明显。



深度树模型的大致结构如上图所示，分为两部分，左边是流量端，包括用户 profile 特征（如标签、cookie 历史行为、频次等）和上下文特征（如媒体广告位属性、黑白名单、时间特性等）。首先把这两种特征 dense 化变成 embedding，再做特征交叉组合，在我们场景下特征交叉采用的是 IPNN 方式，然后再经过 FC+BN 的多层 NN 网络，最终转化为 Search 端的 embedding，线上处理时，左边这个流程每次请求都会计算。右边是候选集端，是一颗检索树，叶节点就包含了广告投放信息，包括尺寸、类目、行业

等广告主设置的定向信息，叶节点和父节点本质都是同一维度的embedding，都会进入 FC+BN 层，形成与 Search 端 embedding 相同维度的 Tree Leaf 和 Parent embedding。左右两端的 embedding 做交叉+concat 操作进入 FC+BN 层，最终得到一个分数，与样本 label 按交叉熵损失进行训练。整体的模型架构就是这样。

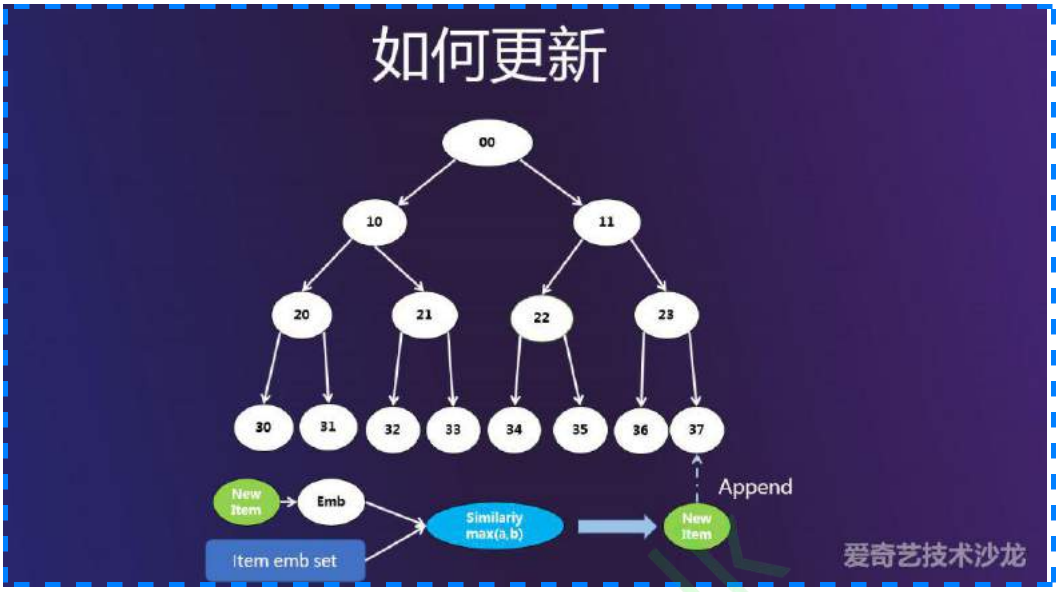
那我们要怎么准备这颗树模型的样本呢？有两部分，分别为 leaf 样本和 parent 样本，对于 leaf 节点，选取精排模型输出的虚拟曝光 Top3 作为正例。负例包括两部分，第一部分是在正例叶子节点同层，随机选取其他叶子节点作为负例，第二部分是选择粗排阶段的一些负分 item 标识为负例。对于父节点，基于 ecpm 最大堆原理，将正例向上回溯的所有父节点都标记为正例，同层随机标记为负例。整体的样本规模是用了 7 天的采样曝光样本，大概在 5000 千万左右。

接下来我们该如何生成这颗树呢？我们这里有两种方式，第一种方式就是随机生成，将 item 随机放到叶结点上，当然也可以根据标签体系手动把相近的 item 放在同一个父节点下，基于随机深度树训练一轮，将叶子结点的 embedding 导出来，然后聚类，这样就可以根据聚类的结果重新生成一颗树，然后再去训练，一直迭代到指标变化不大。第二种方式就是自下而上合成树，先当这颗树不存在，只训练每个叶子结点的 embedding，然后根据叶子结点的 embedding 向上聚类回溯成一颗树（kmeans 或者根据曝光频次来聚类），接着就可以与第一种方式一样一直迭代直到指标变化不大。

我们最后选择方式是第二种方式，第一步就是先训练叶子，把叶子 embedding 导出来，然后聚类生成一颗树，基于这颗树结构重新训练叶子结点和父节点的 embedding，得到最终上线的模型。

接着我们来讲一下这个模型的 loss，我们的目标是去拟合曝光，所以 loss 由两部分组成，第一部分是交叉熵损失，去拟合对应的样本是否曝光。第二部分是 triplet loss，含义是叶子节点中正样本之间距离尽可能相近，而与负样本之间的距离尽可能远，来约束叶子距离。第一部分 loss 考虑的是拟合曝光，并没有考虑点击，所以可以把实际用户点击的样本进行加权，把即曝光又点击的样本着重考虑，体现重要性。

线上如何检索呢？因为我们是基于最大堆的，所以采用的基于 beam search 的方式，就是逐层往下的检索，每一层都保留 K 个节点并往下扩展。



实际线上投放是有变化的，广告主会新增一些投放，对于这种新增投放怎么添加到这颗树中呢？比如新增一个 item，我们会根据训练好的模型把 item 对应特征转换为 embedding，然后把之前训练的叶子节点的 embedding set 拿出来，然后做一个相似度检索，得到 item embedding 最接近的叶子节点 37，将该新 item 加入这个叶子节点 37 的 list 中去（线上训练时，叶子节点实际是一个 cluster，一个列表，并不是一个单一的节点）。

4. 性能优化

最后提一下我们在实际中做的一些性能优化工作，主要介绍两方面：

第一方面是在检索阶段，针对树的最上几层，会根据 Top K 的大小，进行跳层（父节点个数<K），针对最后几层可以返回 cluster，交给后续精排业务去做排序。同时可以进行多路并行，提高性能，也可以将训练的 double 数据裁剪为 uint8 类型，针对相对排序指标影响不是很大。

第二方面是训练阶段，因为我们是基于 tensorflow 构建模型的，所以有常见的几种优化方式，比如避免使用 feed dict，使用 dataset 这种高阶可以并行化的 api。同时可以使用 Timeline Profile 等工具分析性能瓶颈，进行针对性的优化。针对 python GIL 的缺点，将高频调用的 python function 利用 c++ 实现进行调用，并在调用前将 GIL release 掉，避免锁带来的性能影响。

参考资料：

1. Learning Tree-based Deep Model for Recommender Systems

https://arxiv.org/abs/1801.02294

2. 微软 DSSM 项目地址：

https://www.microsoft.com/en-us/research/project/dssm/

DataFunTalk

解密商业化广告投放平台技术架构

嘉宾：辰序　编辑：Hoh Xil

导读：互联网广告是流量商业变现的重要途径之一，涉及服务平台、检索引擎、算法策略、数据工程等多个方向。本次分享的主题为商业化广告投放平台技术架构，分享的内容集中在工程领域，结合业界广告投放平台的通用技术范式，分享智能营销平台是如何打造高性能、高可用、可扩展平台架构的，从服务化、数据传输分发、广告投放引擎、计费、海量数据实时报表等方向切入，深入浅出的阐述一套最佳实践。

——业务介绍——

1. 广告业务简介

- 商业化是互联网公司营收的重要来源
- 业界：Google AdWords / Facebook / 百度凤巢 / 头条巨量 / 腾讯广点通
- 阿里电商场景变现平台 – 阿里妈妈
- 国内市场规模年千亿规模
- 计算广告学

商业化是互联网公司营收的重要来源，业界比较大的商业化产品有 Google AdWords、Facebook Ad、百度凤巢、头条巨量引擎、腾讯广点通等产品，阿里电商场景的变现平台是阿里妈妈。目前国内互联网广告的营收年规模达数千亿元。

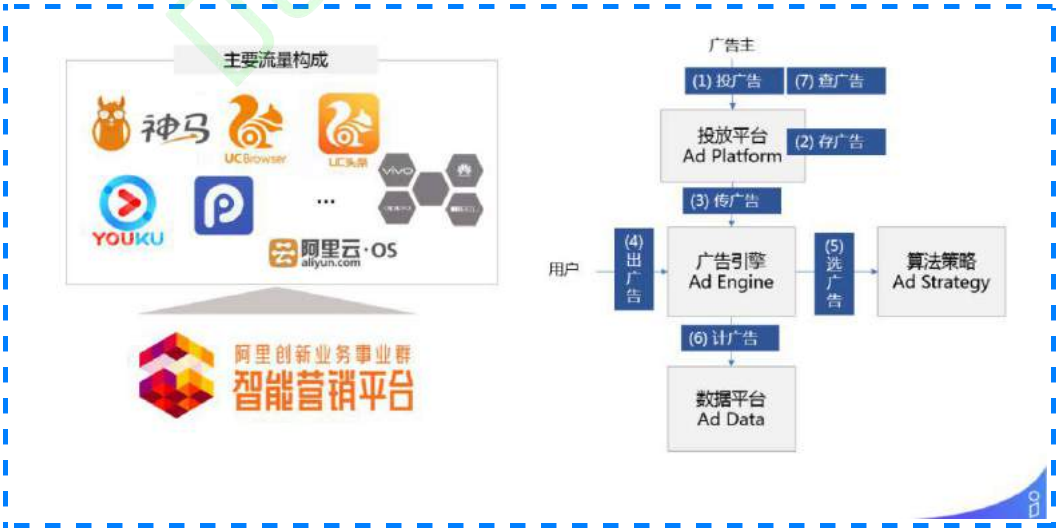
广告平台一般由三方组成：网站主、广告主、用户。

网站主：有流量资源的变现需求。

广告主：有投放预算，希望在流量资源上找到合适的人进行推广。

用户：得到更好的广告推荐结果。

2. 智能营销平台简介 & 通用的广告系统组成



阿里创新事业群智能营销平台的主要流量构成包括：神马搜索、UC 浏览器、UC 头条、优酷、阿里云 OS 以及手机厂商、网盟第三方流量等。

通用的广告系统组成如右图，主要包括 4 部分：投放平台、广告引擎、算法策略、数据平台。

运转的流程如下：

首先，广告主有一定的预算，选择在什么样的流量上投广告，这样就有了输入；然后存广告，通过投放平台来完成，把投放所需的设置、预算、创意等数据实时传输到检索系统；每当用户发起一次检索请求时，简单来说后台相当于做召回+优选，把最合适的 TOP 候选广告返回给用户；还有一部分是计广告，通过数据平台来完成；最终广告主来查广告绩效展点消数据。

本次分享主要聚焦在如下 5 个环节：

——投广告——

业务微服务化构建

第一，来看如何投广告，包括无状态的服务层，来满足广告主进行预算设置、存广告、建创意等操作诉求。

1. 单体模式到微服务化（MSOA）的过渡

任何一个大的产品，都会遇到从单体过渡到微服务的过程，其架构的实施方法论落在三个字：分、合、细：

分：分化、隔离、获得技术、业务上的纵深和弹性。

合：随着业务的增长，要加大复用度，归并同类项。

细：细化分工、架构，深入优化技术&业务。

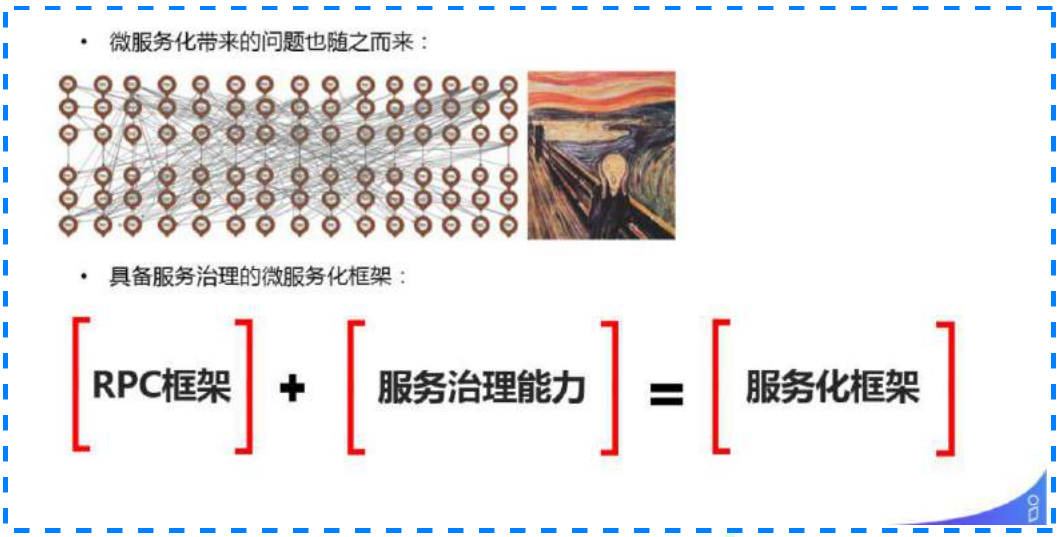
当业务发展到足够大的时候，最终会采用 MSOA 架构。

分合细的基础在于：如何把这些服务给拆开，对于投放平台来说，需要对投放需求进行规范化的表达和产品功能的标准化设计。把这些领域模型建起来之后，服务自然就有了划分的依据。

2. 微服务化分层体系建设

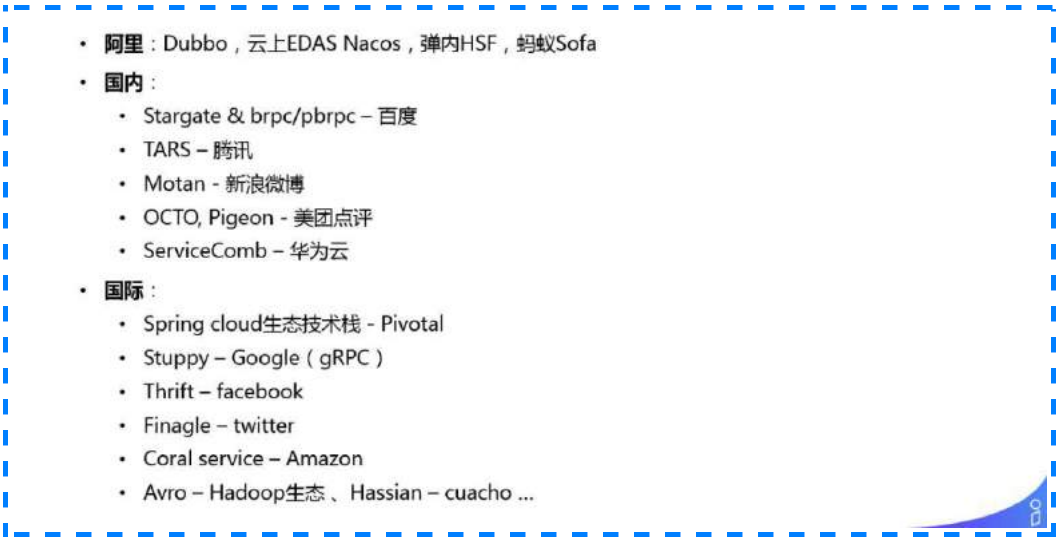
上图为拆分之后的投放服务架构图，水平拆分成簇，纵向拆分成层。外层是 API 和 WEB 端，进行相应的权限验证、流控等；中间为计算层，细分出来各个服务，包括面向业务流程处理、业务逻辑组件以及公共服务组件三层；最底层是基础设施和数据资源。把所有的服务串联起来的就是分布式服务化框架。

3. 微服务化与服务治理



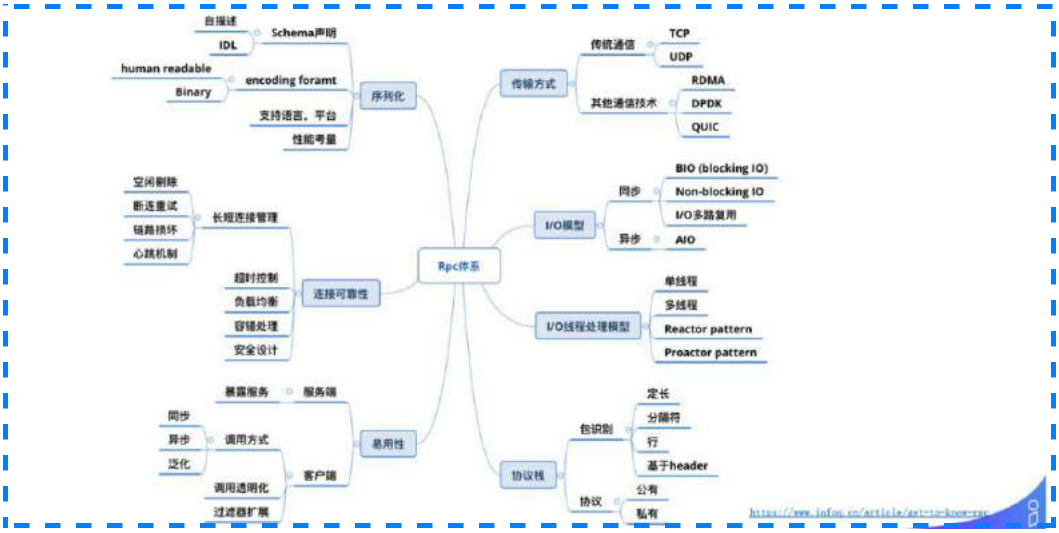
做平台方向的团队可能需要几十人上百人，而服务则可能有几十上百个，它们所组成的网络会非常混乱，最终难以治理。我们需要具备服务治理能力的微服务化框架来解问题，包括 [**RPC 框架**] 和 [**服务治理能力**]。

4. 分布式服务化框架一览



业界的分布式服务化框架一览，如上图所示。这里有的是 RPC 框架，有的才是真正合格的服务化框架，对于投放平台，需要的是一个服务化的框架。我们拆分来看，先认识下 RPC 框架：

① 体系化认知 RPC



RPC 体系的脑图如上所示，作为 PRC 框架应具备如下功能：

1. 传输方式：我们要看是采用的传统 TCP 或者 HTTP 技术，还是运用的最新技术，如阿里很多基础产品是用 RDMA 网络的，或者采用一些用户态的协议，不走 TCP 协议，来做 kernel bypass。

2. I/O 模型：是采用同步还是异步的，可以选用 Blocking IO 或者 Non-blocking IO，I/O 多路复用最有名的是基于 epoll 的，大部分走 TCP 协议的都采用的是 epoll；异步的 AIO 网络通信在 Windows 下用的比较多，在 Linux 上 AIO 主要应用在磁盘 IO 相关。

3. I/O 线程处理模型：可以分为单线程、多线程，以及各种 pattern，大多数情况下都会选择 Reactor pattern，基于 I/O 多路复用 epoll 这种形式。

4. 协议栈：数据可以点对点传输之后，如何识别这些数据，主要就是协议栈的工作。大多数框架都采用基于 header+payload 的方式。

5. 序列化：当点对点的实时的识别出一个具体的包之后，需要序列化的解析包里的数据。

6. 连接可靠性和易用性：可以快速的传输和识别包之后，还要面临可靠性的保证，以及暴露给上层的调用方式，如何更好的调用服务，是用同步、异步，还是泛化调用类似的问题是需要考虑的。

这些范畴，共同组成了 RPC 体系，但是光有这些是不足的，最终还要落在服务治理上：

② 服务治理能力

服务治理能力包括：

- 通信传输：这是最基本的。
- 服务发现：包括服务注册，自动的下发和发现；客户端路由的规则，是同机房路由还是打标分组路由；以及负载均衡的策略，是 RR 还是 Random 等；有了这些之后，可以做服务的拓扑和强弱依赖分析。
- 监控度量：从 logging、tracing、metric 三个维度进行监控。
- 健壮容错：如何 design for failure 做处理异常，做服务的降级和限流。
- 契约管理：管理服务的 IDL 以及进行兼容性分析。
- 持续交付：开发测试，整个应用的生命周期管理，更多的是与 PaaS 平台相结合。
- 安全保证：应用鉴权、服务鉴权、用户鉴权、传输加密等。

通过这些模块，共同构成了服务的治理。

现阶段我们的服务治理方案如下：

- 服务化框架：HSF

- 容器隔离：Pandora-boot
- 配置管理：基于 Diamond
- 监控度量：Metrics+Tracing+Logging（ARMS+SLS+鹰眼 ）
- 健壮容错：Sentinel
- 基础设施：基于 K8S 的云平台
- 持续交付：基于 Aone 的 Paas+CD 平台

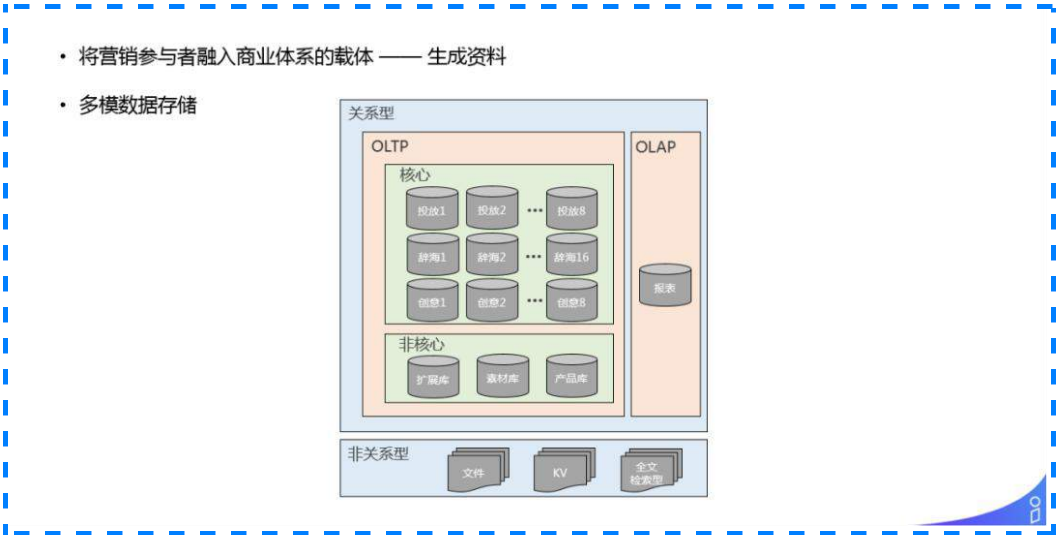
发展未来上来看，云原生是一个方向，代表了先进的生产力，这当中服务的架构设计、开发交付都会被重塑，一些变化主要在于：

- 资源效率：重点在于容器化、调度、混部。
- 开发效率：注重编排、敏捷 CI/CD；Service Mesh、Serverless 等基础设施的下层和 business logic focus 的专注。
- 标准与开放：开源、社区和生态的建设。
- 个人理解：**万变不离其宗，云原生解决的问题覆盖了服务治理，天然的利用"云" 去解决，而非某个框架或者 vendor lockin 的能力。**

——存广告——

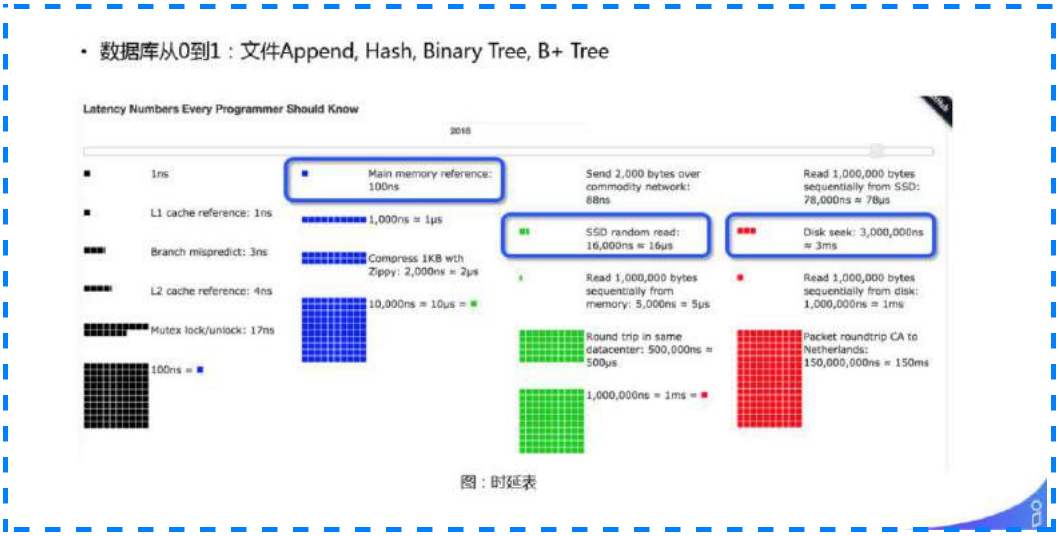
OLTP 海量数据存储

1. 广告数据存储



数据存储可以看做有状态的存储层，要把这些生产资料，存在广告系统中。我们采用的是多模数据存储的方式，OLTP 大部分都是 MySQL 分库分表，OLAP 有相应的报表平台，非关系型的有 OSS、KV 存储以及基于 ES 的全文检索。

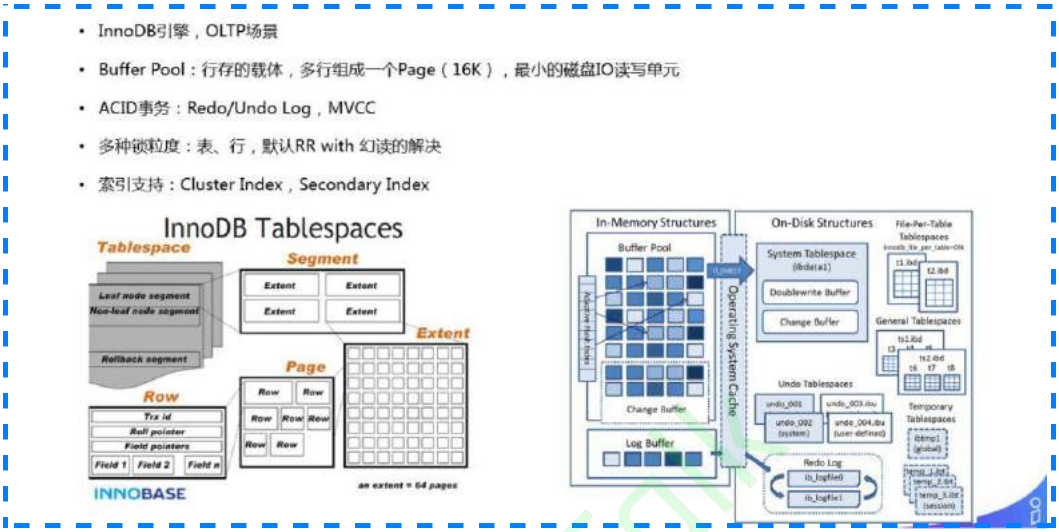
2. 体系化认识数据库



图：时延表

如果数据库从 0 到 1 开始做，最简单通过一个文件把相应的字段存起来，可以用到的数据结构有：Hash，Binary Tree，B+Tree。但是采用这些方式，会面临着不同的时延问题，如图所示：访问内存大概是 100ns，访问 SDD 大概是 16µs，但是一次 Disk seek 需要 3-10ms。所以，为了减少随机 IO，Hash 和 Binary Tree 都是不合适的。因此，有了 B+Tree 这种 MySQL 使用的索引数据结构。

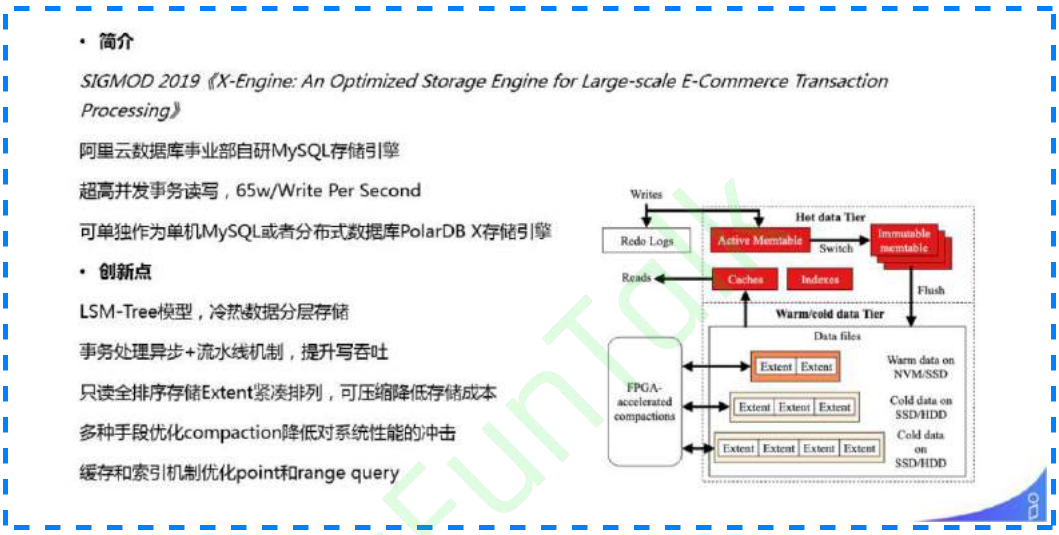
① MySQL InnoDB 引擎



我们大部分数据都是 MySQL 存储的，且采用的是 InnoDB 引擎，主要面向的是 OLTP 场景。InnoDB 引擎是行存，多行组成一个 Page，多个 Page 组成一个 Extent，而每个索引的叶子节点和非叶子节点又由 Segment 这个概念维护，最终形成了一张表（tablespace）。每次检索实际就是按照刚刚提到的 B+ Tree 做点查或者范围查询，可以走 clustered index 或者 secondary index 或者 full scan。

我们不止要把数据存起来，还需要有 SQL 和 ACID 功能，Durability 是靠 Redo Log 来做的，保证数据不丢失；还要满足 Isolation，也就是事务之间是有并发的，并且要进行隔离，不能互相影响，采用 Undo Log+MVCC 机制来实现。InnoDB 还有很多好处，比如它是基于行锁的，有索引的支持等。右图为 MySQL 官方的架构图，大家可以官方查询下资料。所以，通过 MySQL 可以把广告数据高并发、可靠的持久化存起来。

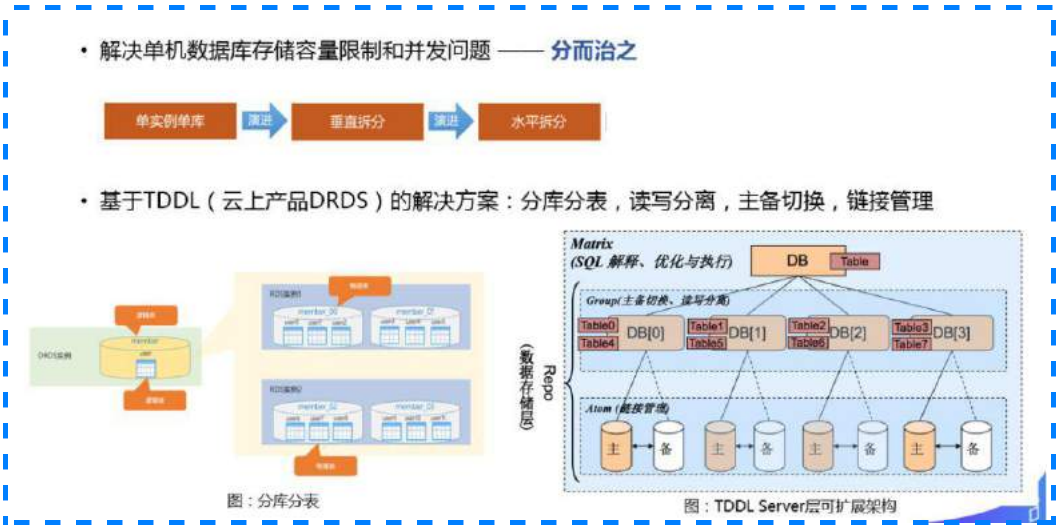
② 阿里云自研 X-Engine 引擎



在海量数据高 TPS 场景下，阿里也在做一些优化。这里介绍下 X-Engine 引擎，阿里在 SIGMOD 2019 发表的一篇论文：《X-Engine：An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing》，X-Engine 可以实现超高并发事务的读写，可以达到 65w/Write Per Second，读的话和 InnoDB 差不多，主要是优化写的方向；可单独作为单机 MySQL 或者分布式数据库 PolarDB X 存储引擎。马上在阿里云上就可以看到云上产品。X-Engine 的创新点，如图所示，这里不再细说。

3. 数据库存储架构

① 可扩展

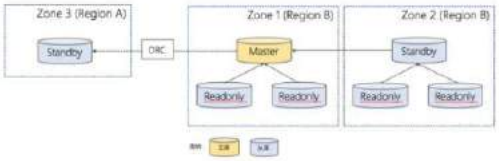


刚刚介绍了存广告的基础，由于广告产品产生的数据量非常大，尤其是搜索广告的关键词维度，基本都是百亿的量级，如何做扩展，是一个很重要的问题。常用的解法是**分而治之**，做垂直拆分和水平拆分。

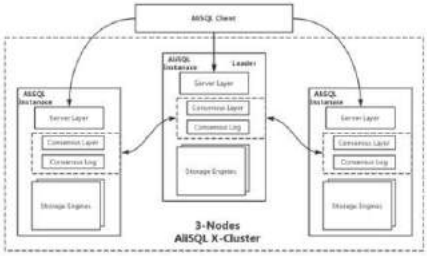
我们是基于 TDDL（ 云上产品 DRDS ）的解决方案，来做分库分表，读写分离，主备切换以及链接管理的。

② 高可用

- 异地高可用：两地三中心，三地五中心
- 异地多活：基于Paxos状态机复制
- 单元化部署：将“分库”上移到无状态服务层的一种方案



图：两地三中心数据库架构



图：3节点AliSQL X-Cluster架构

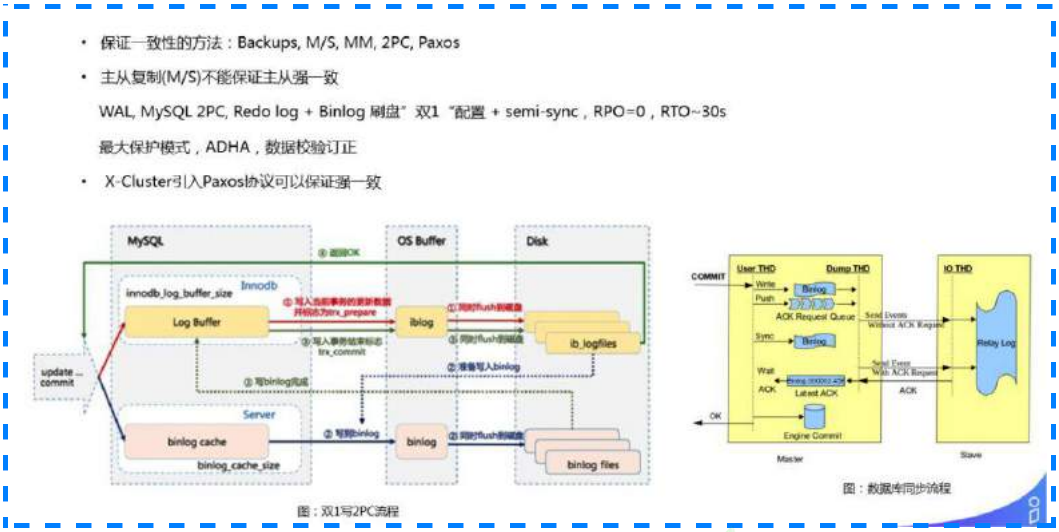
对于数据库高可用的保证，常常采用：

异地高可用：两地三中心（ 如左图，主库和备库都在同一个 Region B 中，但是它们分了两个机房来做相应的同步，同时通过 DRC 把数据异步的传到另外一个 Region 中，这两个 Region 可能距离非常远有上千公里，所以是两地三中心的架构 ），可以扩展成三地五中心。

异地多活：基于 Paxos 状态机复制，可做跨地域一致性保证。

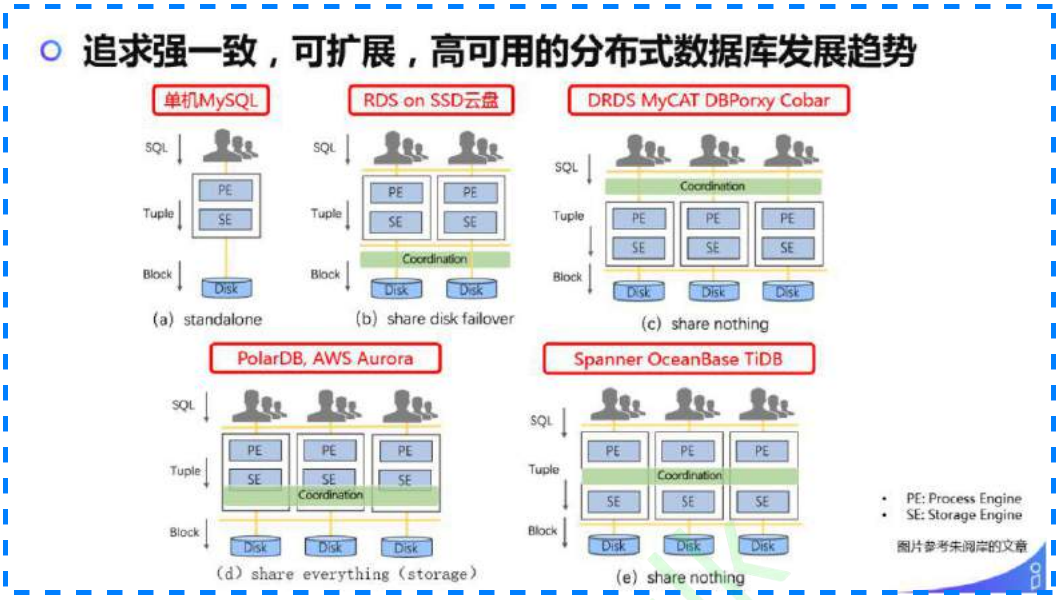
单元化部署：将 "分库" 逻辑上移到无状态服务层的一种方案，某一批用户被固定路由到某个地域，一套产品形成若干个封闭单元。

③ 一致性



基于传统的 MySQL 主从复制机制，使用 "双 1"+semi-sync 的保守配置，还是不能保证主从一致性，如图所示这里不再赘述。为了解决这个问题，我们有很多的 workaround，如开 MP 最大保护模式，强制主从必须同步，或者阿里内部有很多工具，例如 ADHA 来做主从切换的保证，以及数据的校验订正等。现在流行的优雅解决方案还是要依赖于 Paxos 协议保证多副本强一致，例如阿里云金融级 RDS。

4. 分布式数据库发展趋势



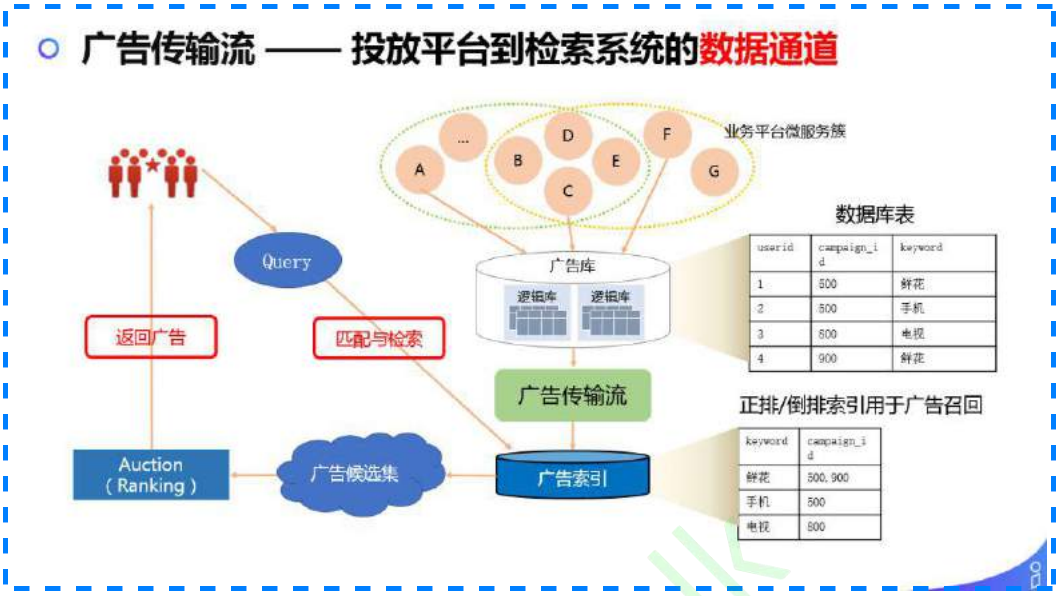
追求强一致，可扩展，高可用的分布式数据库的发展趋势主要分为 5 个阶段：

(a) Standalone -> (b) share disk fallover -> (c) share nothing -> **(d) share everything<storage> -> (e) share nothing**

目前我们处于基于 share nothing 架构的 DRDS 来实现，业务体量到达一定规模的时候，未来一定是朝着高扩展性、高可靠、成本优先的方向发展的，我们也会持续跟进数据库行业的发展，做更好的技术演进和选型。

——传广告——

广告传输流建设

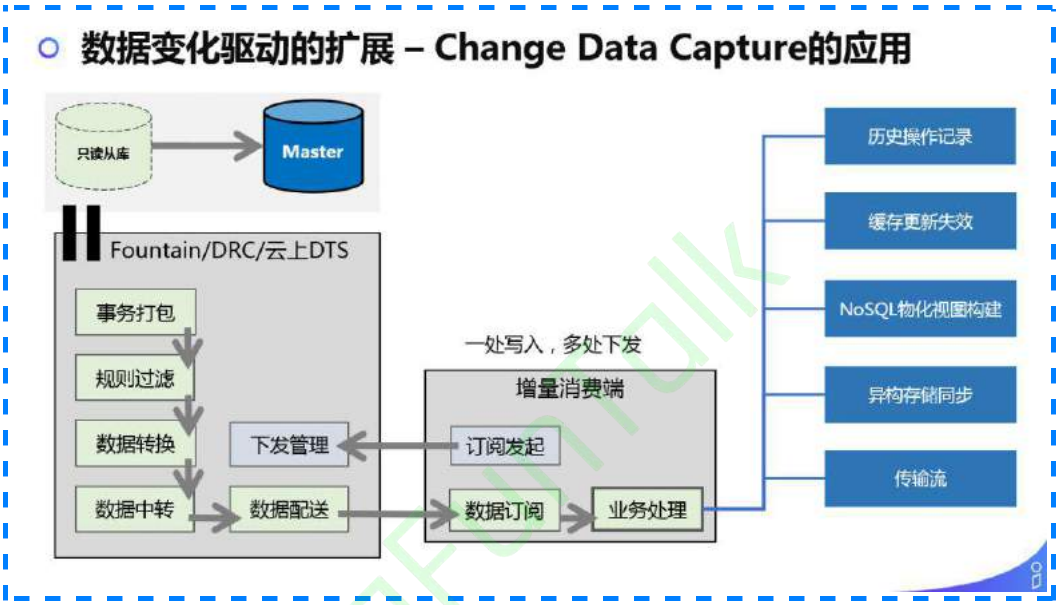


现在广告已经投完了，并且存在了数据库中，接下来就要把广告实时传输到检索系统。如上图，大致介绍了传输流。刚刚提到的服务层，可以有服务治理，广告库存在 MySQL 中，我们通过传输流实时把数据传输到检索端，做建库到广告索引中，以搜索场景为例，用户发起一个 query，需要匹配与 query 相关所有的广告，生成候选集，再进行相应的 Auction 和 Ranking 工作，最终返回给用户 1 条或几条广告。所以广告传输流非常重要，传输的是召回候选集的生成资料，是连接业务系统和检索系统的关键链路。

这条链路上，我们面临的挑战有：

- 海量数据，天级别几十亿增量。
- 高吞吐，Peak 20W+ QPS。
- 高可用，大于 5 个 9 的 SLA，跨机房容灾，计划下线等敏感数据不能延迟否则会造成收入损失。
- 低延迟，同机房数据库 binlog 订阅到检索 SLA TP95 < 200ms。
- 变化快，业务变更频繁。

我们的方案是分库分表之后，通过 binlog/DRC 组件来接广告传输流做实时增量链路，检索端订阅 message queue 来感知变化；全量数据 Dump 会存在分布式存储上，检索端可以天级别的 base+delta 的方式重建索引。



数据变化驱动的扩展，刚刚说的传输流只是其中的一个场景，可以扩展到很多场景中。都可以采用这种方式，做一个虚拟存库，开源的可以用 Fountain、Hiriver、Canal，以及阿里云的 DTS 等，可以做事务打包、规则过滤、数据转换等，最终到增量消费端，传输流只是其中的某一个场景来做平台端到检索端的同步，另外我们可以做历史操作记录，缓存更新失效，NoSQL 物化试图构建，异构存储同步等，想象空间会非常大，我们内部做了很多物化视图类的缓存和索引，辅助业务系统加速查询使用。

——计广告——

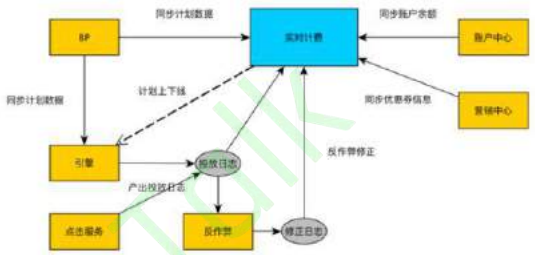
实时计费系统

1. 实时计费平台

- 支持多种计费模式：CPC，CPM，CPT，GD。
- 支持广告主实时调整投放属性，例如预算，有效期，投放时段等。
- 支持广告主实时充值，退款。
- 超投控制
- 计费系统特点：

  高并发、访问量大

  不丢不重、可用性高



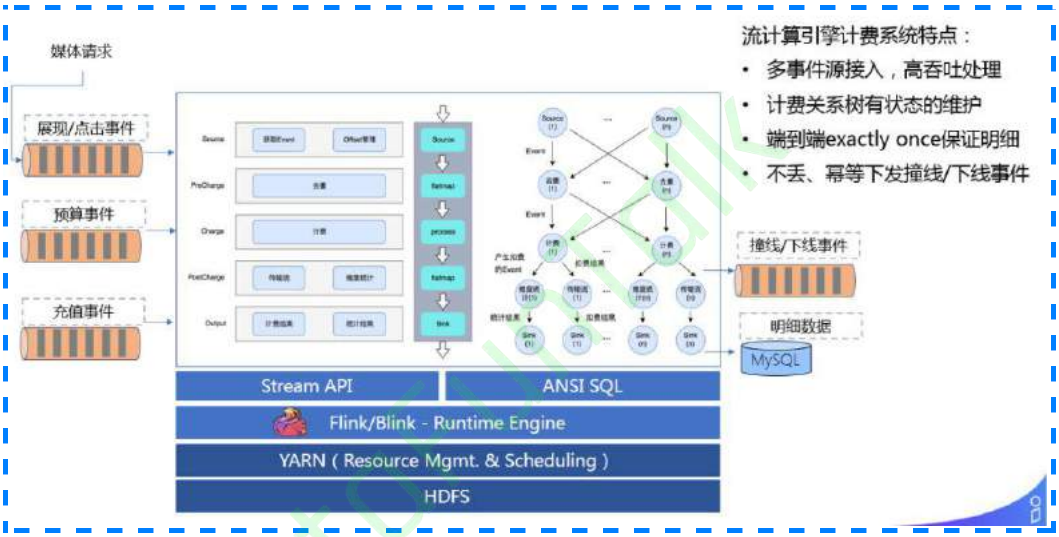
现在业界几乎都会实现一个实时计费平台，需具备基本的实时扣费和充值功能。计费广告最简单的模型是，广告主有钱，比如预算设置 1000 元，每次点击都要扣 1~2 元，最终扣完时，需要把这个广告下线，这是典型的 CPC 计费模式，还有 CPM，CPT，GD 等计费方式。计费平台还需要处理超投控制，避免平台利益受损。

实时计费系统特点是：高并发、访问量大、数据准确不丢不重、高可用性。

右图为实时计费平台的关系图：

用户中心来充值，业务平台同步一些计划的预算数据，引擎根据点击进行实时的扣费，计费系统要把计划上下线的信息，通过传输流实时传输给广告检索引擎。

2. 实时计费系统简介



我们过去的计费系统是基于单机或者分布式的系统来开发的，现在的计费系统跑在 Flink 上，我们接收媒体的请求，展现/点击事件、预算事件、充值事件这三类事件之后，基于 YARN、Flink 之上，通过 Stream API 来做相应的逻辑。每个计费事件来之后，会先做去重，然后再做计费，把明细数据存在数据库中，撞线/下线数据可以实时的通过队列传输给检索系统。采用流式计算引擎做计费的好处在于：

- 可以做到实时处理，敏感信息高优低延迟的幂等下发；

- 对于明细数据可以通过两阶段提交 sink 的方式，保证端到端 exactly once 语义，然后做聚批高吞吐的写入数据库；

- 有状态的存储账户状态，在面对 failover，修历史数据场景下可以做自动和灵活的处理。

总结下流计算引擎计费系统特点：

- 多事件源接入，高吞吐处理

- 计费关系树有状态的维护

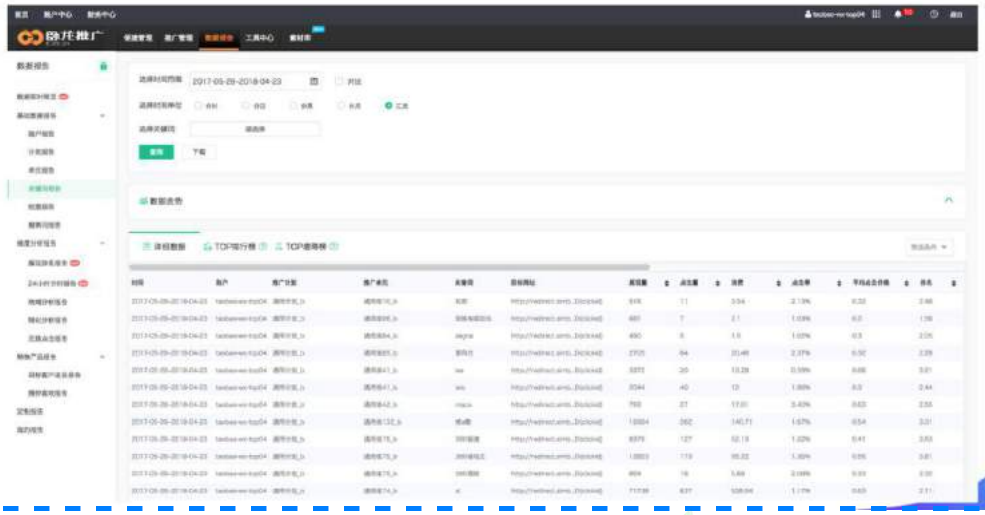
- 端到端 exactly once 保证明细

- 不丢、幂等下发撞线/下线事件

——查广告——

OLAP 海量数据报表

1. 数据报表



计费、展现的数据，都要给用户查看或进行分析。上图为某数据报表截图，某账户、单元、关键词维度下的展现、点击、消费数据。

报表是广告平台的核心业务，报表数据的披露与分析是投放优化的起源。面临的问题：

数据量大：总数据千亿级，单表最大百亿级；大账户单表数据近 2 亿。

快速响应：支持任意时间段查询实时返回；并发高&平均响应百毫秒级

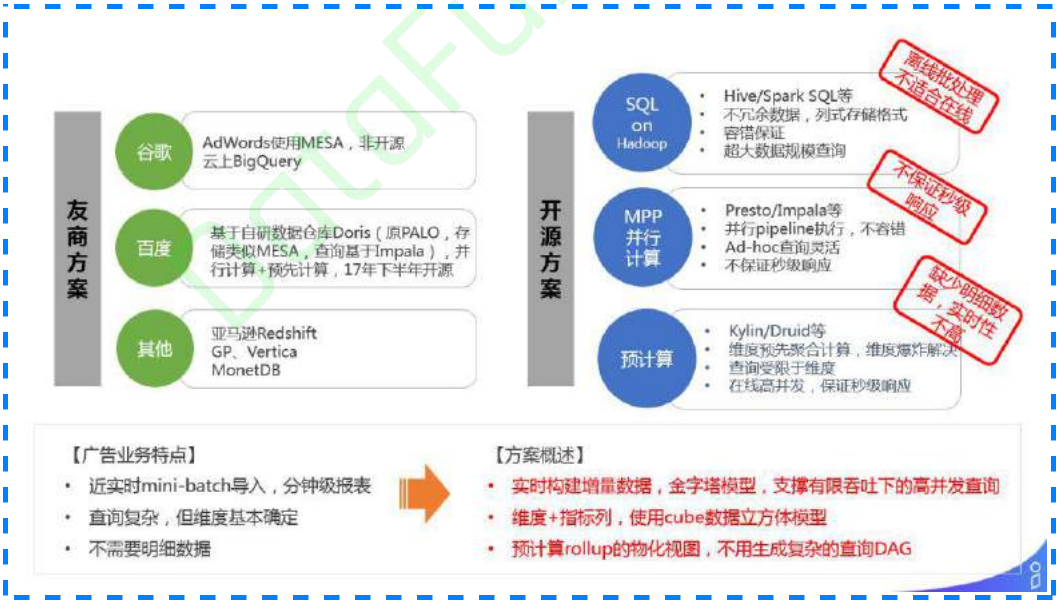
查询复杂：

① 时间聚合：分月，分周，分日，分时等。

② 维度聚合：物料层级，分地域，搜索词等 30+。

③ 功能多样：分页，排序，汇总，导出，对比。

2. 技术选型思考



技术选型上的思考：

友商方案和开源方案如上图所示。根据我们广告业务特点：

- 近实时 mini-batch 导入，分钟级报表需求
- 查询复杂，但维度基本确定
- 不需要明细数据

我们的选型方案：

- 实时构建增量数据，金字塔模型，支撑有限吞吐下的高并发查询
- 维度+指标列，使用 cube 数据立方体模型
- 预计算 rollup 的物化视图，不用生成复杂的查询 DAG

3. 实时报表平台

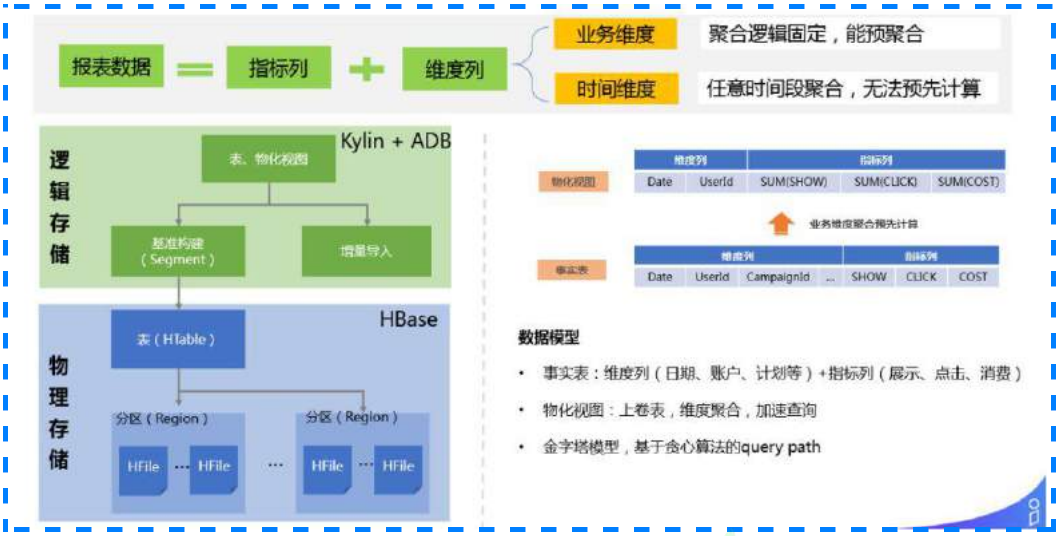
最终我们的实时报表平台，包括 4 层：

- 应用层：关注总吞吐量、总计算量，以及和业务的 join。
- 查询计算：关注高并发，百毫秒级查询延迟。
- 数据存储：有可扩展能力，并且有高效的 scan 能力。
- 数据构建：要保证增量构建和原子更新。

技术选型：

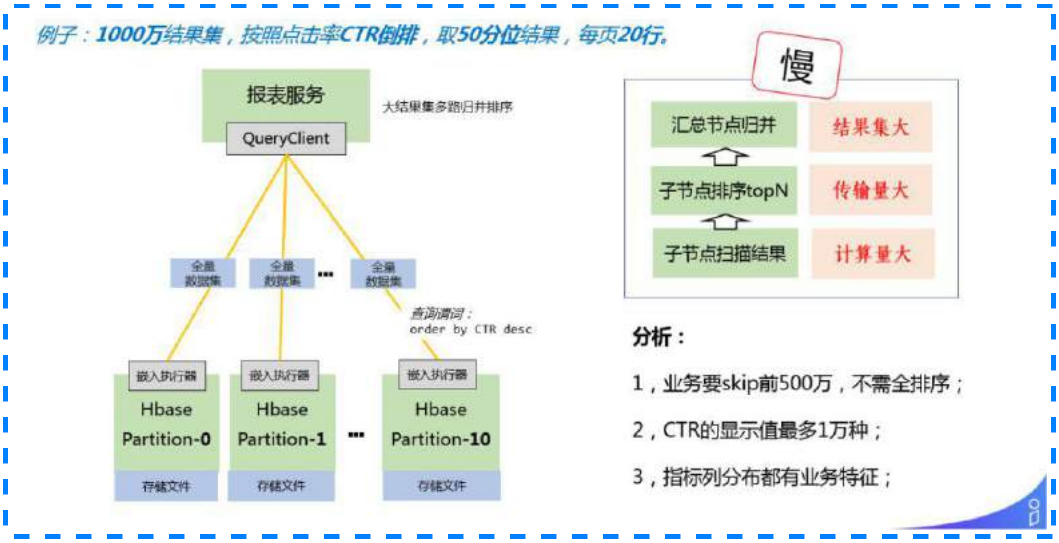
- 采用定制的 Kylin，是 MOLAP 多维的 OLAP 分析；复用数据构建与部分查询组件。
- 实时性保证，基于 Blink 计算分钟级数据，阿里云 ADB 实时导入事实表，Kylin 接入 ADS 来进行分钟级增量构建。

4. 报表平台关键技术点



报表数据分为指标列和维度列，支持业务维度（ 聚合逻辑固定，能预聚合 ）和时间维度（ 任意时间段聚合，无法预先计算 ）。基于 Kylin 来做历史和小时级数据的查询，基于 ADB 来做实时分钟级数据的查询。这里的创新点采用金字塔模型做基于贪心算法的 query path 优化，在扫描数据量、IO 上做最优的查询计划。

5. CBO 优化之 TOP N 深分页优化

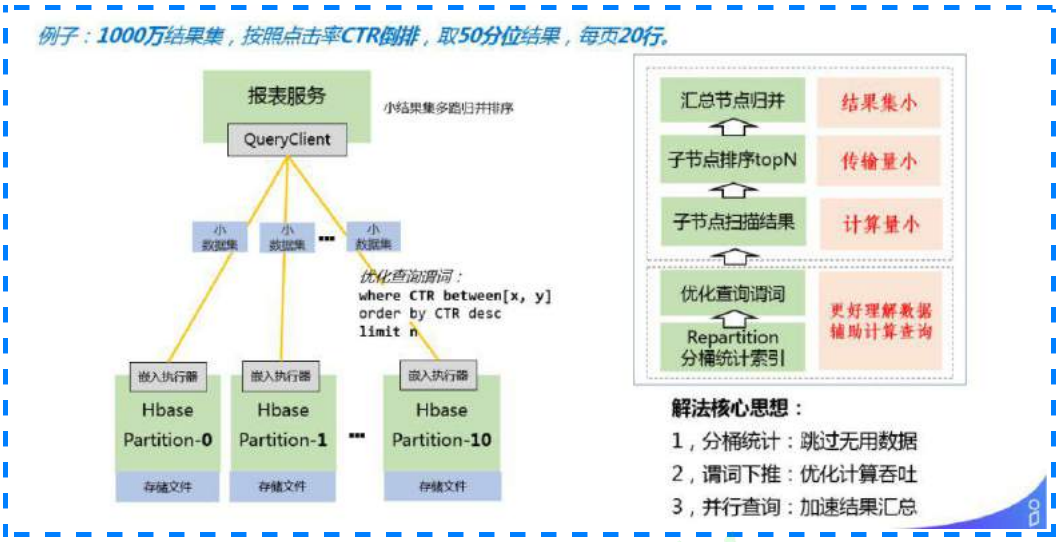


这里举例说明一个比较细的优化点，假设有 1000 万结果集，广告主需求按照点击率 CTR 倒排，取 50 分位结果，每页 20 行。往往很多系统都是禁止这些查询的，因为，一旦在大数据集上做深分页 order by 操作，返回的数据量可能非常大，在最终汇总节点是无法完成这项工作的。所以我们做了一些分析：

1. 业务要 skip 前 500 万，不需要全排序；

2. CTR 的显示值最多 1 万种；

3. 指标列分布都有业务特征。



解法核心思想：

1. 分桶统计：跳过无用数据

2. 谓词下推：优化计算吞吐

3. 并行查询：加速结果汇总

最终在我们的平台上是可以做海量数据深分页查询的，秒级响应返回。

总结：

上图为阿里巴巴技术体系大图：

- 基础设施：硬件网络相关的。
- 重点软件：分操作系统、数据库、存储、中间件等方向。
- 平台相关：计算平台、数据平台、算法平台等。
- 业务平台：业务上会走相应的业务平台和中台的方向。

- 应用层：最终服务于阿里内部众多的产品。

广告投放平台技术侧重点：

包括：数据库、中间件、计算平台、广告引擎、业务平台、广告业务。

本次的分享就到这里，谢谢大家。

DataFunTalk

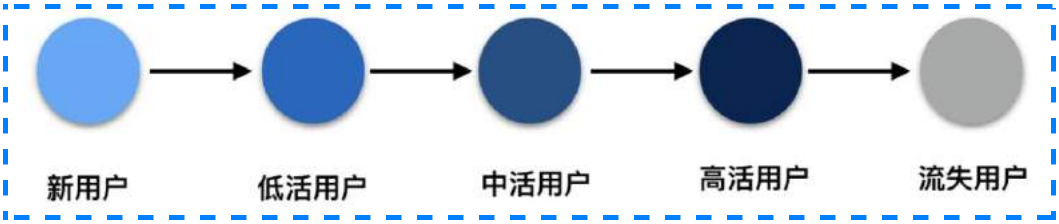
优酷 DSP 广告投放系统架构实践

嘉宾：鸿雁　编辑：Hoh Xil

导读：随着 RTB 网络在线展现广告交易模式的兴起，各大公司都纷纷搭建自己的 DSP（Demand-Side Platform）广告投放系统进行获客。优酷在近几年也搭建 DSP 系统，并且在持续迭代。在这一过程中，经历哪些技术探索？趟过哪些坑？有怎样的技术方案沉淀？下面我将从技术视角分享出来，希望对大家有启发。

——业务目标——

DSP 的核心目标就是用户增长，通过广告拉新和召回，提升用户留存和活跃度。从月活跃度的维度，我们将用户进行分层：新用户、低活跃用户、中活用户、高活用户和流失用户。



具体活动度的指标，可以根据业务的具体情况进行定义。人群划分好之后，就可以针对各分层人群进行召回承接。我们的重点是要召回当月新用户、低活用户、中活用户、流失用户。

高活用户：即使不采取召回，也会主动来；

流失用户（30 天没来过的用户）：召回难度最大。因为有可能用户已经卸载或者转移到其他 APP。从数据上看，流失用户的转化 ROI 也是最低的。

所以我们的召回策略就是要把召回阶段前置，在低活和中活这两个阶段就开始进行召回干预。因为这个阶段用户获取成本相对较低，ROI 比较高，让这部分用户不断触达召回，提升留存，把他们往高活上去迁移（ 当然这部的用户体验也要跟的上，让用户形成使用习惯和心智很重要 ）。

——系统架构——

明确了业务目标，接下来介绍下我们的系统的架构。

DataFunTalk

我们将从渠道能力、策略能力、算法能力、实时监测，数据等几个维度来介绍下 DSP 系统的架构实践与思考。

❶ 渠道能力

召回获客首先要有强大的渠道能力做支撑，这样才能获取和覆盖到足够多的我们想要的用户，本文只涉及到支持 RTB 竞价广告相关的渠道，市面上有很

多 ADX 比如百度 ADX、网易、广点通、tanx 等等，那么这么多渠道，我们也不可能全部接入，流量压力也是可想而知的。

首先要思考和评估我们要重点接入哪些渠道，哪些渠道的获客体量大，哪些渠道的成本低，每个渠道的用户分层是怎样的，我们需要对渠道有一个渠道质量的洞察。

从以下指标进行洞察：

媒体信息	下发流量uv	优酷覆盖uv	多媒体重合度	可触达占比	点击率	转化率

需要分析：

- 媒体下发流量中有多少是安装优酷的用户，用户分层占比；
- 多个媒体间的用户重合度，控制跨媒体曝光频次；
- 每日最多可触达多少优酷用户，并分析未被触达的原因；
- 媒体的点击率、转化率（对应优酷的换端率、播放率），预算分配等；
- 实现最佳的媒体组合进行投放；
- 优化网盟渠道，如广点通。

广点通渠道我们主要对接的是网盟，广点通网盟可能包含上万家的 APP 媒体资源位，每个媒体的效果和表现都不同。这加大了优化难度，且广点通的流量下发多是根据（ 竞价成功 PV/流量下发 PV ）的占比，比例越大，流量下发越多。这就带来一个问题，如果要获取精准用户，势必参与竞价的 PV 会减少，占比降低，最终导致流量下发降低，覆盖不到更多的用户，需要进行权衡。

❷ 策略能力

在 DSP 广告的投放过程中会涉及到很多的投放策略，下面介绍几个比较通用的投放策略。

1. 频次控制

这里说的频次控制包括：参与竞价频次控制、展现频次控制、点击频次控制、换端频次控制等几个维度。

一般说频次控制主要是说展现频次控制，优酷又增加了三个维度的频次控制。

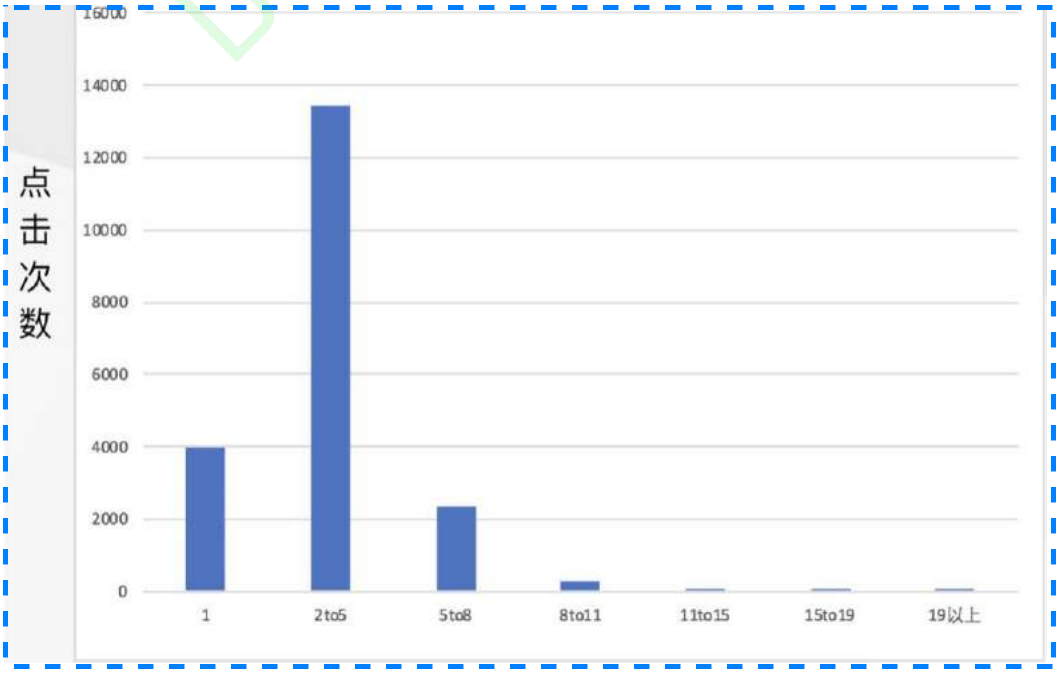
参与竞价频次控制： 这个是分媒体的，在有些媒体效果会很好，有些媒体则不需要，由于有些媒体参与竞价和展现之间的延迟时间比较长，导致一直没有展现，这个时候展现控制就失效了。

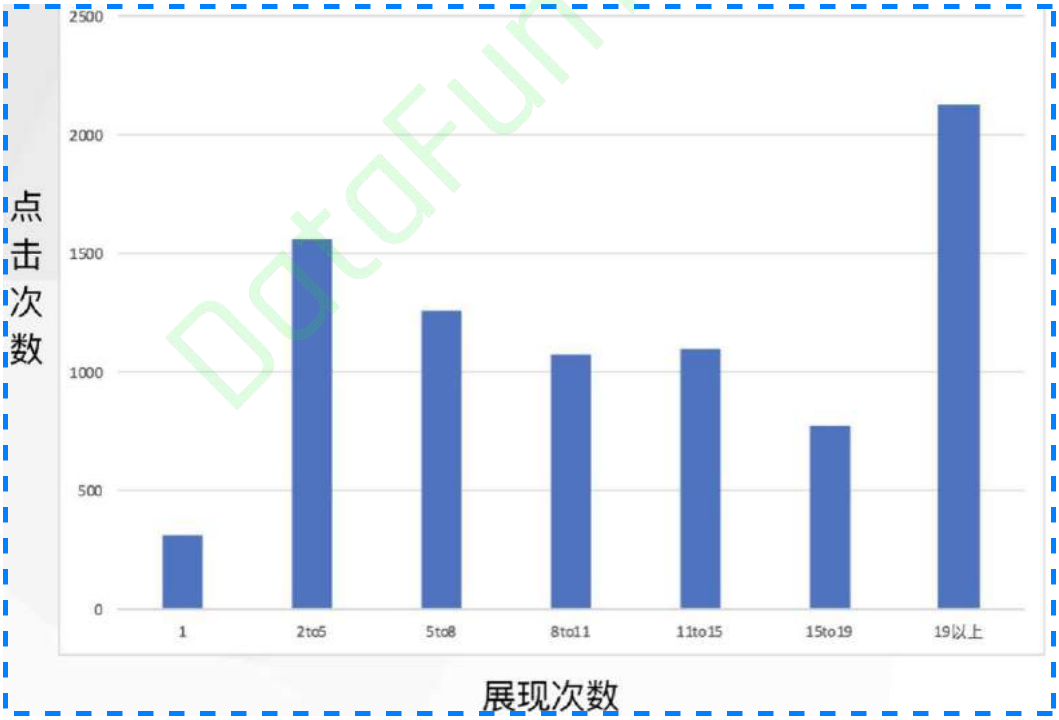
用户可以一直参与竞价，然后集中或者分散开去展现，这种情况展现控制就失效了，有可能一个用户要展现上百次都有可能。

点击频次控制： 对于当天点击过广告的用户，则不在购买，可以防止疲劳。

换端控制： 对于当天唤起过优酷的用户，不在继续购买，由于用户主动或者被动已经来过优酷了，就没必要在进行购买了。

下面举例说明下不同媒体频次控制的次数，不同媒体的效果差的也比较大，尤其是网盟渠道会和头部媒体渠道差别比较大。





2. 素材赛马策略

在投放素材的时候，一个广告计划可能会配置多套素材进行投放，如果没有赛马机制的话就会随机投放。

赛马机制主要解决的是把点击率高的素材给到尽可能多的曝光，点击率低的素材减少曝光或者不曝光。

这个非常类似推荐系统里面的两个经典问题：EE 问题（ 从探索到应用 ）和冷启动问题。

下面介绍一个简单的赛马逻辑实现，具体策略如下：

假定投放方案 A，下面有广告位 M，符合 M 条件的素材 X 套（X>=2）

素材 CTR=素材点击数/该素材展示数

冷启动过程：

第一，在 15min 时间范围内，广告位 M 符合投放方案 A 流量对于 X 套素材流量平均分配。

第二，计算上个 15min 内 X 套素材的 CTR 结果 [CTR1,CTR2,...,CTRx]，并进行排序，假定 CTR 最高的为素材 N。

第三，在下一个投放（15min）范围内，对于广告位 M 符合投放方案 A 竞价返回素材时：

- 对于素材 N 分配 80%的流量
- 对于剩下的 X-1 套素材，平均分配剩余 20%流量

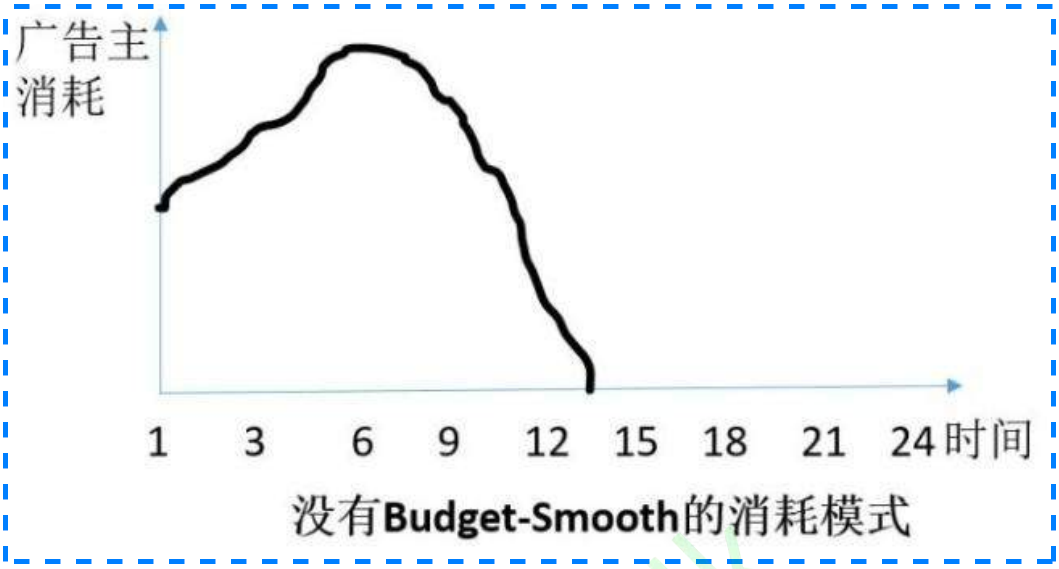
第四，投放中重复步骤 2 和步骤 3 的判断。

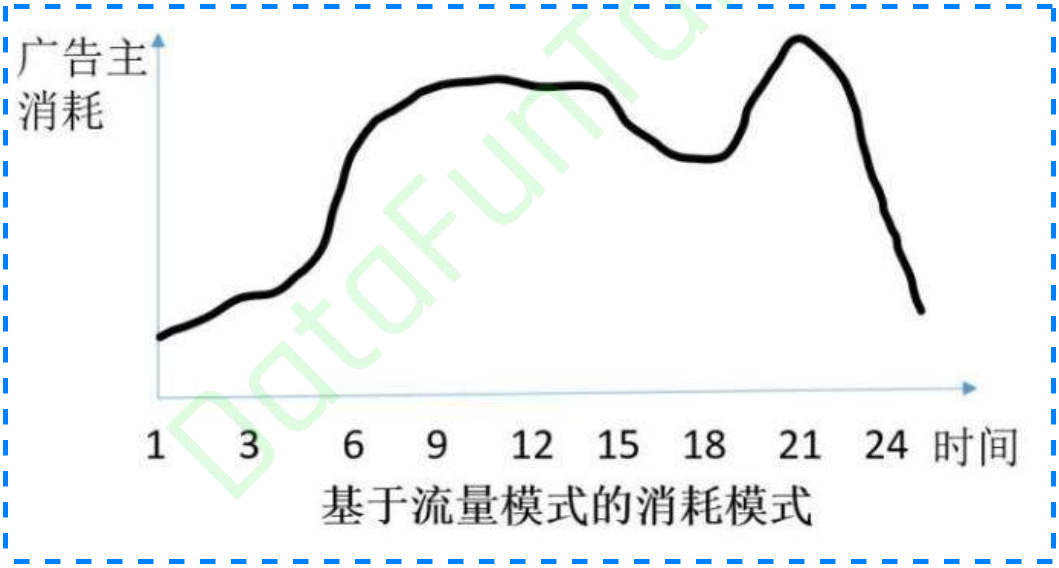
3. 平滑投放（budget smooth）

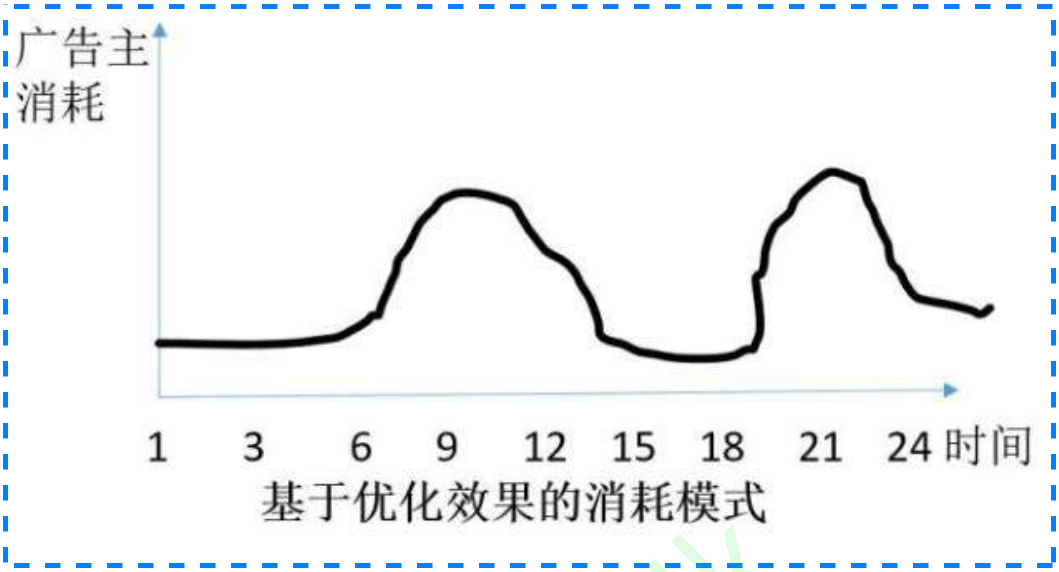
平滑投放是保证广告主在预算一定的情况下，能平滑的消耗预算，而不是在开始几分钟之内全部花完。

DataFunTalk

平滑投放可以让广告主触达更多的人，更多时间段的用户群，下面是几种消耗模式，借网络上的几张图：







之前，我们采用的是基于流量模式的平滑投放，问题是流量大时，分配更多的预算，但是转化 ROI 不一定好，所以目前我们采用的是第三种方式，基于效果的平滑投放，这里的效果主要指的是点击 CTR 效果，比如中午，刷手机比较多，效果比较好，那我们就多分配预算在中午这段时间进行投放。

4. 广告重定向策略（retargeting）

"重定向" 是广告领域的一个术语 ,又被称作 "访客找回广告","在营销广告"，"回头客广告" 等等，是帮助广告主有效获取目标受众的一种重要方式，比如在消费者购物的过程中，常常会因为这样或那样的原因，最终没有完成转化。重定向广告将商品展示到此类未完成转化的消费者面前，将其带回到相关网页。由于展示的商品往往也是消费者最想要的，所以重定向广告的转化率比一般广告要更高。

最近X-X天在优酷看过剧(第N集)（时间由新到旧筛选）	视频观看时长占比在(0,10%]	返回当前集（第N）视频信息
	视频观看时长占比在(10%,65%]	返回当前集（第N+1）视频信息
	视频观看时长占比在(65%,100%]	按照观看时间顺序寻找下个剧集内容

5. 图片视觉优化策略

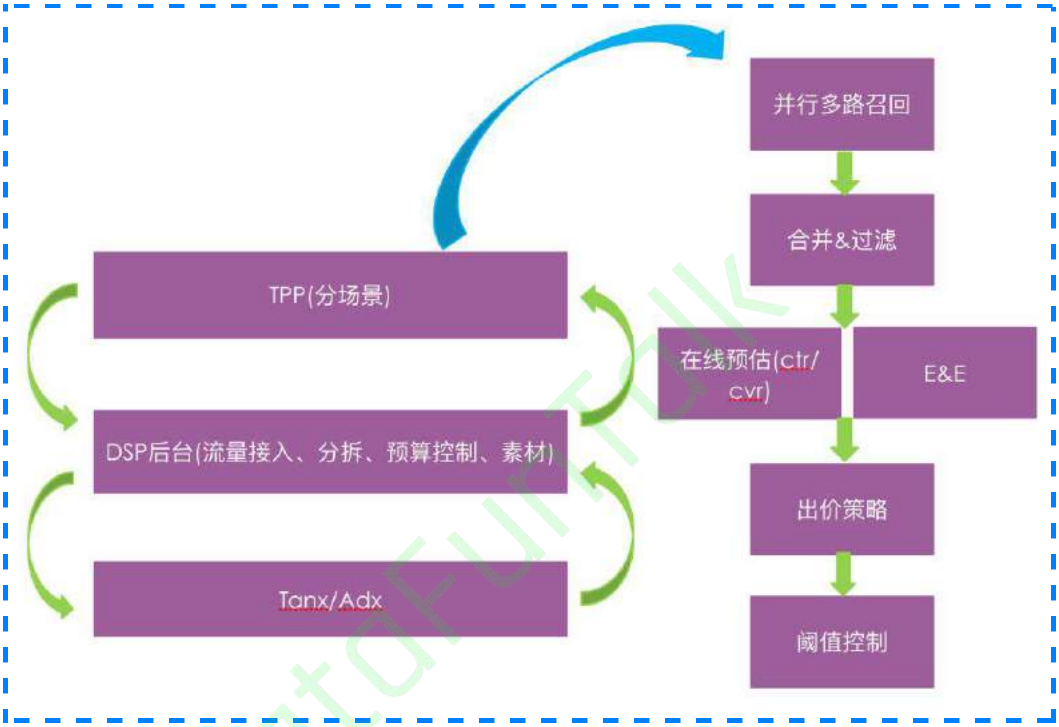
为了给用户更好的体验以及更好的视觉冲击力，提升用户的点击欲望，我们对投放的图片素材进行了二次加工，取得了不错的效果。

测试中发现不同渠道的视觉不同，不能统一设计，所以我们进行了分渠道对比视觉设计实验，比如在某 feed 流中。

通过工程合图能力，我们快速实现并上线了几十套模板，线上进行 AB 实验，测试点击率效果，选择优秀的模板长期投放。

❸ 算法能力

接入算法能力主要为了实现个性化的站外广告推荐，目前接入算的架构。

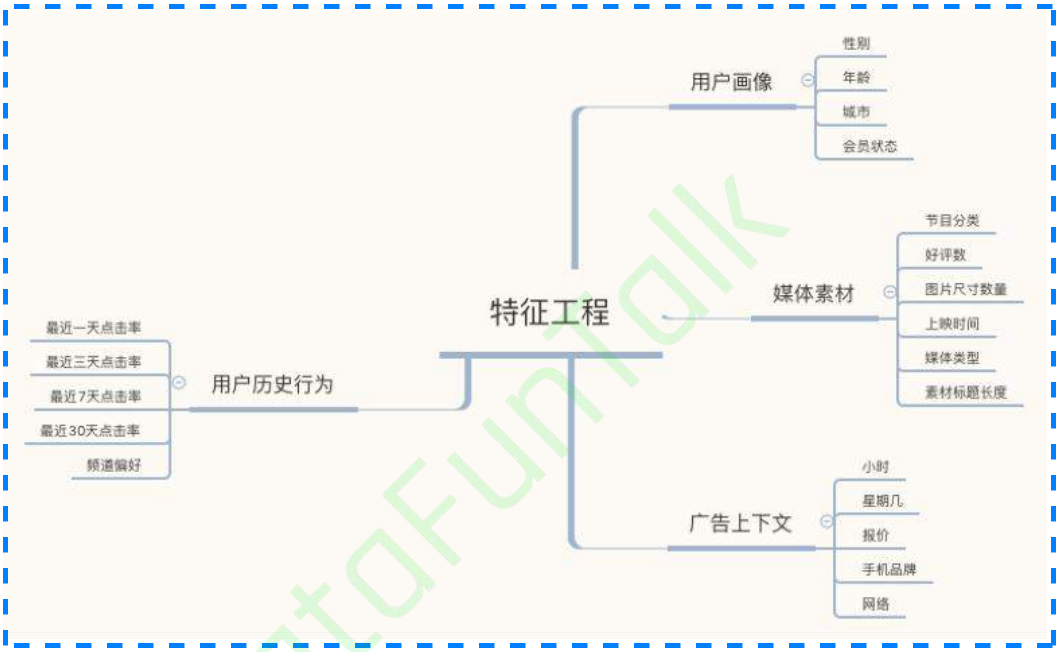


1. 并行多路召回

- 基于播放记录召回 :会根据用户的站内播放行为进行召回 ,按时间进行衰减；
- 基于热点节目召回： 比如最近新上的新热剧等；
- 基于用户偏好召回：比如明星偏好，剧集偏好等；
- 基于站内行为召回：比如收藏行为，点击浏览行为，搜索行为等。

2. 分渠道模型

由于我们接入了很多渠道，每个渠道的特征表现都不太一样，需要分渠道进行针对性的模型训练，简单的特征如下：

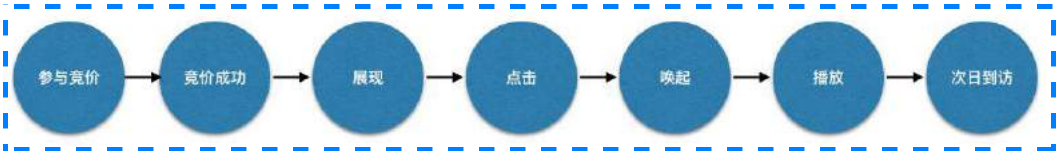


❹ 数据&监控能力

完善的数据分析系统对 DSP 的投放非常重要，通过数据分析，优化转化漏斗，降低成本。

1. 转化漏斗

首先需要定义业务的转化漏斗，见下图：



下面逐步说下这几个漏斗阶段我们可以干预的策略。

参与竞价

参与竞价的量，主要受限于用户的识别能力，如果根据设备信息可以准确识别到是优酷的用户，那么参与竞价的量可以大大提升。

竞价成功

竞价成功的量，主要有这几个方面的受限，第一个是出价，出价过低导致竞价失败率高，第二个是接口性能，目前 ADX 要求是 100ms 内返回，这就要求 dsp 需要不断的优化接口的响应时间，可以很明显的提升竞价成功的量。

展现

展现的量，比如标题长度，曝光的次数等等。另一个是媒体自身的原因了，可能媒体本身有些复杂的频控逻辑导致展现不成功。

点击

这个第一是和算法的推荐能力的准确性有关系，推荐的越准确越容易点击，另一个是和图片和标题的吸引程度以及视觉体验有关系，比如一个猎奇的短视频，可以大大激发用户的点击欲望，还有就是刚才说的视觉图片合图的优化。

唤起

换端率，主要受限于识别用户是否安装了优酷 APP，用户是否是误点击，或者不像打开观看，时机不对？或者还有个原因就是系统启动太慢，有广告，提前退出了。

播放

播放率，主要受限于播放页的性能和前贴广告，如果前贴广告太长，用户等不及就退出了，没有产生有效的播放，再有就是网络状态 4g or wifi 等等。

次日留存

主要和产品的用户体验有关系。

2. 全方位监控

广告系统的交互各个环节都需要进行详细的监控，才能做到有问题及时发现。

比如突然的消耗上升或者降低，点击率的下降，参与竞价量的下降等等，都需要进行详细的监控，避免资损的发生。

目前建立的监控体系指标有如下：

维度	分钟级监控项
广告计划	参与竞价、竞价成功、点击、展现、唤起、消耗 ...
媒体	参与竞价、竞价成功、点击、展现、唤起、消耗 ...
大盘	总花费、竞价总pv、竞价成功总pv、曝光pv、点击pv、换端uv、换端成本，cpm ...
其他	错误码、流量下发总量、rt区间分布、状态码..

——总结——

最后在总结下整个 dsp 的建设过程要考虑的点：

- 渠道建设，流量接入
- 投放策略沉淀
- 算法模型发挥推荐优势
- 完善的监控体系

当然本篇文章由于篇幅有限，还没有涉及到 DSP 的另一块核心，流量价值预估和出价算法，这块后续会有单独的篇幅去介绍，优酷的这套体系上线后在拉新和召回上发挥了重要的作用，为用户增长打下了坚实的基础。

参考链接：

1. https://zhuanlan.zhihu.com/p/52237643

2. http://www.swanlinux.net/2017/02/10/papers/real_time_bid_optimization/index.html

3. https://tech.meituan.com/2017/05/05/mt-dsp.html

4. https://zhuanlan.zhihu.com/p/28406017

5. https://help.aliyun.com/document_detail/59480.html

6. https://36kr.com/p/5189571

阿里妈妈点击率预估中的长期兴趣建模

嘉宾：卞维杰　编辑：林志宏

导读：CTR 预估作为广告和推荐领域的基础技术，在用户行为较丰富的场景，一般通过建模用户历史行为序列来挖掘用户兴趣。历史行为的时间跨度越长，用户兴趣偏好更稳定，因此当我们希望刻画用户长期兴趣时，需要引入更多的行为数据，这给在线服务引擎和算法模型提出了一定挑战。我们通过算法、工程 Co-Design 的视角设计了一套增量计算服务框架，解决了长期兴趣建模带来的存储和在线服务压力，并在此基础上提出了两种 Memory Network，分别从时间和多峰兴趣两个角度去建模用户长期兴趣，从而提升了模型预估准度，带来定向广告业务收益的上升。

本文将从以下三部分介绍阿里妈妈关于点击率预估中的长期兴趣建模的工作实践。

- 相关背景
- 主要工作
- 实践总结

——相关背景——

❶ 业务背景

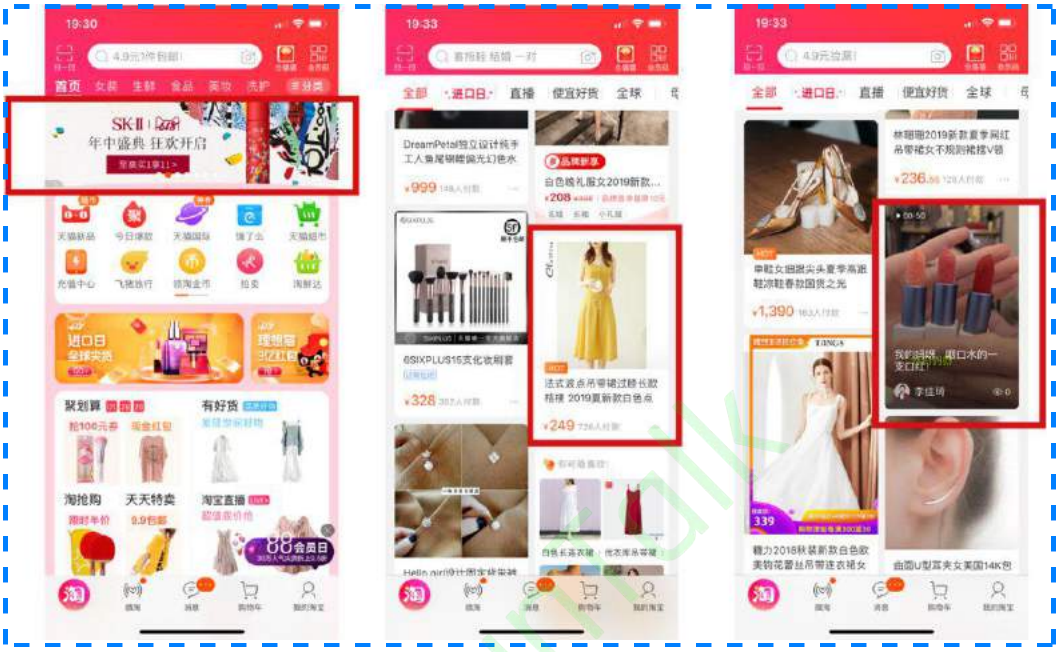


图 1

阿里妈妈定向技术团队主要负责手淘客户端大部分的定向广告，广告的业务形态包括：焦点图广告、单品广告卡片以及诸如视频的内容化广告等，手淘场景主要有：首焦、猜你喜欢以及例如物流详情等购后链路。广告推送模式与推荐相似，就是进行用户的个性化点击率预估，然后结合广告主出价，进行 ECPM 排序，取 top k 进行展示，k 值根据广告位的数量而定。

❷ 技术现状

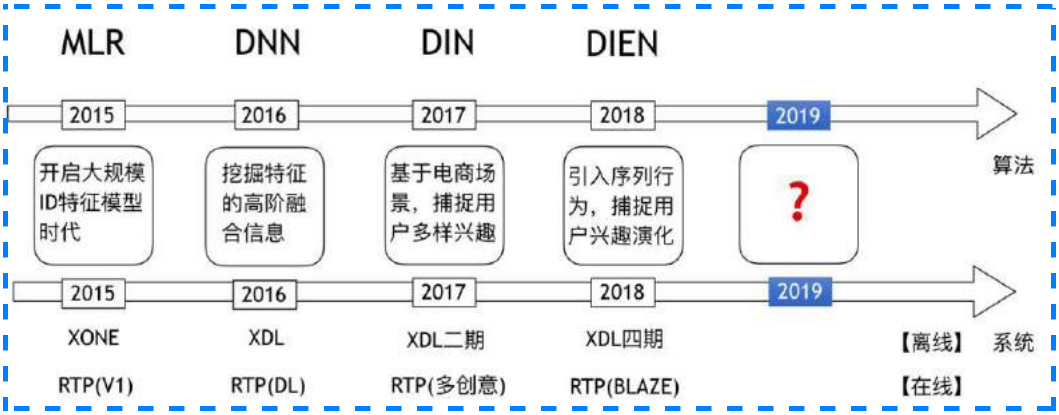


图 2

阿里妈妈定向技术团队技术迭代的路线如图 2 所示。在 2015 年的时候从 LR 升级到了分片线性模型 MLR，开启了大规模 ID 特征模型时代；2016 年引入 DNN 模型，利用深度学习挖掘特征的高阶融合信息；2017 年提出了 DIN 模型，通过引入序列信息，基于电商场景，捕捉用户的多峰兴趣；2018 年创造出了 DIEN 模型，通过 GRU 时序结构，引入序列行为，捕捉用户兴趣的演化。每一次技术创新的背后都是在挖掘数据的价值。与此同时，每次技术创新的背后都有系统服务升级的支持，训练引擎从 XONE 升级到 XDL 一期、二期、四期，在线服务引擎也从 RTP 的 1.0 版本逐步升级到高性能 RTP 的 BLAZE 版本。可以总结出整体的技术发展思路是：**数据价值的挖掘推动算法模型的创新，而算法模型的创新又推动了系统服务的升级。**按照这样的思路，2019 年进行的技术迭代就是**用户长期兴趣的挖掘。**

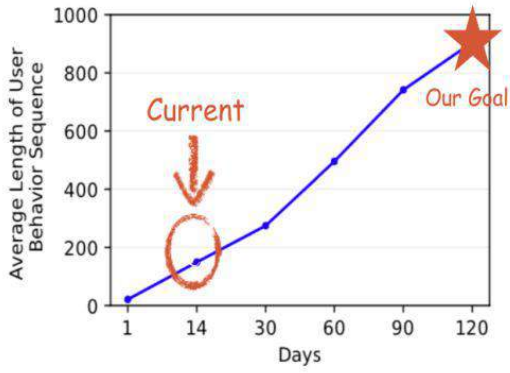


图 3-a

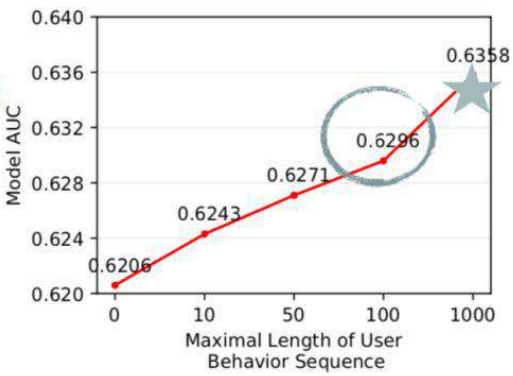


图 3-b

在谈长期兴趣挖掘的价值之前，得先看看目前团队建模能力的现状，首先 DIN 模型与 DIEN 模型都通过用户的历史行为挖掘用户兴趣，其中 DIN 模型对用户的原始行为序列进行 attention 操作，获取用户相关的兴趣，进行用户的点击率预估，而 DIEN 模型是通过 GRU 对用户原始行为序列进行时序的建模，但是由于一定的系统瓶颈，可以处理的最大历史行为长度为 100。从图 3-a 可以看出历史行为长度 100 仅对应用户 14 天的行为，当用户行为窗口增加到 120 天时，用户历史行为长度爆炸到了 1000。为了看出长期历史行为的价值，我们做了简单的实验来评估不同长度用户行为引入 CTR 预估模型后的离线表现，我们采用 DNN 模型，把行为序列特征（包括用户点击的 item 以及 item 的属性特征等）进行简单的 embedding sum 之后输入模型，如图 3-b 所示，利用 1000 长度用户行为信息相比 100 长度，能够带来 0.6% GAUC 的提升。这里采用的 GAUC 是以用户粒度为 group 计算的 AUC，然后根据用户的 frequency 进行 weight sum。其中 0.6%离线提升对在线业务有重大意义，由此可见长期历史行为具有显著的利用价值。

❸ 系统瓶颈

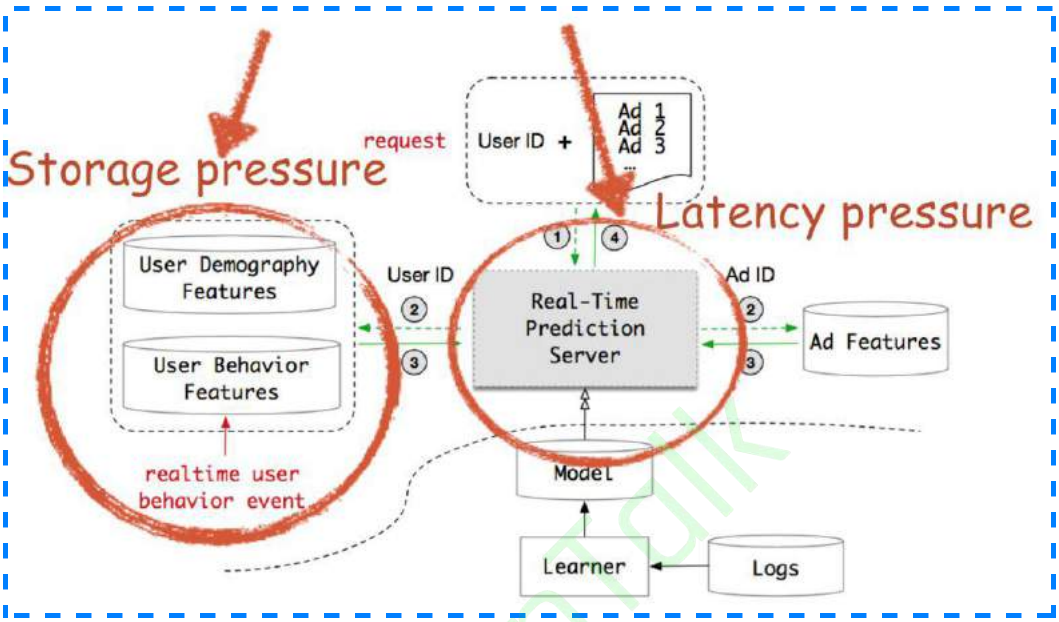


图 4

图 4 展示的是在线 RTP 引擎，当一个 request 过来的时候，会打分大概 400 个广告，我们根据 Ad ID 去 Ad Feature 这个索引库里面提取广告特征，根据 User ID 去提取 User 的 Demography 特征以及行为序列特征，然后送入模型进行打分，排序后将广告返回给用户。点击率预估与大部分工业应用一样需要一个实时的响应。在阿里这个电商平台里，点击率预估系统不仅面临高并发的请求，还需要在一个极短的时间内进行响应。我们的 DIN 和 DIEN 模型由于原始序列和 Target 均有显式交互，因而无法解耦计算，当行为序列长度为 100 时，DIN 和 DIEN 现有的复杂度逼近计算极限，由于计算资源的约束，在现有的模型下想把行为序列往 1000 扩展是不可能的。另外一方面，用户的行为特征是存储在线上服务引擎，阿里有亿级的用户，

如果每个用户存储的行为序列特征由 100 扩展到 1000 ,将带来极大的存储消耗和模型服务 IO 问题。因此我们面临着延迟约束和存储限制的系统瓶颈。

为了解决以上的系统瓶颈，必须从一个 high level 的角度进行思考，我们通过 Co-Design 算法&系统的突破，得出以下三个突破点：

- 计算解耦；
- 存储压缩；
- 增量复用。

——主要工作——

❶ 整体设计

围绕着计算解耦、存储压缩、增量复用这三点我们进行了如下的整体设计。

首先，在模型的选择方面，我们选择了 Memory Network 来进行我们基础模型的构建，Memory Network 有以下的几点好处：

1. 自组织的用户兴趣聚合，因为它可以通过 content-based address 的方式将相似的宝贝映射到一个相同的 Memory 中;

2. 有限的 Memory Slot 换取无限的行为历史，只需少数几个 Memory Slot 就可表达上千个行为的信息;

3. 实时动态 Memory 向量更新，避免重复 Memory 计算。

围绕这个模型能力，我们进行以下的系统设计：

1. 异步计算模块，实现兴趣向量独立更新；

2. 兴趣向量的存储，不仅服务在线计算，而且辅助兴趣向量增量更新。

我们将整体的设计细节总结成如下两篇学术论文：

1. SIGIR 2019 : Lifelong Sequential Modeling with Personalized Memorization for User Response Prediction；

2. KDD 2019 :Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction

❷ 系统改进

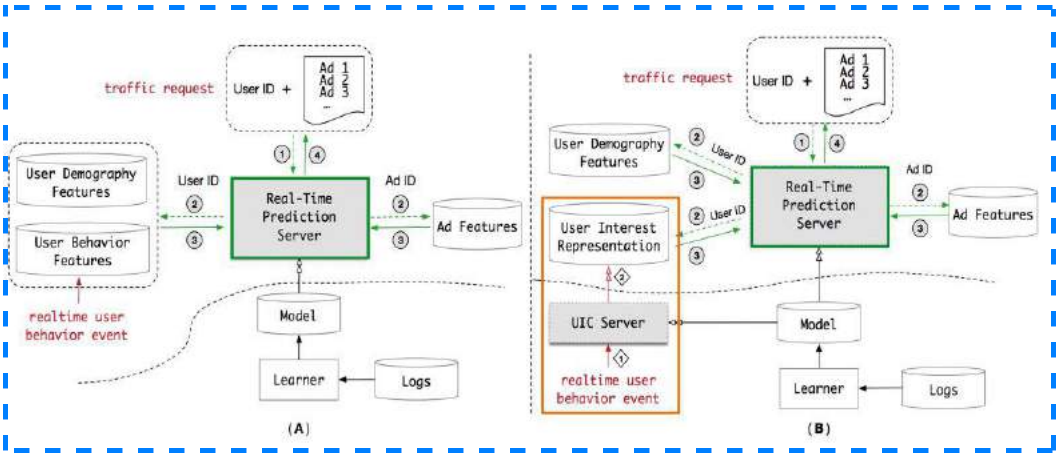


图 5

在原有的 RTP 系统（左图）的基础上我们引入了 UIC 模块（右图），即用户兴趣中心，它能实现我们想要的三个能力：异步计算、兴趣向量的存储以及增量更新计算。在原有 RTP 系统流程的基础上，新系统把取用户行为特征改成取用户兴趣向量，直接对兴趣向量进行 attention 操作，辅助点击率预估，同时当有实时行为进来的时候，UIC 会直接增量更新用户兴趣向量。

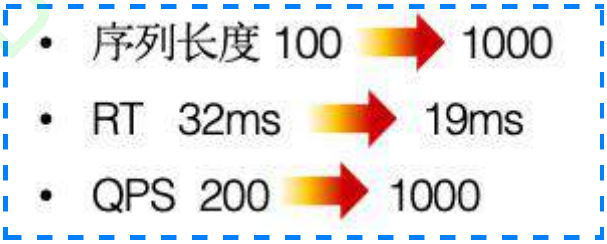


图 6

对比新老系统，同时采用 DNN+ 行为特征 embedding sum 的结构，性能的提升如图 6 所示。

❸ 算法创新

在介绍算法创新之前，首先介绍记忆网络。记忆网络主要有 facebook 的 Memory Network（和 attention 模型类似）以及 Google 的 Neural Turing Machines：NTM（它引入的是 external memory，显式对行为序列进行存储，帮助建模长期依赖）两种。NTM 相比 LSTM 和 GRU 更适合建模长期序列，后两者随着序列的增长存在梯度弥散等问题。记忆网络的

优势包括：适合建模长期依赖；Memory 向量的存储避免直接存储长期行为序列，缓解存储压力；Memory 的读操作相比 GRU+Attention 减小了计算压力。

NTM 版本的 Memory Network 存在一个 Controller 模块，控制着 Memory 的读写。下图为一个电商场景下的 NTM 示意图，假设一个用户浏览了鞋、包和杯子，Controller 会在发生行为时产生一个写头来实时更新 Memory，当物品相似的时候会写到一个相同的 Memory slot（content-based addres 机制），当一个请求过来的时候会产生一个读头来获取兴趣向量，辅助线上 ctr 预估。

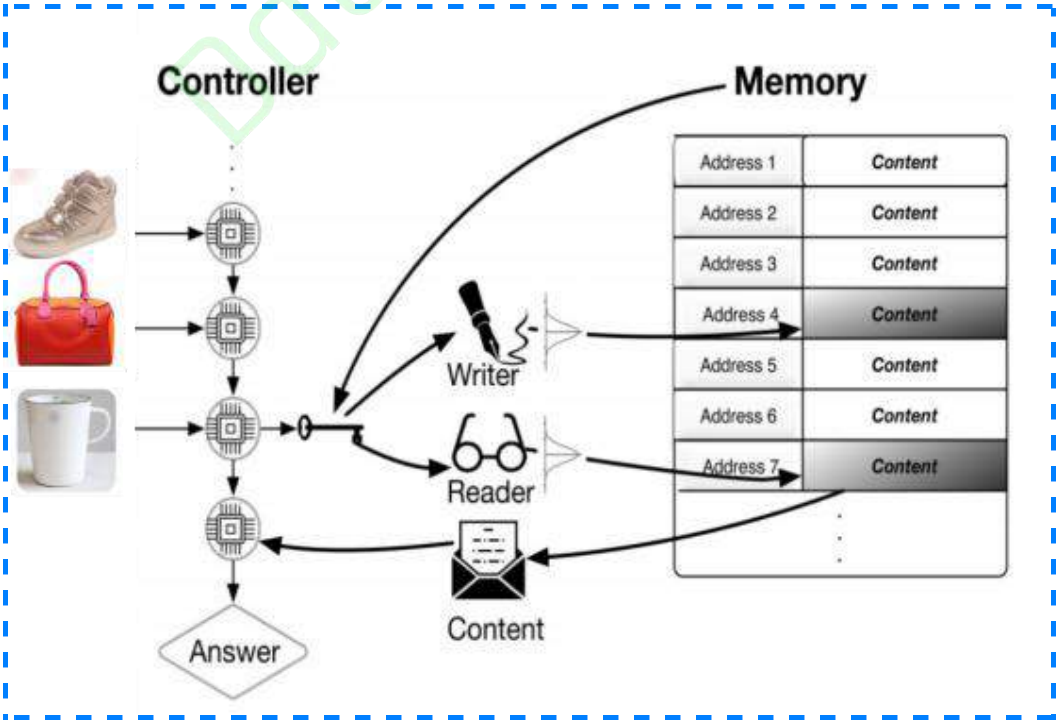


图 7

在 Memory Network 的基础上，我们进行了算法创新设计，发表了两篇论文。分别从不同的视角去构建用户兴趣向量。其中 MIMN（Multi-channel user Interest Memory Network）是从多峰兴趣角度进行记忆模块构建，因为在电商环境用户的兴趣广泛。而 HPMN（Hierarchical Periodic Memory Network）则从时间维度进行记忆模块构建，因为不同的商品存在不同的时间属性。

首先介绍一下 MIMN 的设计，它借鉴 NTM 的思路，采用 memory-based model 建模，在 GRU 时序模型的基础上引入 external memory 向量，显式存储行为信息。我们提出了两个创新点：利用 Memory utilization regularization 进行信息记录最大化以及使用 Memory induction unit 进行兴趣演化和高阶兴趣提取的建模。模型概览图如图 8 所示，我们在整个应用中引入了时序模块，它的基础是 NTM 加 Memory induction unit，在训练的时候会通过 target 对时序模块产生兴趣向量，与其他的静态特征一同输入 DNN 网络进行点击率预估模型训练，当训练完成之后，会进行分开部署，右边灰色部分会部署到线上实时 RTP 服务引擎，左边粉红色部分会部署到 UIC server 里面。

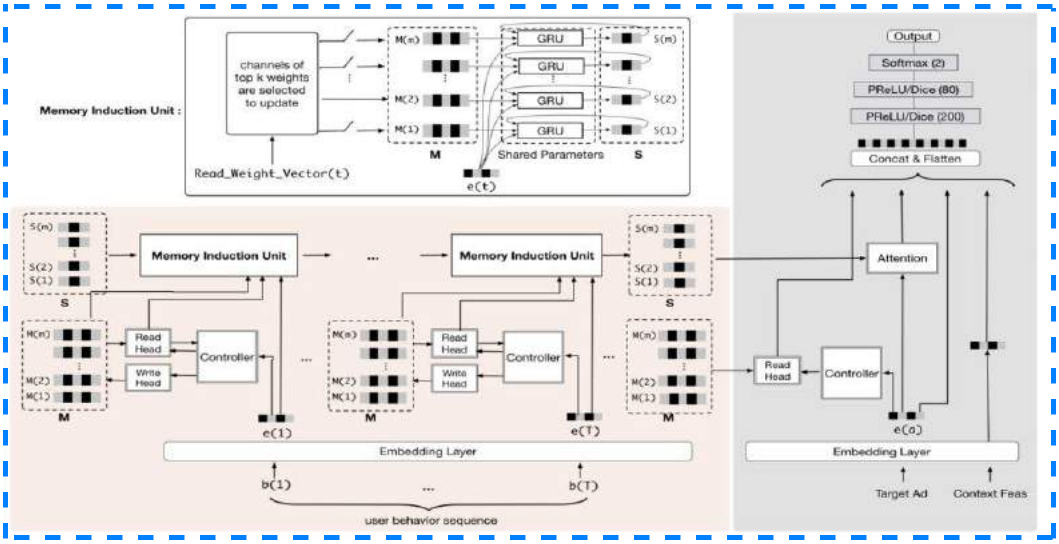
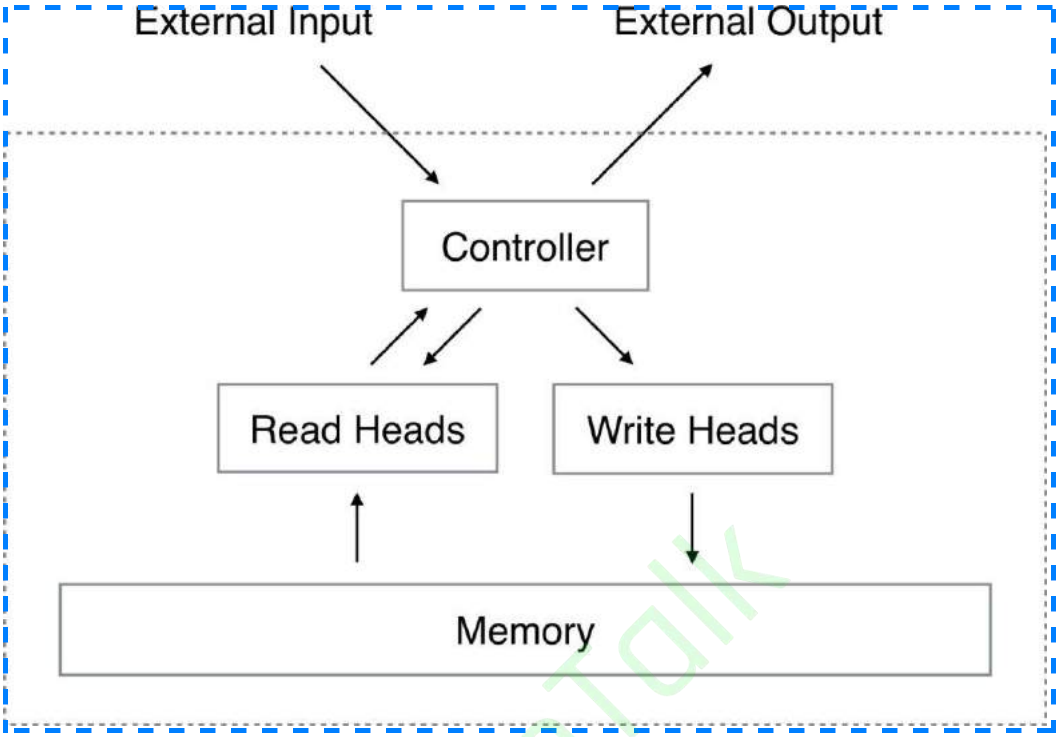


图 8

关于 NTM 模型细节方面，它包括几个关键部分：Controller，读写头（一般集成在 Controller 里）以及 Memory。



DataFunTalk

图 9

当一个新的输入进来的时候 Controller 会产生一个 Key Vector，它可以与 Memory 进行交互得到读权重向量，交互过程与注意机制相似，但是有一点与注意力机制不同，即 attention 是在做点积操作，而 NTM 中是在计算余弦相似度，它是为了避免 memory 之间 scale 具有差距，产生马太效应。例如一个 memory 的 scale 很大，在点积操作下产生的 score 也很大，以致于它长期具有一个很大的 weight。当我们得到每个 memory 的 read weight 之后，我们对每个 memory 进行 weighted sum 可以得到响应向量。详细计算公式如下。

$$\mathbf{w}_t^r(i) = \frac{\exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right)\right)}{\sum_j^m \exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(j)\right)\right)}, \quad for \ i = 1, 2, \ldots m$$

$$K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right) = \frac{\mathbf{k}_t^T \mathbf{M}_t(i)}{\| \mathbf{k}_t \| \| \mathbf{M}_t(i) \|},$$

$$\mathbf{r}_t = \sum_i^m w_t^r(i) \mathbf{M}_t(i).$$

Memory 更新方面，write weight 计算等价于 read weight，不同的地方在于 Controller 会产生 erase 向量和 add 向量，这两个向量会分别与

DataFunTalk

write weight 相乘得到 erase 矩阵和 add 矩阵 ,Memory 更新公式如下，减去想擦除的信息加上想增加的信息就得到新的 Memory。

$$\mathbf{M_t} = (1 - \mathbf{E_t}) \odot \mathbf{M_{t-1}} + \mathbf{A_t}$$

其中 $\mathbf{E_t} = \mathbf{w}_t^w \otimes \mathbf{e}_t$, $\mathbf{A_t} = \mathbf{w}_t^w \otimes \mathbf{a}_t$ 。

为了有效提高 memory 利用率，在 NTM 的基础上，我们加入了 Memory Utilization regularization 的考虑，希望约束不同 memory slot 中的写入方差。NTM 使用基于内容的寻址方式，会导致部分的 slot 写入利用率很低（ 定义为：累计的写入权重 ），如图 10 所示，绿色的柱条是亚马逊数据集使用原始的 NTM 得到的各 slot 的利用率，可以看出最后两个 slot 利用率都很低 ,线上使用 36 个 slot 的时候部分 slot 利用率很低的现象更为严重。

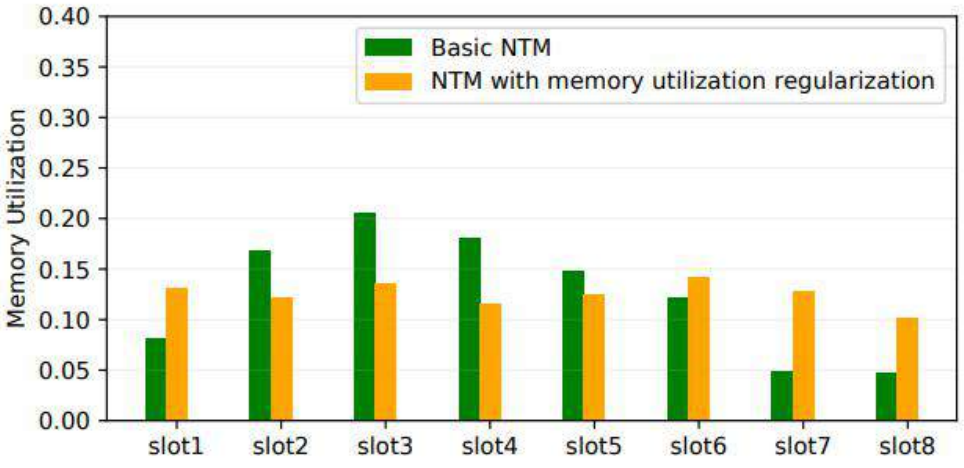


图 10

图 10 中橙色的柱条为经过 Memory Utilization regularization 之后得到的 memory slot。

为了平衡各 slot 的使用率，我们对 write weight 进行了 re-weight 的操作，即累积之前的 write weight 得到 gt 向量，经过转换后得到转移概率矩阵 Pt 再乘以 write weight 向量得到改写后的写入权重。改写权重的计算方式如下：

$$\mathbf{g}_t = \sum_{c=1}^{t} \mathbf{w}_c^{\tilde{w}}$$

$$P_t = softmax(W_g \mathbf{g}_t)$$

$$\mathbf{w}_t^{\tilde{w}} = \mathbf{w}_t^{w} P_t.$$

$\mathbf{w}_t^{w}$ 是原始写入权重，计算方式参考 NTM。$\mathbf{w}_t^{\tilde{w}}$ 为改写后的写入权重。同时 W_g 则利用正则 loss 来学习：

$$\mathbf{w}^{\tilde{w}} = \sum_{t=1}^{T} \mathbf{w}_t^{\tilde{w}},$$

$$\mathbf{L}_{reg} = \lambda \sum_{i=1}^{m} \left(\mathbf{w}^{\tilde{w}}(i) - \frac{1}{m} \sum_{i=1}^{m} \mathbf{w}^{\tilde{w}}(i)\right)^2,$$

M 表示 slot 的数量，L_{reg} 能够减少不同 memory slot 利用率的方差。

NTM 算法 memory 通常用来存储原始数据信息，遗失了对于更高阶的信息的捕捉。为了能更好捕捉用户兴趣，MIMN 设计了 Memory Induction Unit（MIU）来捕捉用户兴趣演化。

DIEN 模型做兴趣演化的时候有这样一个考量，在电商场景下用户的行为序列可能是跳跃的，比如用户在这个时间段全部看电子产品，另一个时间段全部看衣服，其间并没有时序关联关系，DIEN 的设计思路是通过对 Target 进行显示的 Attention 操作，把 attention weight 比较高的部分挑选出来，相当于挑选出比较相似的序列宝贝，然后经过 GRU 网络进行演化。

在我们的 MIMN 中，已经把 target attention 解耦开了，可以发现 NTM 的余弦相似计算也是一种相似关联操作，变相把相似的宝贝聚集在一个相同的 memory slot 里面。因而我们可以将相同 memory 里面的 items 分配到相同的 channel 里面进行演化。

DataFunTalk

图 11

如上图所示，衣服都分配到了 channel 1，而母婴产品分配到了 channel 2 与 channel 3，在此基础上我们又把 channel 所对应的 memory 信息送入 GRU 模，如下图和公式所示，进行兴趣演化和高阶兴趣的挖掘。

$$\mathbf{S}_t(i) = \mathrm{GRU}(\mathbf{S}_{t-1}(i), \mathbf{M}_t(i), e_t),$$

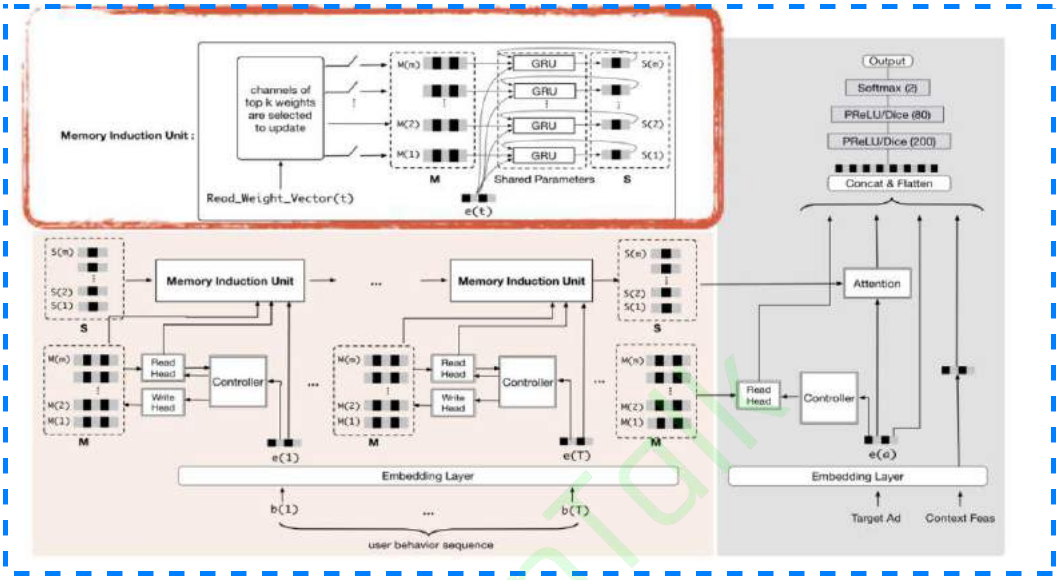


图 12

MIMN 的在线实现如图 13 所示，时序模块存放在 UIC 系统中，当有个新的用户行为时，会通过 Controller 实时更新用户的 memory 向量，并通过 memory induction unit 来更新 channel 演化向量，然后将这两组向量存放在 TAIR 以供线上实时请求系统应用。

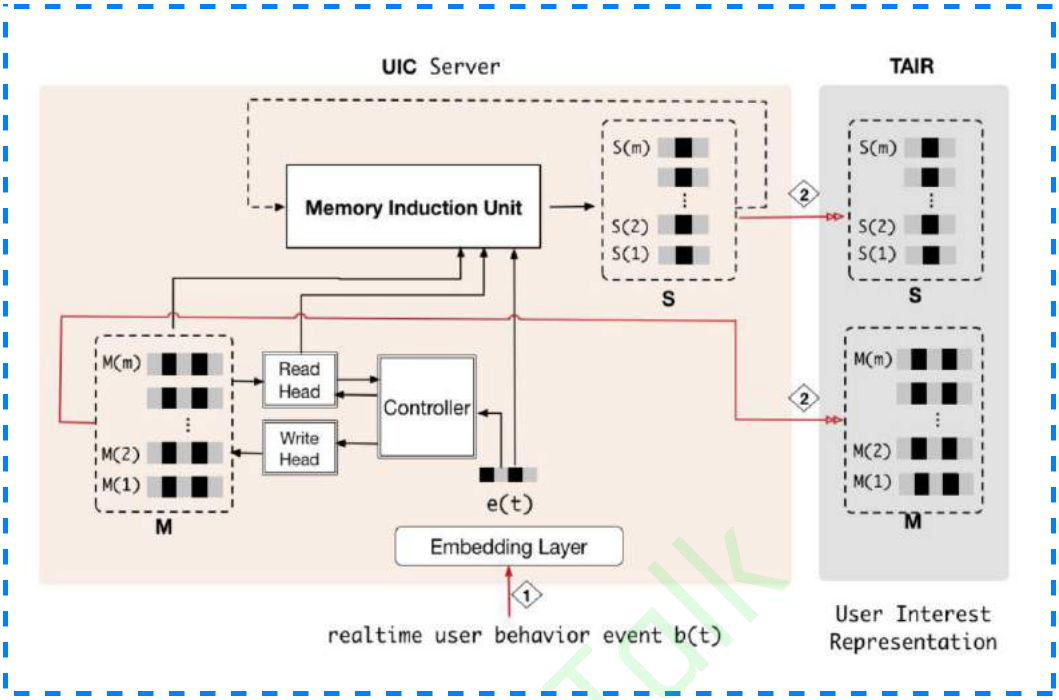


图 13

接下来简单介绍一下 HPMN。HPMN 是从时间维度进行 memory 构建。它的设计理念认为用户存在长、中、短各变化趋势的兴趣。它的设计思路借鉴层级 RNN ,采用周期向上归纳的设计 ,分层刻画不同时间跨度的兴趣偏好，与此同时，为了防止不同层之间具有相似的表达，它也加入协方差约束（ 兴趣多样化约束 ）。结构如图 14 所示，比较的简单明了，采用幂次固定周期进行归纳总结的方式。

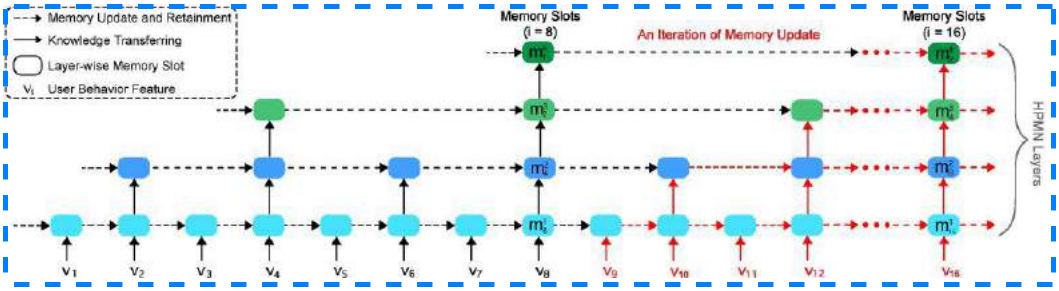


图 14

Memory 的更新公式如下：

$$m_i^j = \begin{cases} g^j\left(m_i^{j-1}, m_{i-1}^j\right) & 若\ i \bmod t^j = 0, \\ m_{i-1}^j & 其他, \end{cases}$$

它的兴趣向量读取为 :当一个 request 来的时候 ,有一个目标商品的特征 v ；然后计算记忆槽的注意力权重

$$w^j = softmax(E(m^j, v))$$

最后得到用户在第 i 步的表示 $r = \sum_j^p w^j \cdot m^j$ ，把这个结果返回来帮助点击率预估。

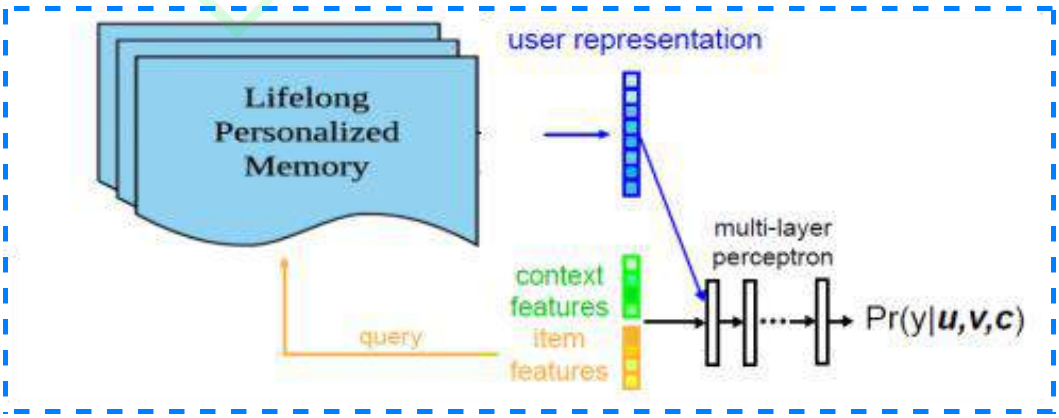


图 15

它采用了多样性正则策略，约束不同的记忆槽存储用户的不同时间跨度的兴趣，如以下公式所示：

$$C = \frac{1}{p}(M - \overline{M})(M - \overline{M})^\top, \text{where } M = [m^1, ..., m^j, ..., m^D]^\top$$

$$\mathcal{L}_{\text{cov}} = \frac{1}{2}(\|C\|_F^2 - \|diag(C)\|_2^2)$$

——实践总结——

❶ 效果对比

我们采用了亚马逊以及淘宝的公开数据集来进行实验，对比的模型有：Embedding & MLP、DIN、GRU4REC、ARNN、RUN、DIEN，从表 1 可以看到在这两个数据集上的效果 MIMN 模型都优于对照的模型。

Model	Taobao (mean ± std)	Amazon (mean ± std)
Embedding&MLP	0.8709 ± 0.00184	0.7367 ± 0.00043
DIN	0.8833 ± 0.00220	0.7419 ± 0.00049
GRU4REC	0.9006 ± 0.00094	0.7411 ± 0.00185
ARNN	0.9066 ± 0.00420	0.7420 ± 0.00029
RUM	0.9018 ± 0.00253	0.7428 ± 0.00041
DIEN	0.9081 ± 0.00221	0.7481 ± 0.00102
MIMN	**0.9179 ± 0.00325**	**0.7593 ± 0.00150**

表 1

关于线上效果，如下表所示，在 GAUC 的评价指标下，我们的模型相较于 DIEN 能提高 1 个百分点的效果。表 2 中倒数第二行是在参数不同步的情况下的模型效果，可以看出 UIC server 和 RTP server 模型参数存在 1 天的差异时模型效果未下降；最后一行是当训练数据中有大促数据时（由于大促用户行为丰富，同样长度的行为序列涵盖的时间跨度较短），模型效果略有下降，大概下降了两个千分点。

Model	AUC
DIEN	0.6541
MIMN	0.6644
MIMN under out-synch setting (within one day)	0.6644
MIMN trained with big-sale data	0.6627

表 2

HPMN 模型的效果如表 3 所示，实验做了两个对比，上面的 s 指的是一个短序列，下面的 T 指的是长期行为序列，s 只做了 group 2 的实验对比，说明当我们把行为序列扩长，会带来明显的增益，另外 HPMN 在长期行为序列下对比其他模型也是有明显的效果提升。

Model Group	Model	Len.	AUC			Log-loss		
			Amazon	Taobao	XLong	Amazon	Taobao	XLong
Group 2	GRU4Rec	s	0.7669	0.8431	0.8716	0.5650	0.4867	0.4583
	Caser	s	0.7509	0.8260	0.8467	0.5795	0.5094	0.4955
	DIEN	s	0.7725	0.8914	0.8725	0.5604	0.4184	0.4515
	RUM	s	0.7434	0.8327	0.8512	0.5819	0.5400	0.4931
Group 1	DNN	T	0.7546	0.7460	0.8152	0.6869	0.5681	0.5365
	SVD++	T	0.7155	0.8371	0.8008	0.6216	0.8371	1.7054
Group 2	GRU4Rec	T	0.7760 ↑	0.8471 ↑	0.8702 ↓	0.5569 ↑	0.4827 ↑	0.4630 ↓
	Caser	T	0.7582 ↑	0.8745 ↑	0.8390 ↓	0.5704 ↑	0.4550 ↑	0.5050 ↓
	DIEN	T	0.7770 ↑	0.8934 ↑	0.8716 ↓	0.5564 ↑	0.4155 ↑	0.4559 ↓
	RUM	T	0.7464 ↑	0.8370 ↑	0.8649 ↑	0.6301 ↓	0.4966 ↑	0.4620 ↑
Group 3	LSTM	T	0.7765	0.8681	0.8686	0.5612	0.4603	0.4570
	SHAN	T	0.7763	0.8828	0.8369	0.5595	0.4318	0.5000
	HPMN	T	**0.7809***	**0.9240***	**0.8929***	**0.5535***	**0.3487***	**0.4150***

表 3

另外还可以对容量进行分析。MIMN 中 memory 的数量可以任意定义，但是它的效果并不是随着 memory 的增长线性增长的。在淘宝数据集和亚马逊数据集上，最优的 memory slot 数量分别是 8 个和 4 个，因为我们能处理的行为序列长度和最优的 slot 个数是存在关联关系的，当行为序列较短的时候，例如亚马逊数据集，往上扩 slot，并不能带来效果提升。

Model	Taobao (mean ± std)	Amazon (mean ± std)
MIMN 4 slot	0.9046 ± 0.00135	**0.7522 ± 0.00231**
MIMN 6 slot	0.9052 ± 0.00202	0.7503 ± 0.00120
MIMN 8 slot	**0.9070 ± 0.00186**	0.7486 ± 0.00071

表 4

HPMN 也会存在这样的边际效益，而且在逐渐扩展 slot 的过程中可能还会导致效果的下跌。

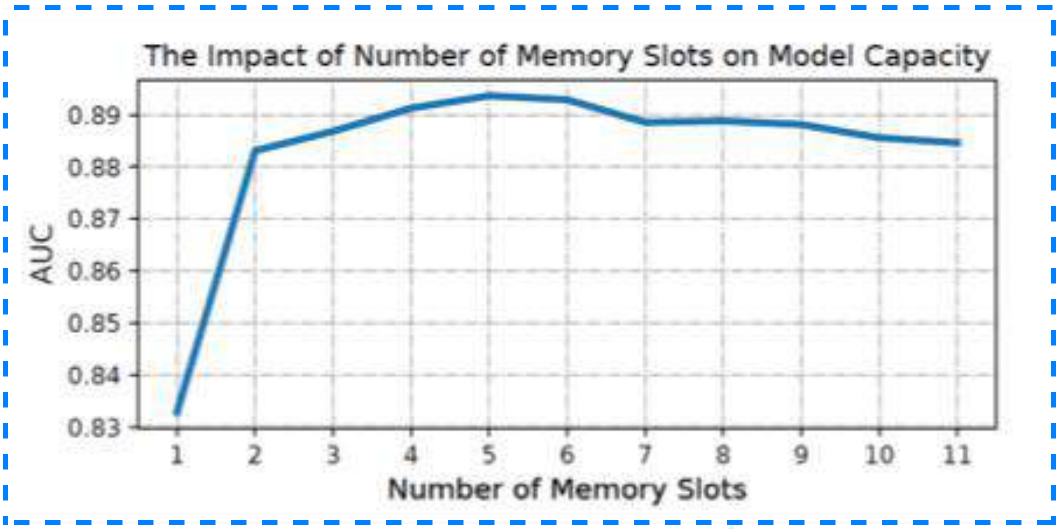


图 16

线上收益方面，在猜你喜欢的场景带来了 7.5%的 CTR 提升，而 RPM 有 6%的提升。

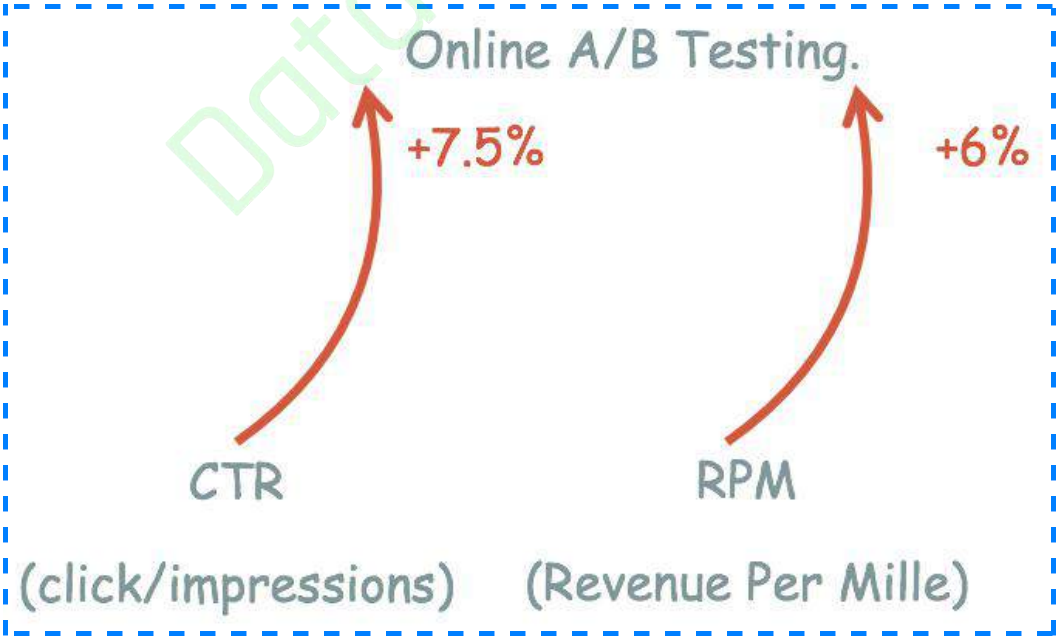


图 17

除此之外，我们模型上线的过程遇到了很多困难并将其总结成经验。

关于整个实践细节 UIC server 和 RTPserver 模型参数同步问题

MIMN 算法是由两部分组成，一部分是用户兴趣提取，一部分是 CTR 预估，在线计算会涉及到 UIC 和 RTP 两组服务。因此会存在模型参数同步的问题，在表三中我们做了相关实验，证明 UIC 和 RTP server 模型参数相差一天，对于模型离线效果无影响。我们系统是采用增量训练的方式，每小时就会更新模型参数，会大幅降低参数不一致风险。

大促数据影响

电商场景经常存在大促，比如最出名的双十一促销。在这样的情况下，数据的分布和用户行为与平时存在很大的差异，而且大促当天丰富的用户行为会影响对用户长期兴趣的刻画。我们对比了引入促销时期用户行为刻画用户兴趣，离线效果有 0.2%的下降。对此我们对大促的行为通过降采样的方式进行一定的丢弃。

初始化策略

尽管 UIC 可以增量计算用户行为，但是累计用户长时间行为会耗时严重。于是我们设置了一个初始化机制。将模型学习到的用户 120 天行为的兴趣表达导出，放入 TAIR 作为初始化。

回滚机制

为了防止线上出现故障，例如行为数据被污染，带来的在线效果问题。我们设置了断点保存机制，定时将的用户兴趣状态导出到离线存储。出现故障后，加载最近的离线存储。

❷ 总结思考

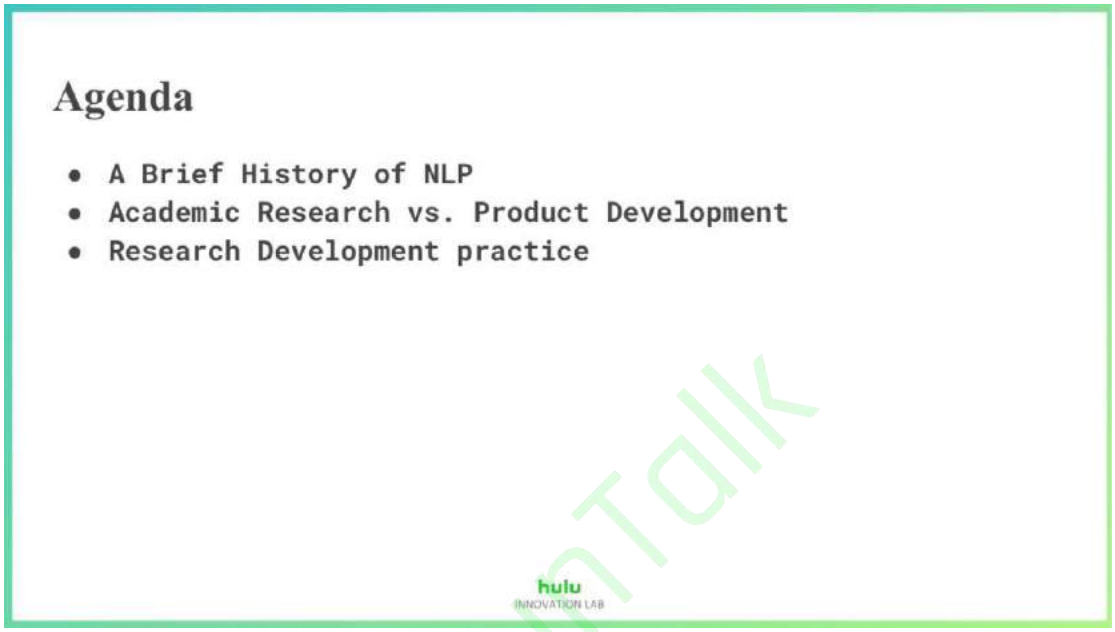
我们团队围绕长期用户行为的兴趣挖掘，分别从多峰兴趣和时间跨度进行了模型设计，并且根据该模型设计进行了系统结构的新颖创新，从而充分利用了长期用户行为的数据价值。长期兴趣挖掘有助于更好地刻画用户，值得深入研究下去，未来我们也将进一步从算法和系统推进研究，从而更灵活的捕捉用户行为模式。与此同时我们认为未来的模型发展离不开系统的适配，一个好的系统能更好的释放模型能力，因而 Co-Design 的系统模型设计思维可以更好地打开发展空间。

DataFunTalk

NLP 篇

NLP 在 Hulu 的研究与实践

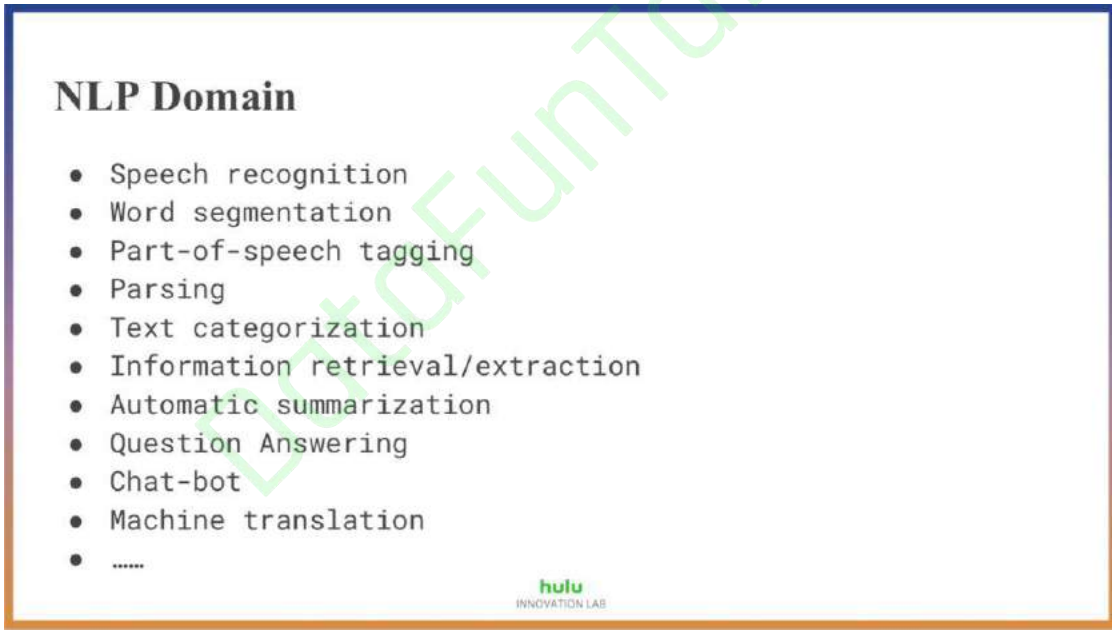
嘉宾：李凡丁　编辑：程晓杰



前言

“当你在凝视深渊的时候，深渊也在凝视着你。”当你在看 Hulu 的时候，Hulu 以及它背后的那些算法工程师们又在想些什么？不妨，就通过我加入 Hulu 以来的经历，一窥自然语言处理研发工程师在 Hulu 做的一点微小的工作。

本文共讲三件事，看似没什么内在联系，却是一个 NLP 研发工程师的切身体会和感悟。



“以史为鉴，可以知兴替”

从 NLP 简史出发，叩开 NLP 应用的大门

时间、人类、未来，都有人写其简史，NLP 作为当下 AI 研究的热点之一，似乎鲜有人总结它的兴衰胜败。但毋庸置疑，NLP (Natural Language Processing，自然语言处

理）涉及领域众多——语音识别、分词、词性标注、句法分析、文本分类、信息检索、自动摘要、QA、机器翻译、对话机器人、阅读理解，几乎所有跟自然语言沾边的话题都可以划分到 NLP 领域里。

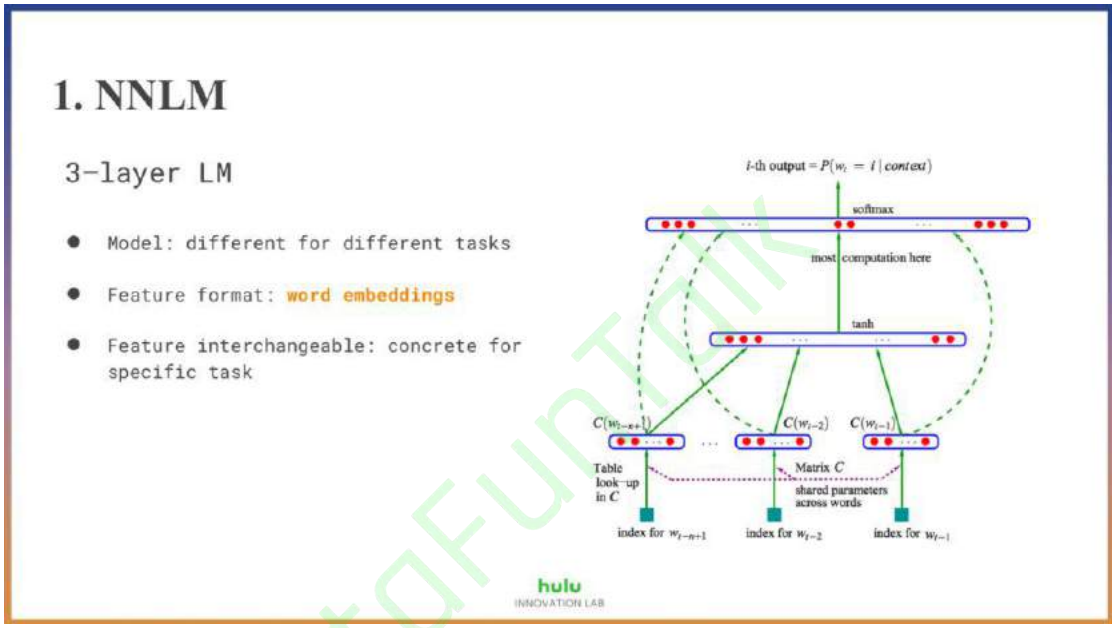
在 2018 年 10 月底，Google 推出的论文 BERT 可谓一鸣惊人，在 NLP 界引起了轰动。BERT 之前，哪些工作和模型可以进入 NLP 名人堂尚未有定论，但从文章引用数和影响力的角度不难看出一些端倪，将以下三个工作列为近 20 年 NLP 的发展具有里程碑应该并不为过。

· 2003 年 Bengio 的 NNLM；

· 2013 年 Mikolov 的 Word2vec；

· 2017 年 Vaswani 的 Transformer；

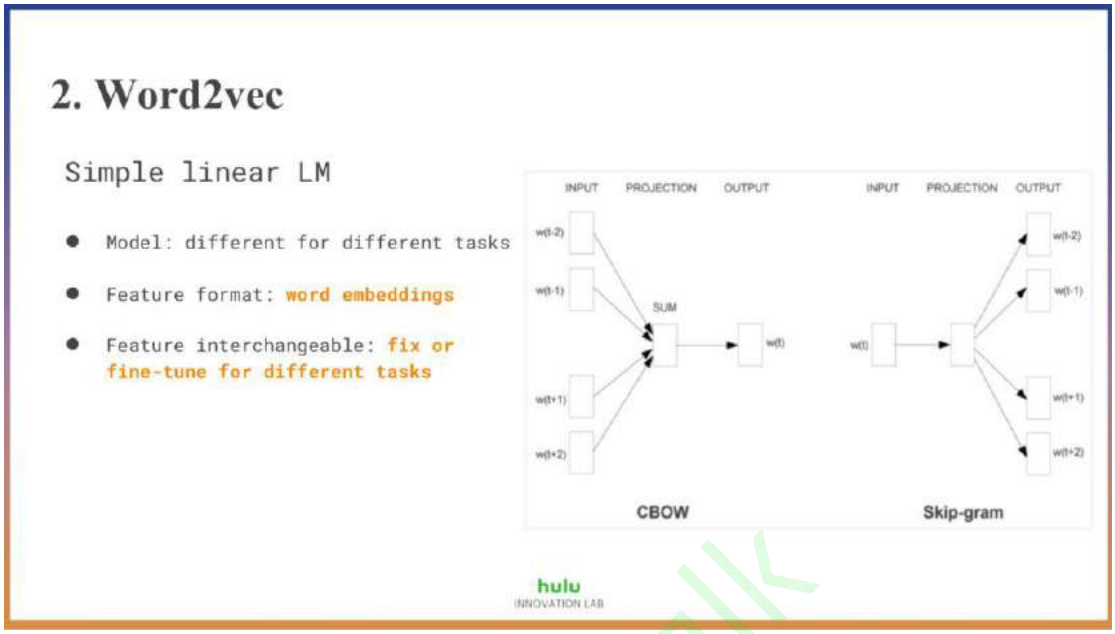
在这三个模型之前，研究者针对 NLP 的不同任务，通常采用各不相同的模型架构去处理。比如命名实体识别任务用 HMM 隐马尔科夫模型或者 CRF 条件随机场；再比如文本分类任务应用词袋模型、主题模型提取特征，再用 LR 逻辑回归或者 SVM 支持向量机做分类。而这三个模型，都可谓时代的开创者，被记录在 NLP 的史册中。



1. NNLM (Yoshua Bengio, 2003)：

上图可谓是上古时代的一张结构图，以现在的眼光来看，它其实就是一个非常简单的三层神经网络模型。沿袭语言模型（Language Model）的目标，即用前 n-1 个词来预测当前的词是什么，NNLM 只是把这个预测模型换成了神经网络。但在当时，由于这个模型最后一层是个结点到个结点的全连接层（为词向量维度，为词表大小），模型参数过多，而且当时计算力也并不算强，训练一个 NNLM 模型可能需要几个月的时间。而且，针对不同任务需要训练不同的 NNLM 模型，资源消耗极大。但无论如何，这篇论文都具有划时代的意义，因为它统一了 NLP 的特征形式——即 Embedding。在此之

后，我们做各种 NLP 任务时，一般都用词向量（word embedding）的形式来表达一个词。

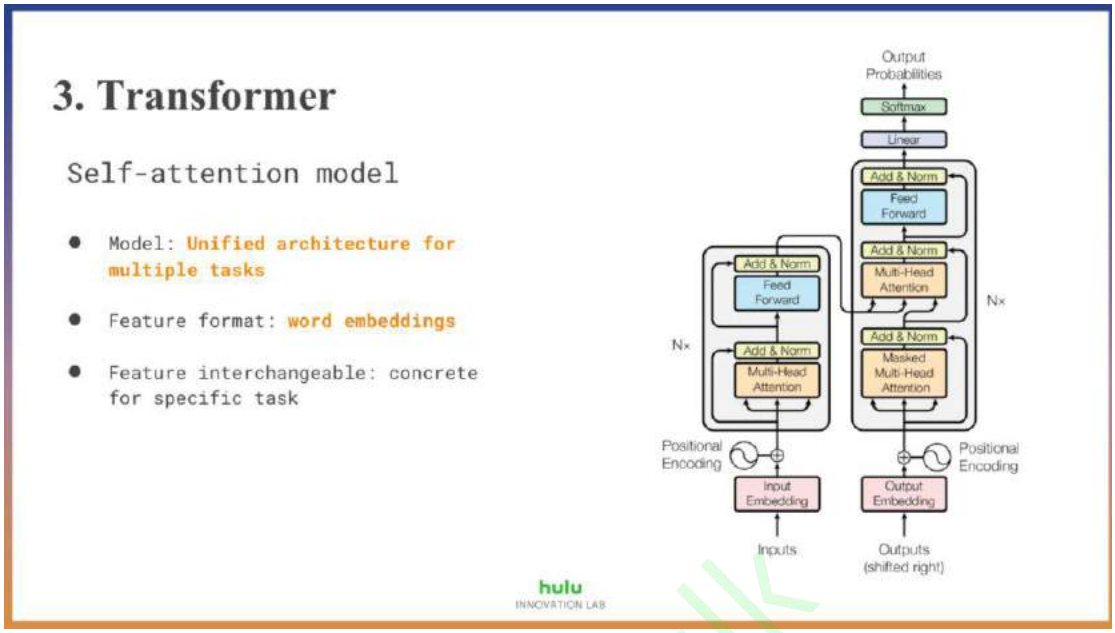


2. Word2vec (Mikolov, 2013)：

从模型的角度看，Word2vec 比 NNLM 更简单，因为它其实就是一个线性模型。但正靠着模型的大幅度简化，以及 2013 年深度学习和其硬件的飞速发展，Word2vec 可以依赖大规模语料进行快速训练。从革新的角度，它不仅延续了 NNLM 的特征形式，而且不再需要针对特定任务从头训练模型，而是在一个大规模语料上先进行预训练，再利用类似迁移学习的思想，把预训练好的 embedding 应用到其他下游任务中。

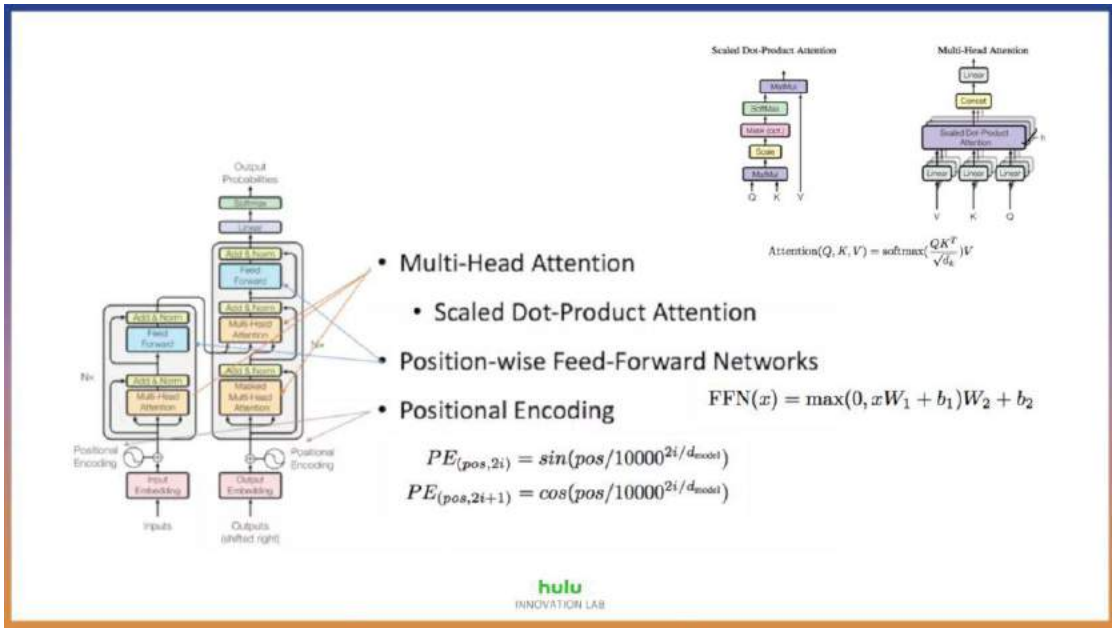
回看当年的历史，计算机视觉之所以能蓬勃发展，很大程度上靠着 ImageNet 这样一个大规模、有标注的数据集。在自然语言处理领域，却并没有一个如此大规模的标注数据集。怎么办呢？从另一个角度出发，人们发现语言这个东西，在说出来或者写出来的时候就已经成为很有价值的数据了。Word2vec 正是利用了人类语言学高度抽象、高度概

括的特点，在一个大规模语料中训练语言模型，再将语言模型里学到的信息（embedding）应用到下游不同的 NLP 任务中去。



3. Transformer (Vaswani, 2017)：

Word2vec 在 NLP 届问世以后，大家通常采用 Word2vec 作为 embedding 的初始值，再利用 RNN、CNN、attention 等深度学习模型去训练下游任务。此时，文本分类的主流模型已经变成了 LSTM, charCNN 等，命名实体识别领域也大多被 LSTM+CRF 的框架所占据。人们不禁开始思考：统一了 embedding 这种数据特征形式和 pre-train 这种特征预训练方式，我们能不能用统一的模型来处理不同的 NLP 任务？2017 年 Transformer 应运而生，像无所不能的变形金刚一样，Transformer 放弃了 RNN、CNN 的结构，完全只用注意力(Attention)机制构建模型，处理多种 NLP 任务。



鉴于 Transformer 的模型结构较前两者复杂许多，而且又是最近一年 NLP 届的耀眼新星，我们不妨细看一番，以探其究竟。从结构上，Transformer 可以拆分为三个部分：1）Multi-Head Attention；2）Feed-Forward Network；3）Position encoding。

A. 多头注意力 Multi-Head Attention

如右上图所示，Multi-Head Attention 是由 Scaled Dot-Product Attention 单元堆叠而成的。Attention 机制想必已是耳熟能详，简单来说 Attention 就是 seq2seq 模型中 Decoder 在进行解码时，学习 Encoder 输入中各词的权重分配。Dot-product Attention 是 Attention 的一种形式，它把注意力表达成 Q(query), K(key), V(value) 三元组。其中 Q 是待生成的序列，V 是输入的序列，K 为权重。例如机器翻译任务中从源语言翻译成目标语言，V 表示源语言序列，而 Q 就是待生成的目标语言序列；每产生 Q 中的一个翻译词时都需要计算与输入 V 中哪个词最有关系，这便是注意力权重 K；Attention 层之上，再经过一个 softmax 预测真实的翻译词。Scaled 则是指在 softmax

计算之前，进行尺度缩放，除以跟 Q 和 K 维度有关的一个量，用以防止点积时数值过大。如果其中 Q=K=V，我们一般又称其为 self-attention。

把很多 Scaled Dot-Product Attention 单元堆叠，如同卷积一样形成通道形式，然后将结果级联到一起，再经过一个线性映射，这便是 Multi-Head Attention 的过程。

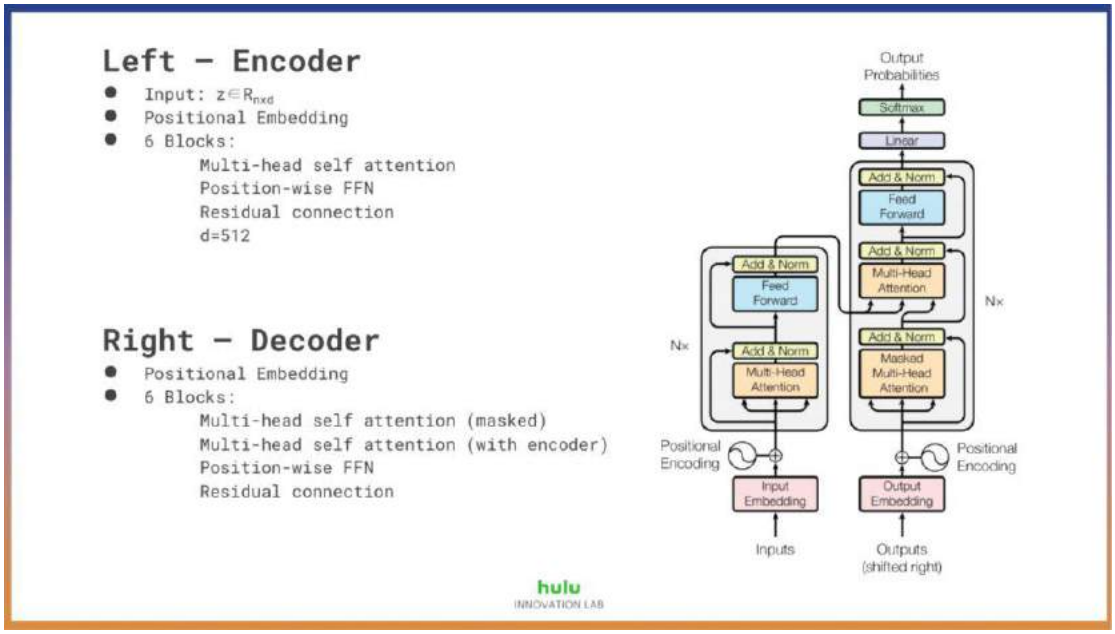
B. Feed-Forward Network

通过 Multi-Head Attention 层后，再经过一个前向神经网络 Feed-Forward Network，用以增加网络的非线性。并且这个 FFN 添加了像 ResNet 一样的 shortcut 连接结构。

C. Position Encoding

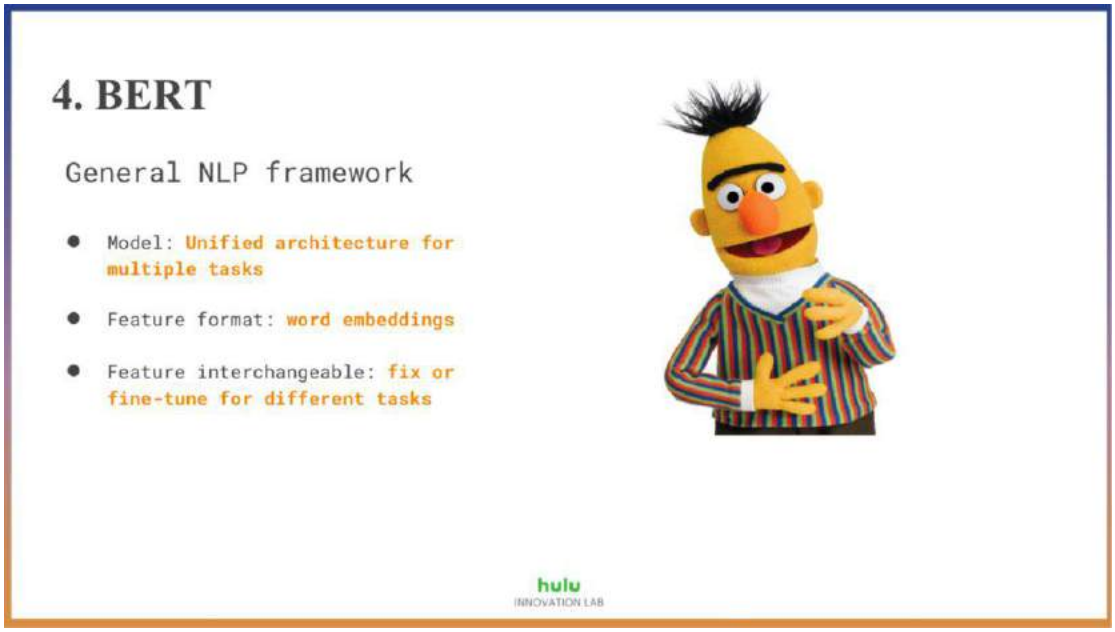
因为在 attention 机制中每个词都会关注前后所有的词，导致所有词在位置信息上其实是等价的，所以 Transformer 中引入 Position Encoding 表达每个词在原始句子中的位置信息，来弥补这一缺陷。Position Encoding 既可以用一个固定的函数表达，也可以用一个可学习的函数表达。实验发现两者效果差不多，干脆就采取固定的，sin 族函数来表示。由于 sin 函数的特性，一定程度上还可以有效地表达句子中词的位置的周期性。

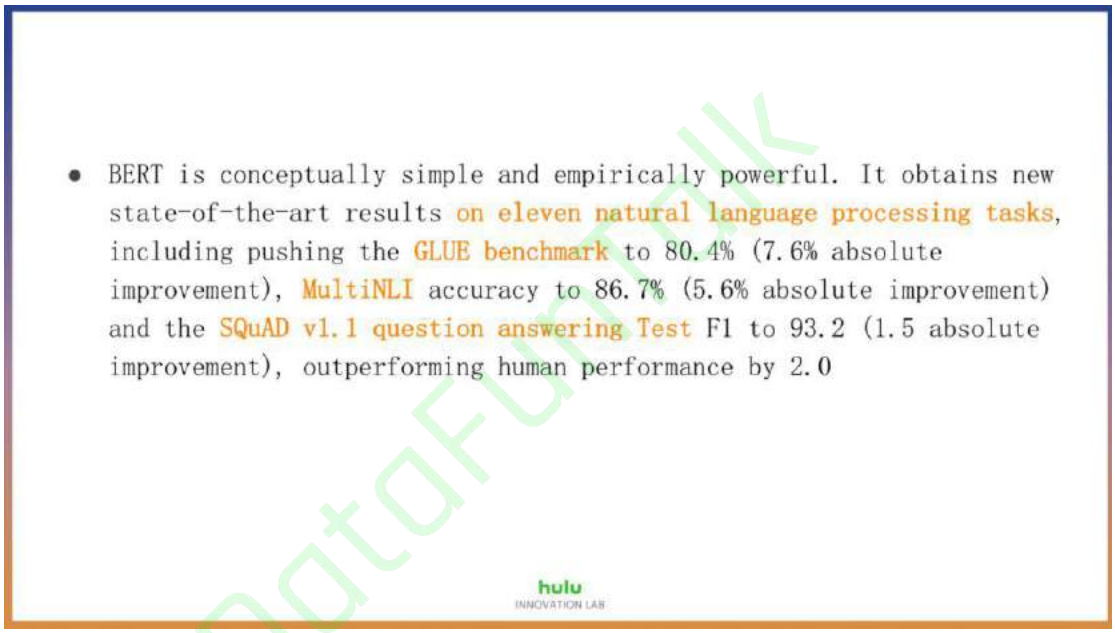
DataFunTalk



从整体上看，Transformer 结构的左边是 Encoder 部分，右边为 Decoder 部分。Encoder 堆叠了 6 个 Multi-Head Attention + Feed-Forward Network + shortcut 的结构。Decoder 同样堆叠 6 层这样的结构，然后与 Encoder 的 self-attention 结果共同送入 Multi-Head Attention，预测最终的输出。

那么，Transformer 结构为何能饱受青睐呢？首先，使用 Attention 机制后，每层的复杂度比 RNN 要小。再者，RNN 是一个递归的模型，t 时刻的计算需要依赖 t 时刻前的状态，很难进行并行计算。而 Attention 结构可以很容易地支持并行，因此可以用到更多的数据。另外，Attention 机制解决了一个 RNN 里一直以来很难处理的长距离依赖问题。RNN 中诸如 LSTM 模型，虽然添加了遗忘门等机制，但是依然不能很好的解决长距离依赖。Transformer 由于会关注输入的每一个位置，因此可以很好的解决这一问题。Transformer 也于是在之后几乎所有 NLP 相关的任务都展露了锋芒。





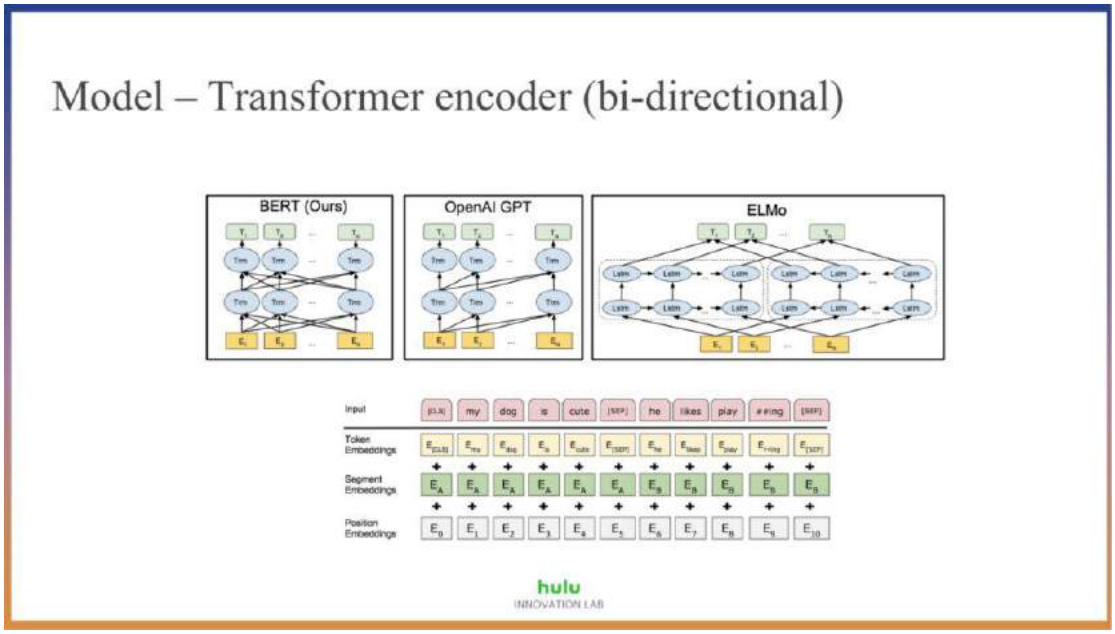
DataFunTalk

4. Bert：

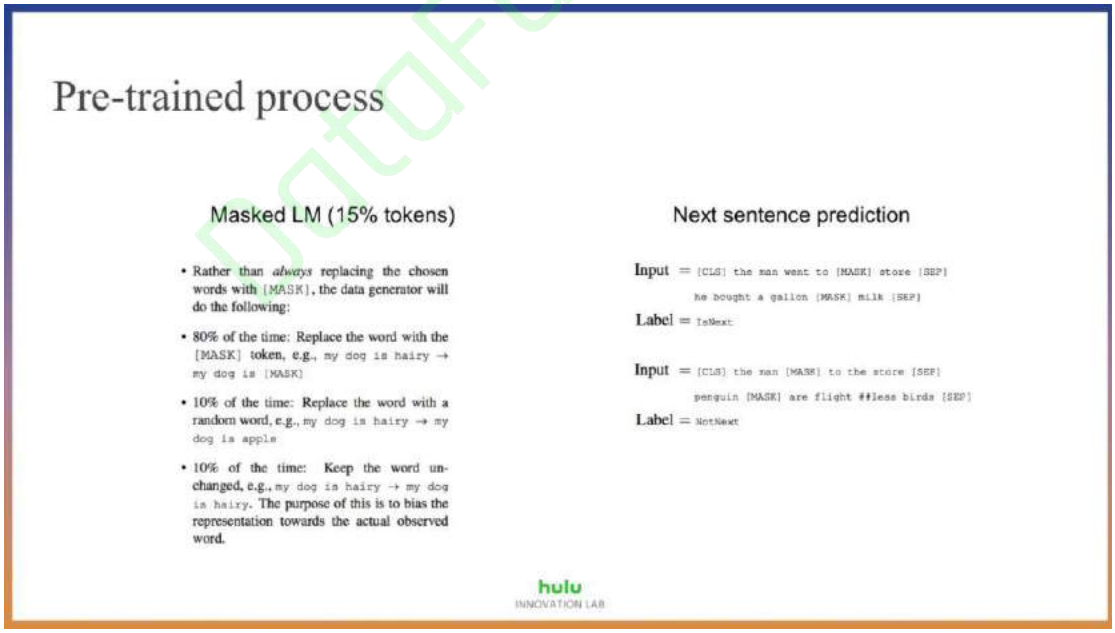
至此，我们既统一模型架构，也统一特征形式，下一个问题也自然而然——能不能在大规模数据集上训练一个 Transformer，然后再用到其他各式各样的下游任务中？其实，包括 AllenAI 的 ELMo，OpenAI 的 GPT，Google 的 Bert 等，都在做这样的尝试。其中 Bert 从 2018 年 10 月提出便一骑绝尘，引起 NLP 届广泛关注，凭借的是其惊人

的效果。用论文中原话来说，Bert 是一个理论上非常简单，但是非常有效的模型，它在 11 个自然语言处理的任务场景中取得了当前最好结果，并且把 GLUE benchmark 提高了 7.6%，在 SQuAD 数据集上已经超过人类表现。

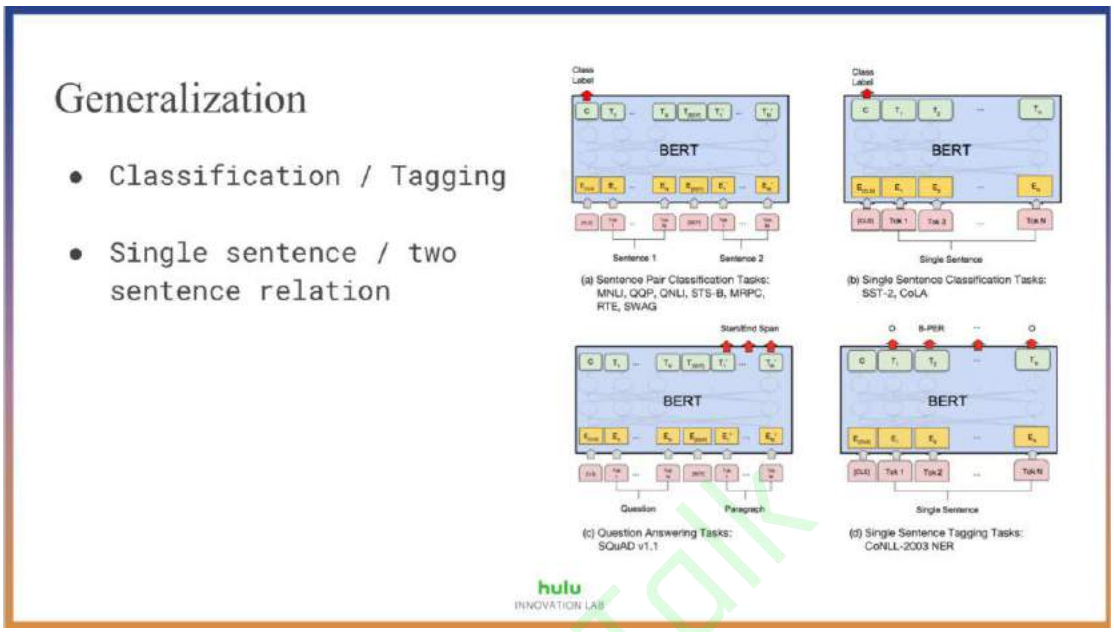
上图是 GLUE（General Language Understanding Evaluation）的榜单，GLUE 将一些 NLP 任务集中在一起，可以评价一个模型在不同任务上的通用性。这个表是 2018 年 11 月份的统计，可以看出 Bert 当时技压群雄，比每个数据集之前最好的结果还要好。



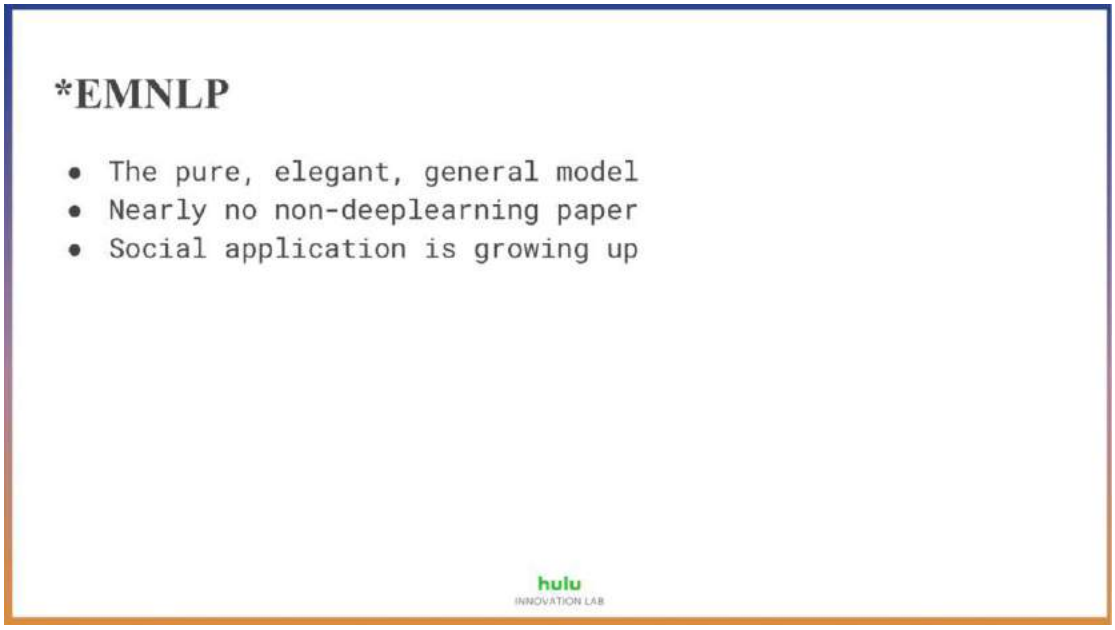
简单来说，Bert 采用了 Transformer 中 encoder 的双向结构。在输入端除了有 token embedding，position embedding 之外，还加上了 segment embedding。这主要是为了处理双句任务，比如判断两个句子之间的关系，以及 Question Answering 等，通过 segment embedding 来区分前后句子。



Bert 的预训练包含以下两种操作，一是把句子里的一些词随机掩盖，然后让模型去预测这些词。另一个是用前一个句子来预测下一个句子。 通过优化这两个目标，Bert 就可以在大规模的语料中进行训练了。



继承 Transformer 的优点，Bert 自然也可以应用到多种不同的 NLP 任务中。文章提出了上图的四种架构，对应 Bert 四种不同的应用场景，既可以做分类任务又可以做标注任务、既能处理单个句子也能胜任双句任务，因此 NLP 很多问题都可以用以上模型来套用。



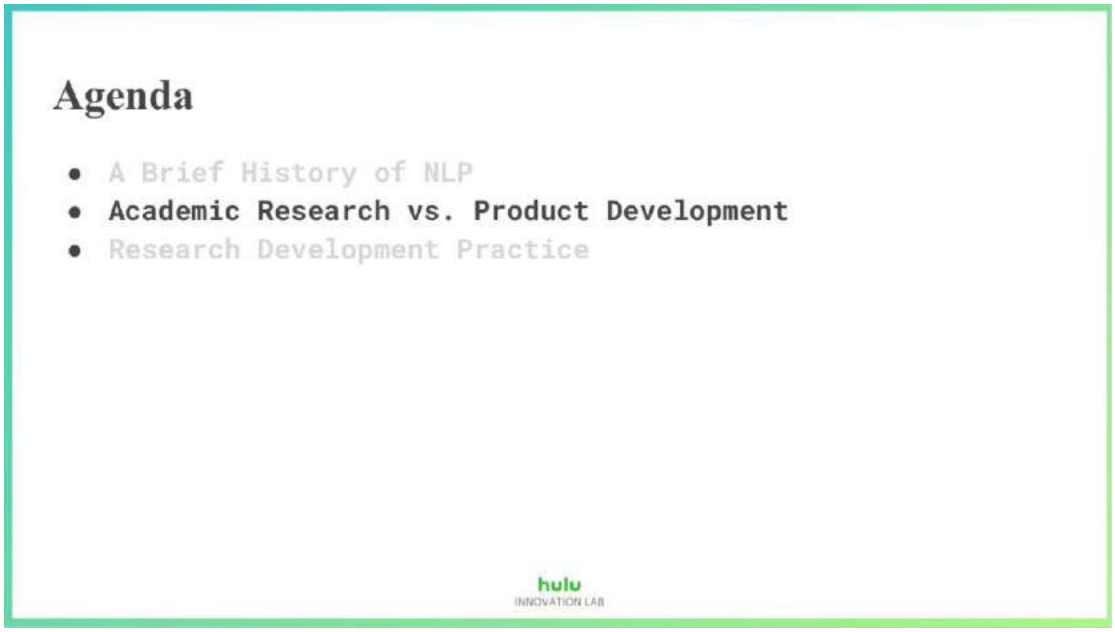
5. NLP 展望

在此也不妨分享一下我在 10 月底参加 EMNLP 会议时对 NLP 发展方向的一点思考。

A. 在学术上大家还是一直在追求纯粹的、优雅的、通用的模型。

B. 从现象上来看，会议中已经很少能见到非深度学习的论文，深度学习现在基本占领 CV、NLP 领域。

C. NLP 在社会应用和工业届应用受到大家更多的关注。会议上有多个 session 讨论这个问题，而且参与人数众多，可以发现人们越来越关注自己的研究领域在生活中或者在产品中的应用。 这也引出我们下一部分要讨论的问题：学术研究和产品开发有什么区别和联系？这对很多刚毕业不久或还没毕业的同学来说，更容易引发共鸣和思考。



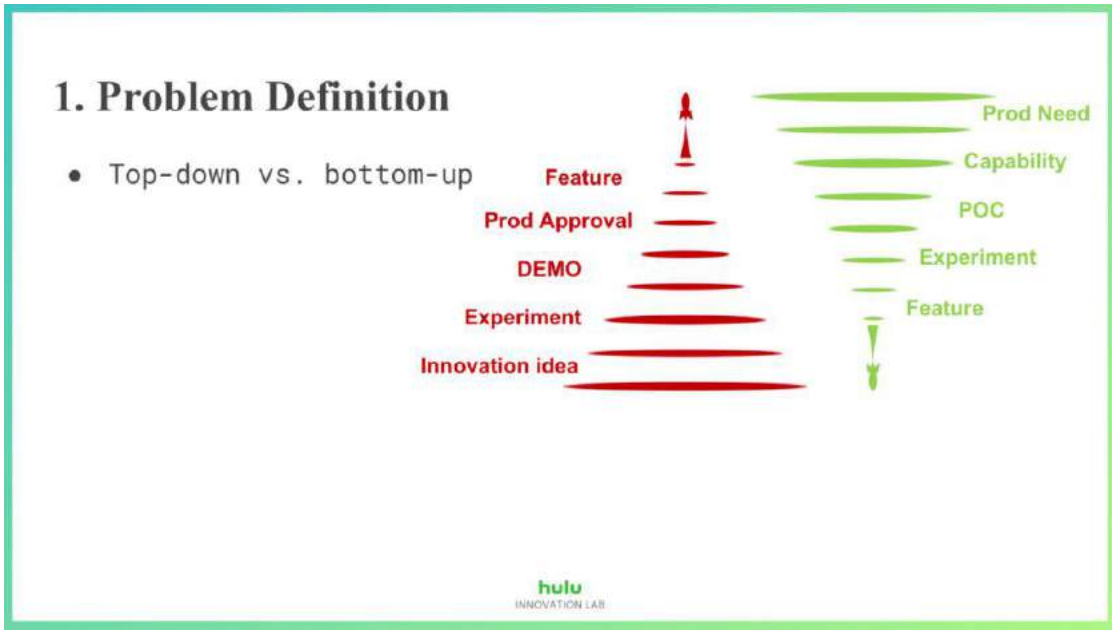
“当学术研究遇到产品开发”

从一个算法研究工程师的角度，讲述从学校到公司的心路历程

以下将从 5 个方面介绍学术研究和产品开发之间的对比和利弊权衡 :1. 问题的定义；2. 数据的处理；3. 模型的选择；4. 评价指标；5. 开发流程标准化。

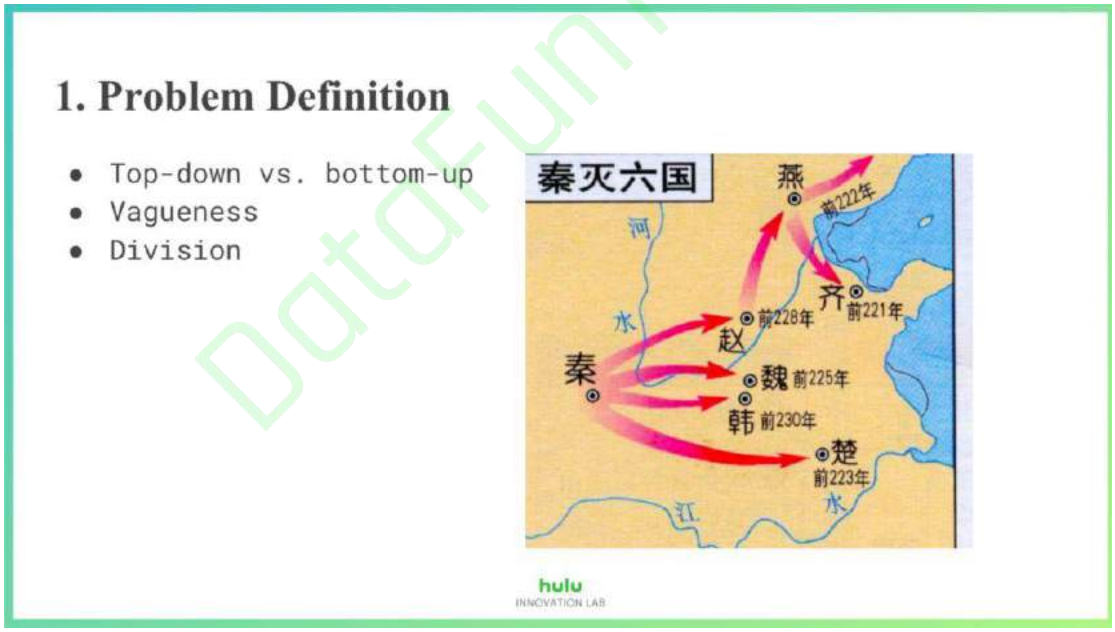
1. 问题的定义：

在学校时我们往往知道问题的输入是什么，只需要用一个合适的模型不断优化，来找到一个更好的解，提升模型的效果。但在公司场景中，更多时候你可能需要先定义问题是什么。



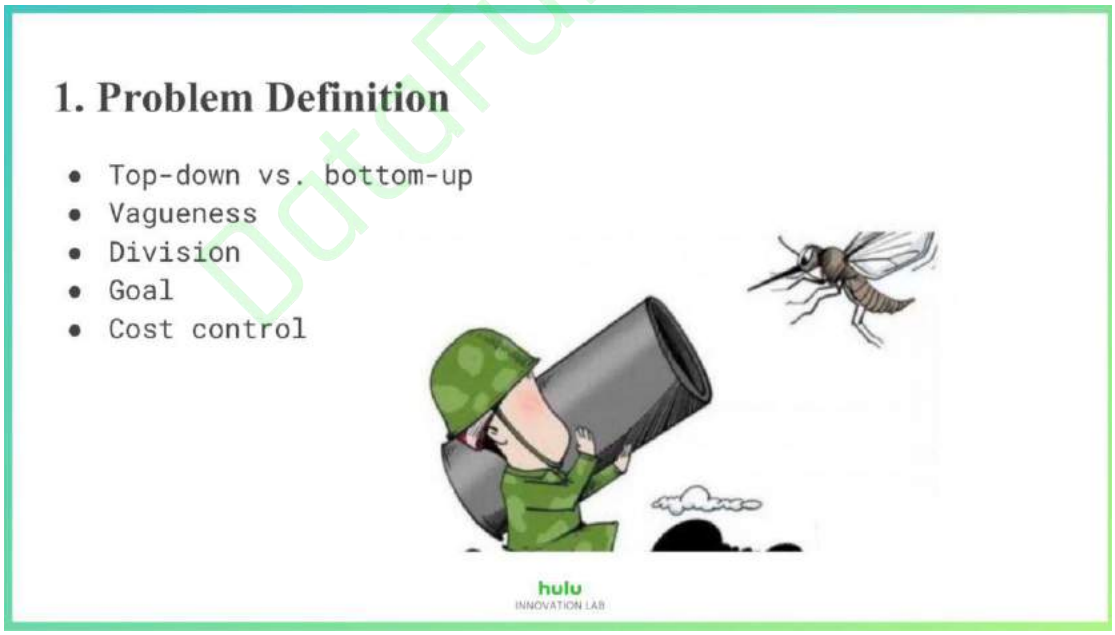
A. top-down 和 bottom-up：从问题产生背景上讲，问题主要分为两种方式定义：top-down 和 bottom-up。理想情况下 bottom-up 的问题一般是：你有一个创新点，然后去做一些实验进行验证，之后设计成一个小 demo 形式给产品经理查看，产品经理认为效果不错，再去做一些 UX 上的优化，最终成为一个产品特征。而 top-down 通常是：产品经理有一个需求，然后去找公司里哪个团队有这样的能力去做，他发现你有能力完成这个需求，找到你后针对这个项目去做 POC (proof of concept)，在此基础上针对真实用户进行 A/B 测试，如果效果不错，最终就可以转化为产品特征。

B. Vagueness：但很多时候产品经理提出的需求是并不一定十分清晰，而是相对模糊，所以作为算法工程师，对问题，对通过技术能解决、善于解决什么问题要有自己的判断和见解，以此给出问题正确的定义。



C. Division：对问题要有划分，并且有策略。定义问题时需要分清哪些是重点，哪些是眼前亟待解决的问题，哪些是需要长远去看的问题。

D. Goal：先定一个小目标。在公司做产品，如果目标不同，对于问题的定义也不同。比如要分清是做一个 PoC 还是一个简单调研，甚至目标是做一个产品特征，这些都决定了定义问题时侧重点不一样。比如需要做一个 demo，在模型的选择和实验上，就不必过分关注细节，更重要的是强调一个新的概念，新的设计或者功能。



E. Cost Control：注意产品的成本控制，包括对时间、金钱、人力成本等的控制。在定义一个问题时也要考虑是不是对公司或者对问题解决的领域有长远影响？比如做一个机器学习任务，可能很多时候缺少标注数据。那我们在定义问题的时候就要计划清楚数据应该从哪来？是外包，还是自己上阵，亦或找一些实习生去做，都是成本控制所要考虑的问题。

2. 数据的处理：

A. 有哪些数据可以用？

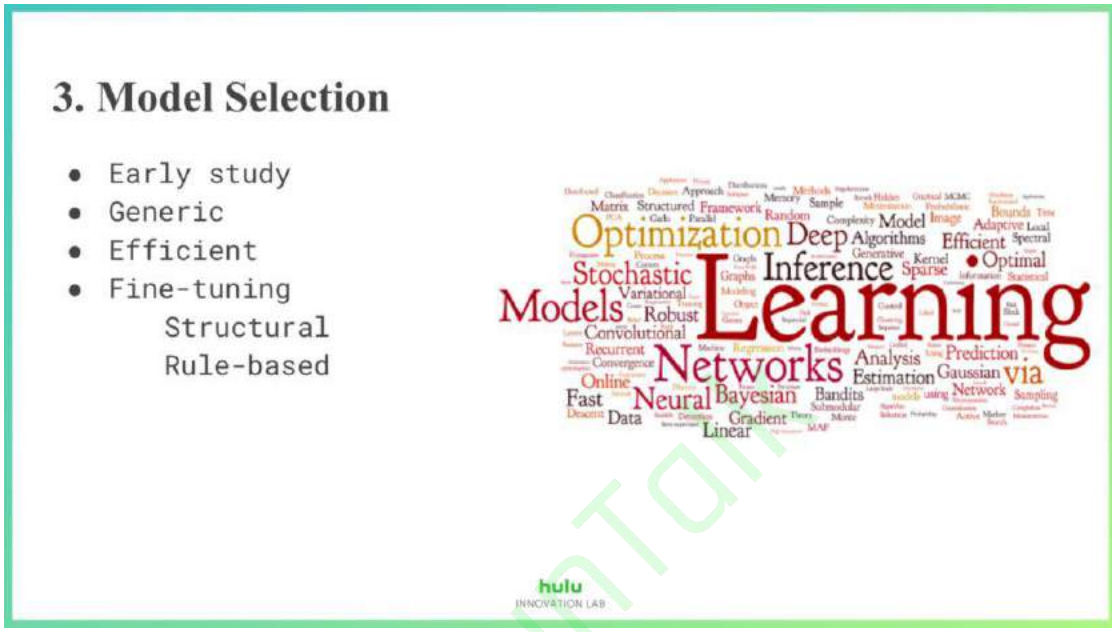
Hulu 有自己的视频，也包括携带的音频、字幕等，同时还有内容提供商给的一些演员、导演、类型、描述、标签等辅助信息。当然，我们还站在巨人的肩膀上，因为有很多其他组的同事、前人留下的各种各样的模型和数据供我们使用。

B. 应该使用那些数据？

数据显然不是越多越好，因为不同的数据有不同的处理难度。

C. 怎么用？

无论中英文，真实的数据中往往是比较脏的数据，显然从这些数据中训练好的模型是不可能的。在公司场景下，数据预处理是一个非常重要的环节，在 NLP 中包括对关联符号，标点，数字，人名等的处理，经过预处理的数据才能进入下一步，被模型所使用。



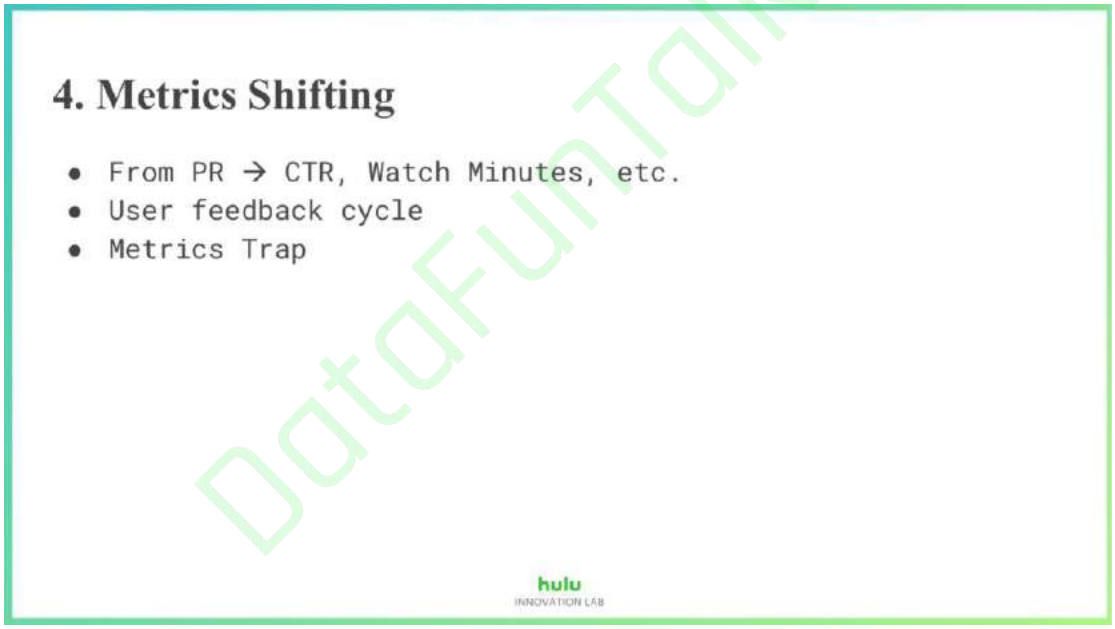
3. 模型的选择：

A. 无论做什么任务，都需要有前期调研工作，吸取前人之经验，看看在这个问题上成熟的、主流的方法有哪些。

B. 如果不知道哪个模型适合产品目标，可以先选一个最通用的模型。比如 seq2seq 问题就用我们刚才讲过的 Transformer 结构，序列标注任务用 LSTM+CRF，分类任务选用 TextCNN、fastText 等模型。当然了，现在你都可以使用 Bert。

C. 选择比较高效的模型。在公司往往要选更高效率的模型，先做出一个基准结果，评价此方法是否可行。比如公司产品开发周期为 2-3 个月，倘若你花很长时间调试出的模型最后效果一般无法使用，很容易就会陷入进退两难的尴尬境地。

D. 模型微调(fine-tuning)。对于模型 fine-tuning 分为两个部分，1) 对模型结构上的 fine-tuning ,比如说保留一个大模型的前面若干层 ,对最后的 1-2 层进行 fine-tuning。2) 在模型的基础上加入一些人工规则。诚然，我们都在追求通用的模型架构，并且这也是 AI 前进的一个方向，但是现阶段的很多场景中如果不引入人工干预，单靠机器学习或者深度学习其实很难支撑产品级别的应用，所以 fine-tuning 有时需要一些针对用户、针对数据、针对结果的规则过滤。

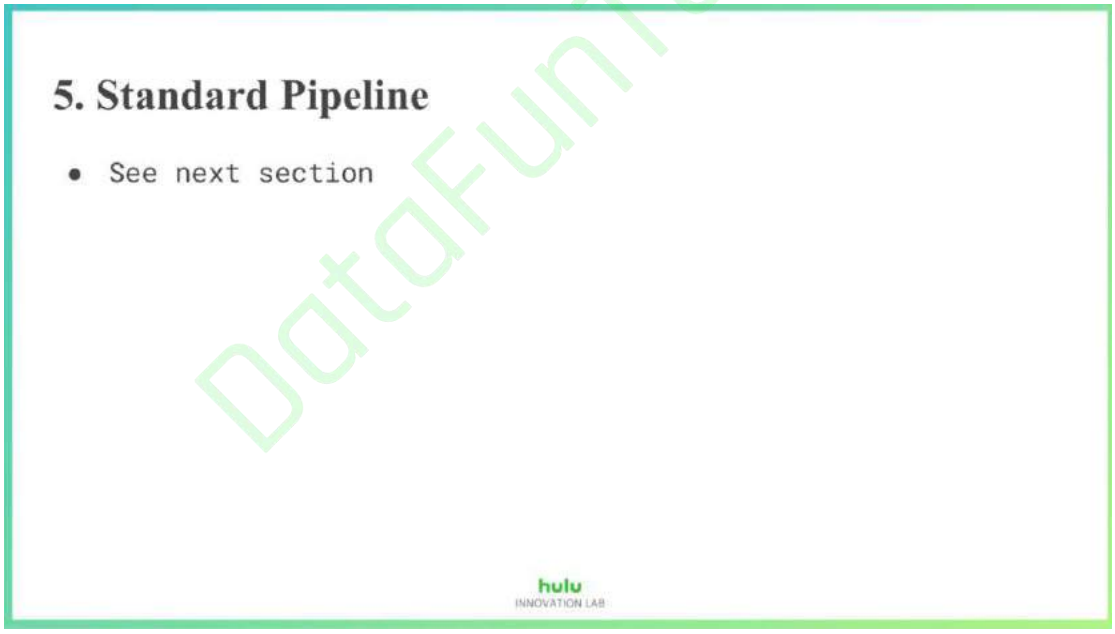


4. 评价指标

A. 在学届，我们可能更多关注 Accuracy、Recall、ROC 等指标。但在公司，我们要更关注和用户行为相关的评价方式，比如展示位点击转化率 CTR、用户的总观看时间等指标。

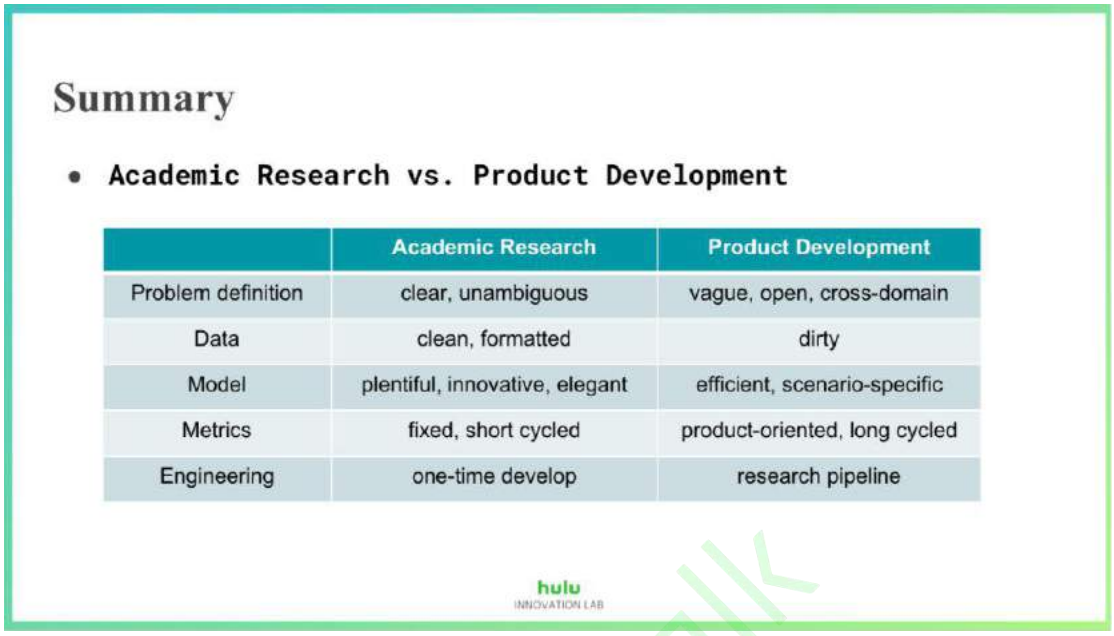
B. 采用多个版本的控制，将用户的反馈传递到模型中。比如要做在线 A/B 测试，但其反馈到模型的周期往往很长，这时就需要用多个版本来控制线上运行和线下实验的模型。

C. 警惕 metrics trap。在学术界常用的评价指标在实际产品中往往并不适用。比如在机器翻译中多使用 BLEU 作为评价指标，但是此评价指标会忽略语法上的准确性。比如我们把翻译结果里的词序进行颠倒，可能 BLEU 评价很高，但实际却表达出截然不同的语义，这将对真实的用户体验造成极大的影响。



5. 流程标准化：

在产品开发中，流程的标准化对项目管理、开发效率、后期维护等都有着深远影响，我们将在下一部分中详细介绍。



6. 总结

总结学术研究和产品开发的区别如下：

A.从问题定义上讲：在学术研究时是明确的、没有歧义；在产品开发时是模糊的、开放的问题。

B.从数据角度上讲：学术研究时数据是干净的、规范的；在产品开发时可能会遇到脏数据。

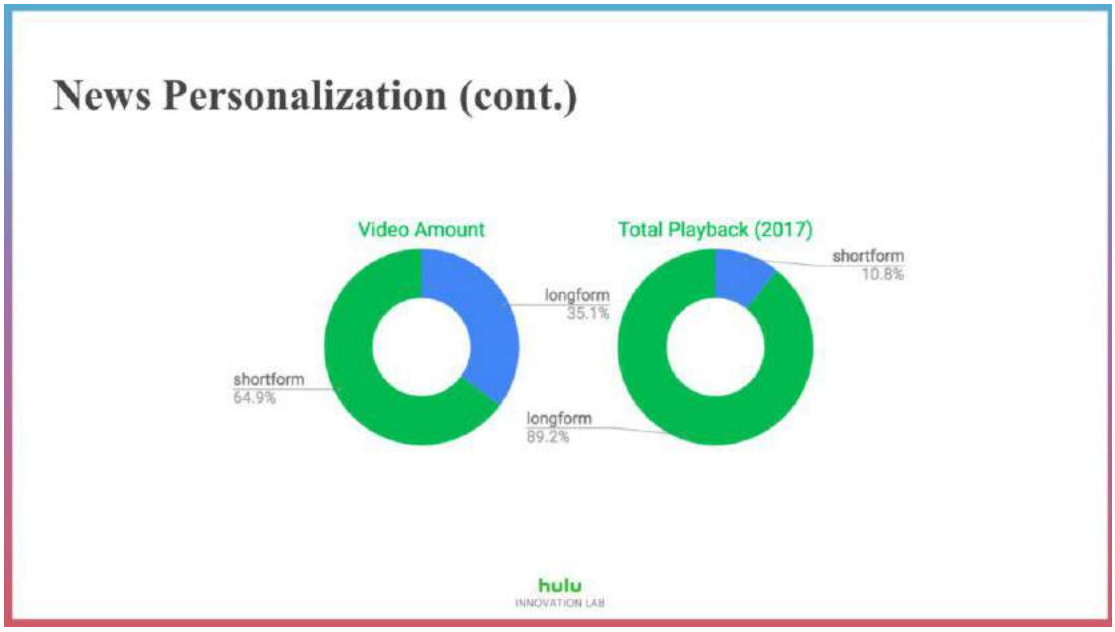
C.从模型角度上讲：学术研究追求优雅的、创新的模型；在真实的用户场景中可能需要制定相关的人工规则。

D.从评价指标上讲：学术研究采用比较固定的评价指标；在公司场景中，则会采用一些产品导向的评价指标。

E.从工程的角度上讲：学术研究可能写一次代码，调整参数得到理想的结果就满意了；在公司，要不断进行产品迭代，观察具体数据和结果，对产品进行细致维护。

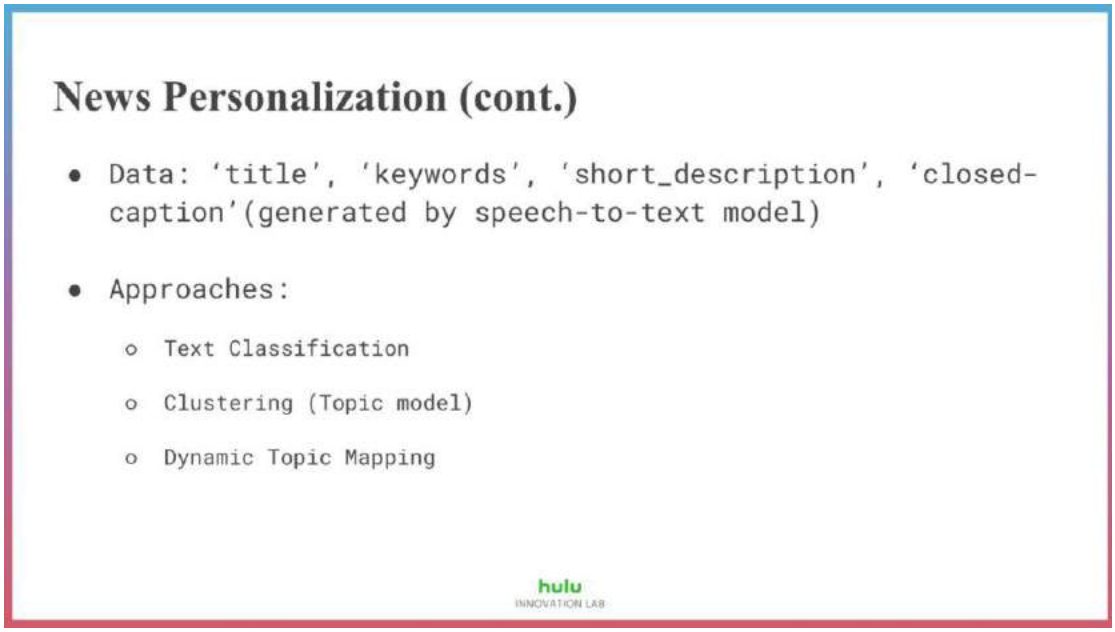
接下来，不妨从两个我最近在做的项目出发，一探在 Hulu 做 NLP 开发的真实情景：

1. News Personalization：



A. 问题定义：

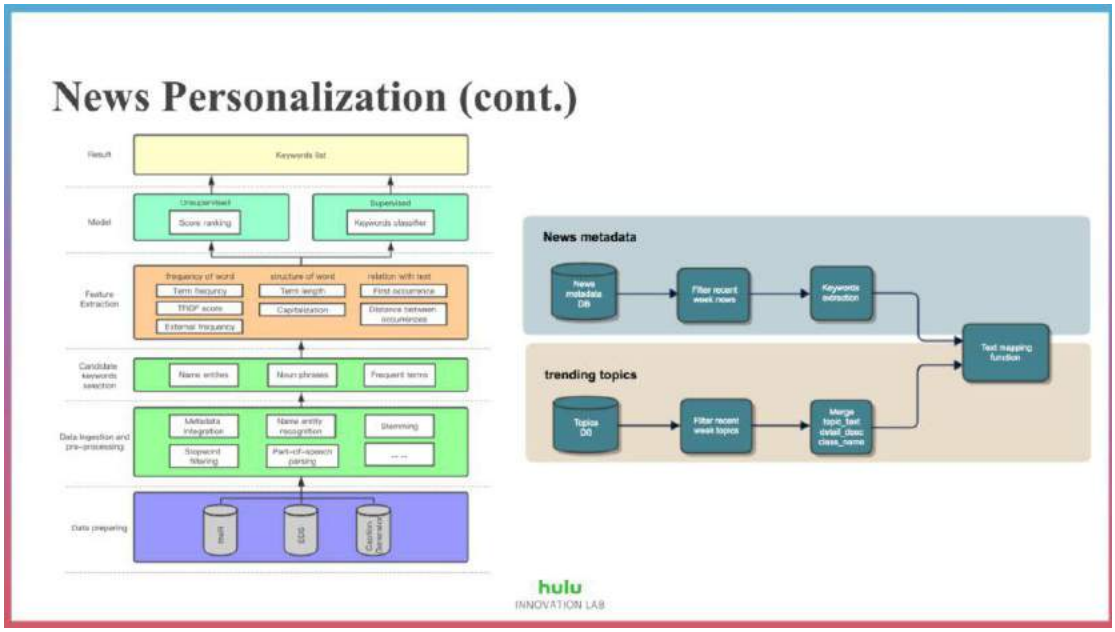
News personalization 项目的任务是给用户推荐个性化的新闻。这是一个源自产品经理的 top-down 的需求。它分为两个场景：一个是在用户登录时推荐，一个是用户看完新闻后推荐下一个相关的新闻。这个问题的定义比较明确，就是推荐算法。在 Hulu 有很多成熟的推荐算法，比如协同滤波推荐、基于 DNN 的推荐等。但是 Hulu 新闻的场景属于短视频推荐，其特点是有很多的视频数量，但是有较少的观看数量，这就决定了不太适合用基于协同滤波的推荐算法。并且新闻具有时效性，我们不能给用户推荐几年前的事情，而是需要推荐近期、甚至当天的新闻。再者，一些用户虽然在当天或者近段时间内观看比较少，但用户历史上偏好某一类新闻，这也是我们可以利用的特征。于是最后，我们将任务定义为：给新闻做分组，然后让用户去观看他喜欢的分组中最新的新闻。



B. 数据与模型：

在此 NLP 任务中可用的 feature 包括有视频标题、关键词，简短描述、字幕（由语音转文本的模型生成）等。NLP 中有很多的方法来处理这个问题，比如文本分类、聚类、主题模型等。其中 Dynamic Topic Mapping 就是假设我们已经有一些新闻主题，再将提取出的视频主题对应到到新闻 topic 上。

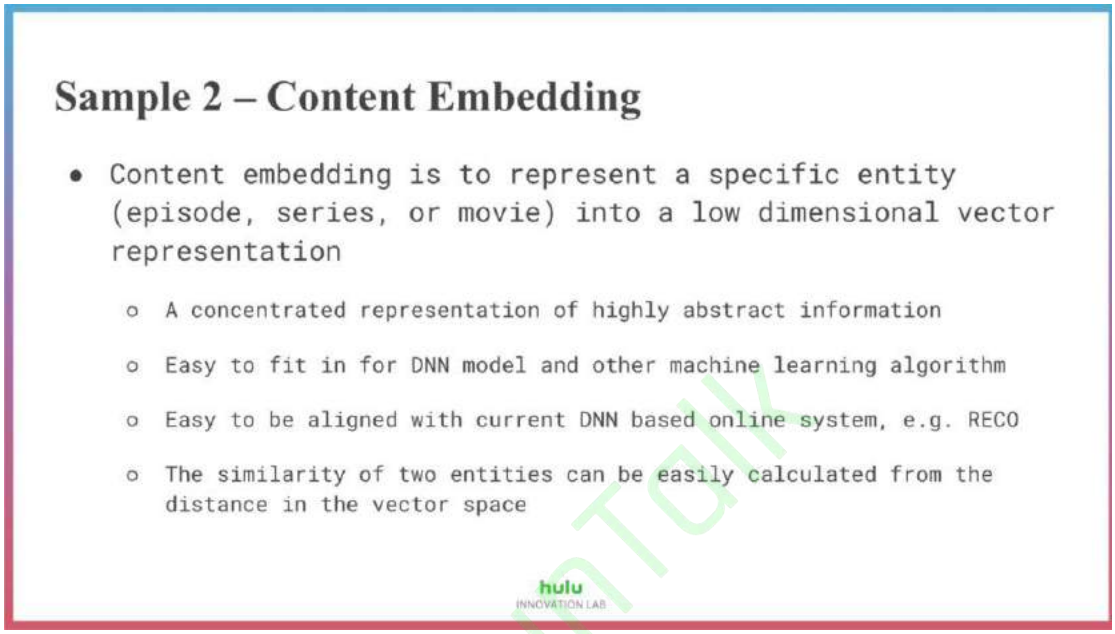
对于文本分类任务存在的问题是其类别是固定的，而现实中新闻的类别会随着时间的推移而变化。聚类也会存在这一问题，一些动态的聚类方法又非常复杂。于是我们设计了了 Dynamic Topic Mapping 方式。这主要因为我们有第三方提供的近期发生的新闻事件，一方面可以对我们对新闻提出的动态主题进行验证和监督，另一方面也可以把它们作为 trending topic 的信息。最后，只需要将新闻提取的新闻关键词与 trending topic 进行匹配即可。



C. 整体架构：

上图为系统的整体架构：首先对 data 去做一些数据预处理，比如：去停用词、命名实体，句法分析等。再去做词频分析，有监督分类出关键词或者无监督地提取关键词。然后将这些关键词和 trending topics 的关键词进行 mapping，最后计算相似打分。

从目标出发，这个项目是一个 PoC，我们更多的注意力是先把这个项目实现。当然之后有很多事情需要进一步迭代，比如：classification 应用更复杂，更准确的模型，topic mapping 里引入层次结构，优化相似性评价指标等。一旦有一定的用户实验反馈，就可以很轻松地进行下一步持续的优化。

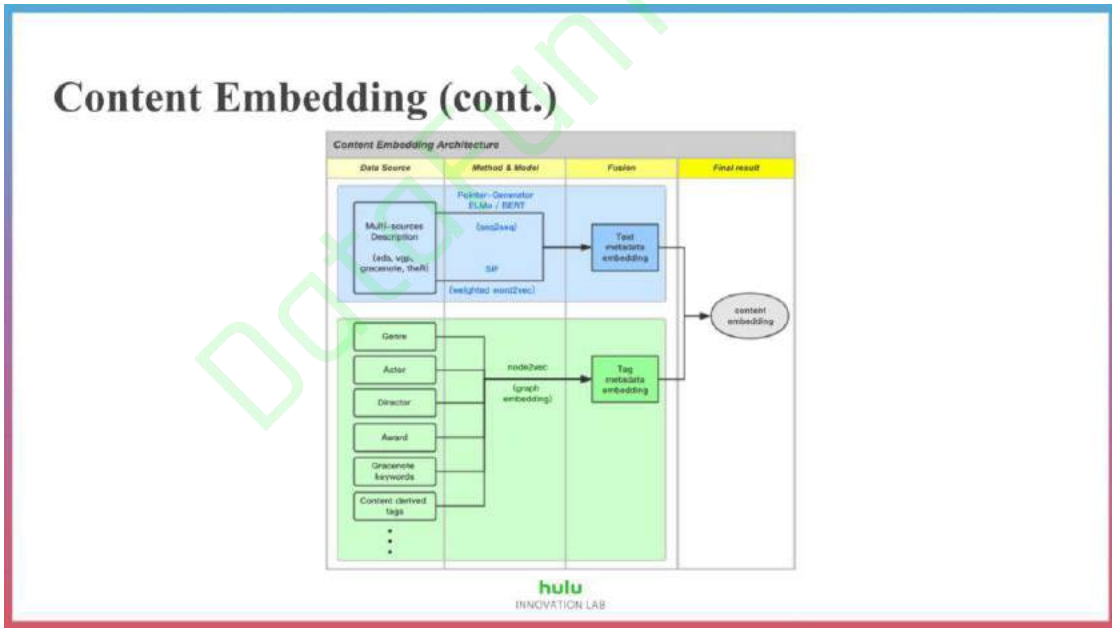


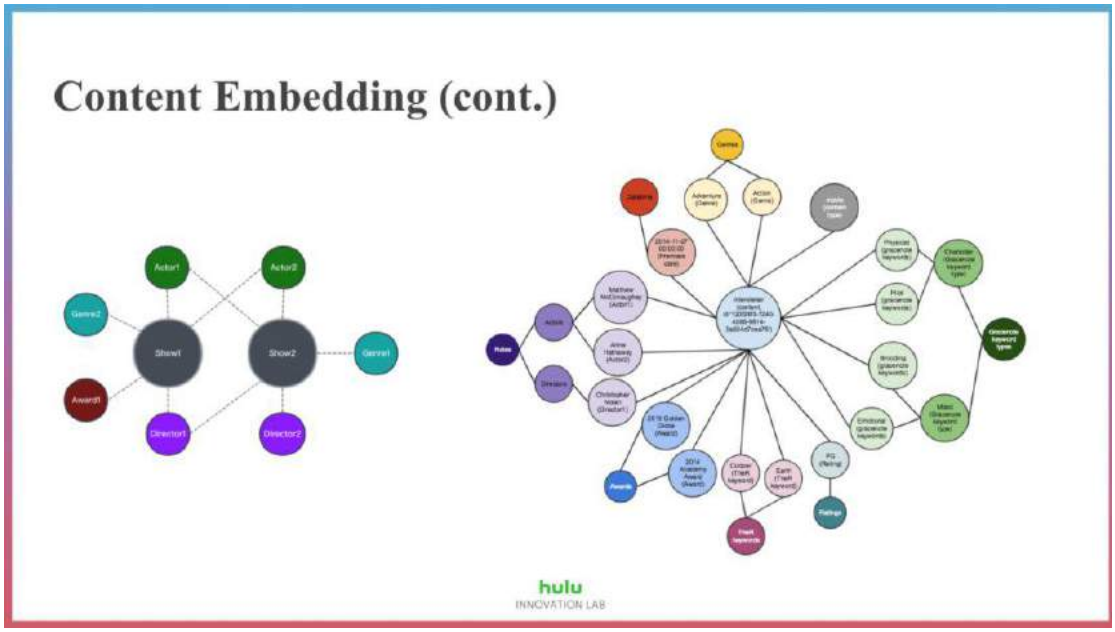
2. Content Embedding：

A. 问题定义：

利用每部视频的事实和描述性的标签，对视频进行 embedding 向量化表示。这是介于 top-down 和 bottom-up 之间的需求。从 top-down 角度讲：推荐系统经常会面临冷启动的问题，在用户还没有很多点击的时候，经常会推荐不出去。这时我们会将库中内容和用户看过的内容做相似度匹配，然后推荐给用户。从 bottom-up 角度讲：很多场景如电影、电视剧推荐，很难显式计算两个视频之间的相似度。这时丰富的标签信息可以帮助我们对 show 进行 embedding 便于相似度计算。Content Embedding 的优势

有一下几个方面：1）它是对视频标签、元数据 metadata 的一个高度抽象表示；2）它适合用于 DNN 等各种机器学习、深度学习算法中；3)作为向量表示，它很容易集成到现有系统中；4）可以通过直接计算两个实体在向量空间中的距离来计算它们的相似度。



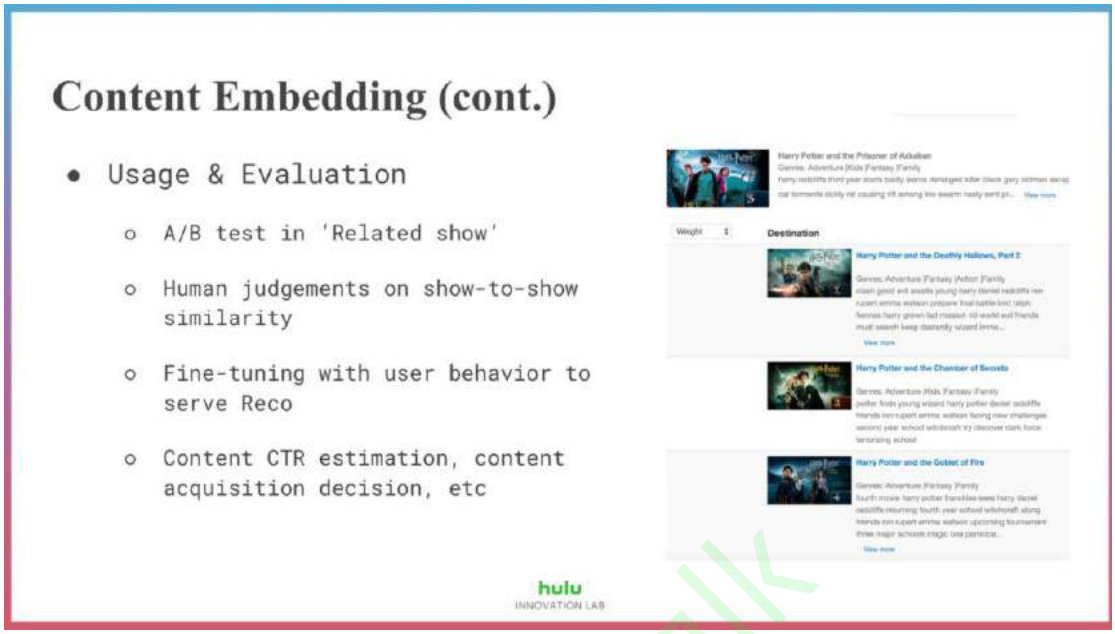


B. 数据与模型：

电影、电视剧推荐场景中可以使用的 NLP 特征主要：影片发布时间，评级，演员和工作人员，影片描述等；从第三方获得的外部信息，如关键字，标签，奖项，票房收入等。这些 NLP 特征又可以分为文本类型和标签类型。文本类型可以通过一些 seq2seq 模型提取高层语义特征，得到 featrue representation。

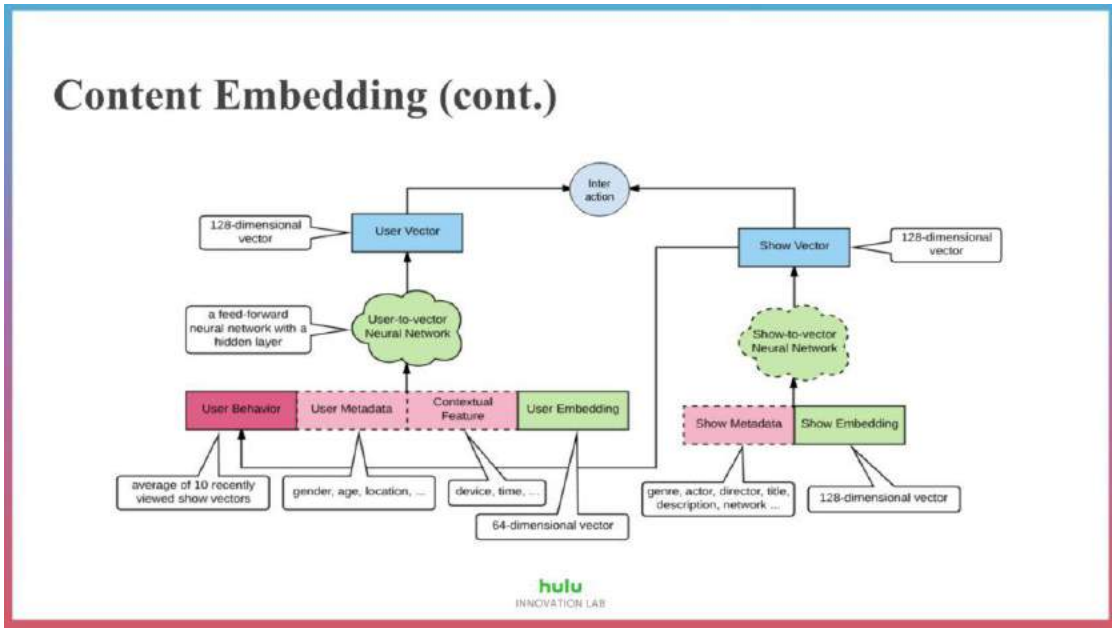
对于标签类型，如果考虑实体之间的关系，可以把它看成知识图谱问题。但如果淡化实体之间关系，只考虑每个content和它共现的信息，就可以当做为network embedding的过程。在我们的场景中，要去计算两个 content 之间的距离，显然可以剥离掉它们之间的实体关系，于是可以采用一些网络学习的方法。最后我们选择了基于 node2vec 的解决方案，从理论上这是一个比较简单的模型，它的思路和 Word2vec 完全一致，只是需要先对图中的边进行路径采样，再将每个路径当做 Word2vec 中的一个句子，每个节点当做 Word2vec 中的一个单词，之后在这些采样路径上去学习 embedding。

这其中根据元数据和标签的特性又做了很多结构和优化目标上的细节调整，这里就不一一展开叙述了。



C. 评价指标：

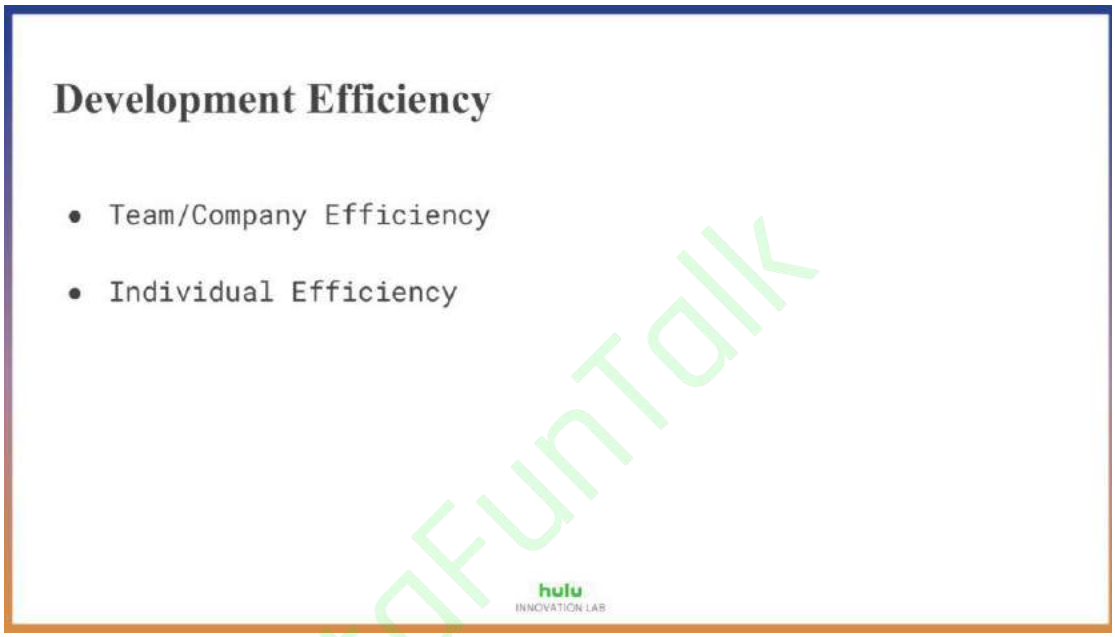
Content Embedding 的评价有很多种方法：1）每一部 show 在观看的时候有一个相关内容列表，可以用这一列表中内容的点击率作为评价标准；2）通过人工编辑团队判断内容之间的相关性；3）基于 Content Embedding，利用用户的行为对线上的其他服务进行 fine-tuning，以获得更好的线上指标；4）用 Content Embedding 为特征，进行内容点击率预估等。



D. 整体架构：

上图为 content embedding 集成到推荐系统的一个架构图。对于 user，可以利用用户标签、用户上下文信息等特征得到 user vector 。对于 show，可以利用前文介绍的方式获得 show embedding。将两者作为输入，通过一个深度神经网络来建模它们的交互信息，以此为用户进行更精准的推荐。

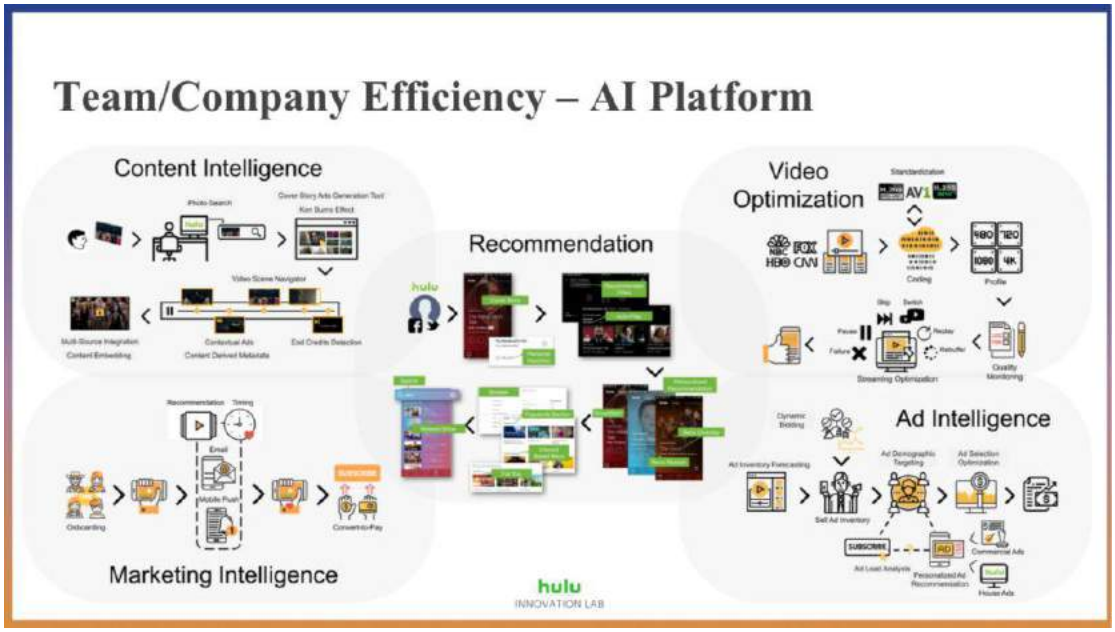
DataFunTalk



“你不是一个人在战斗”

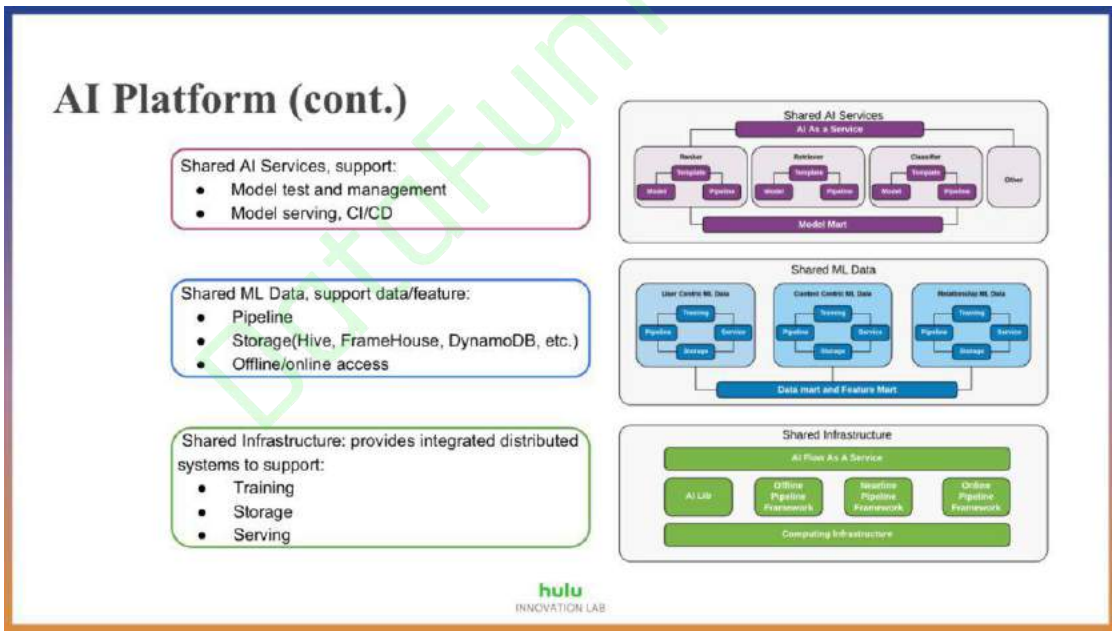
平台、架构对于 Hulu Research 研发效率的长远影响

关于开发效率可以分为两类，一类是团队的效率，另一类是个人开发效率。

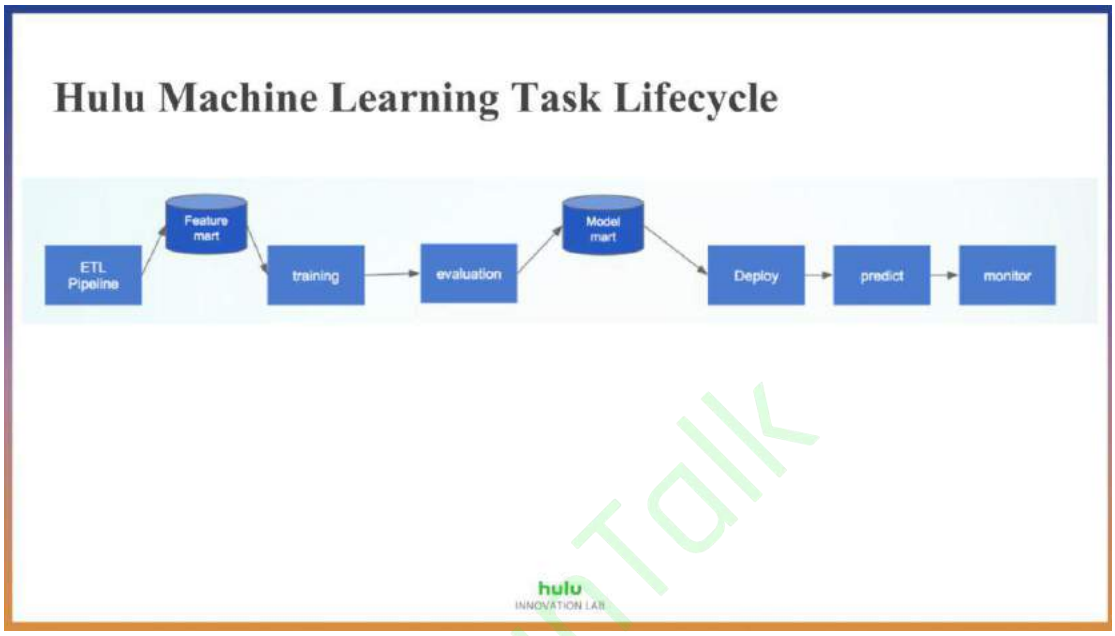


1. 团队效率：

Hulu 有五个研究团队，团队间更多时候是相对独立的，各自都在做各自的项目。但事实上，团队之间有很多数据、特征、模型其实是可以复用的。在此基础上，为了避免大家做重复性的工作，我们构建了 AI Platform 架构，可以为开发团队减轻重复造轮子的压力，也让研究团队能将更多精力关注于模型优化上。

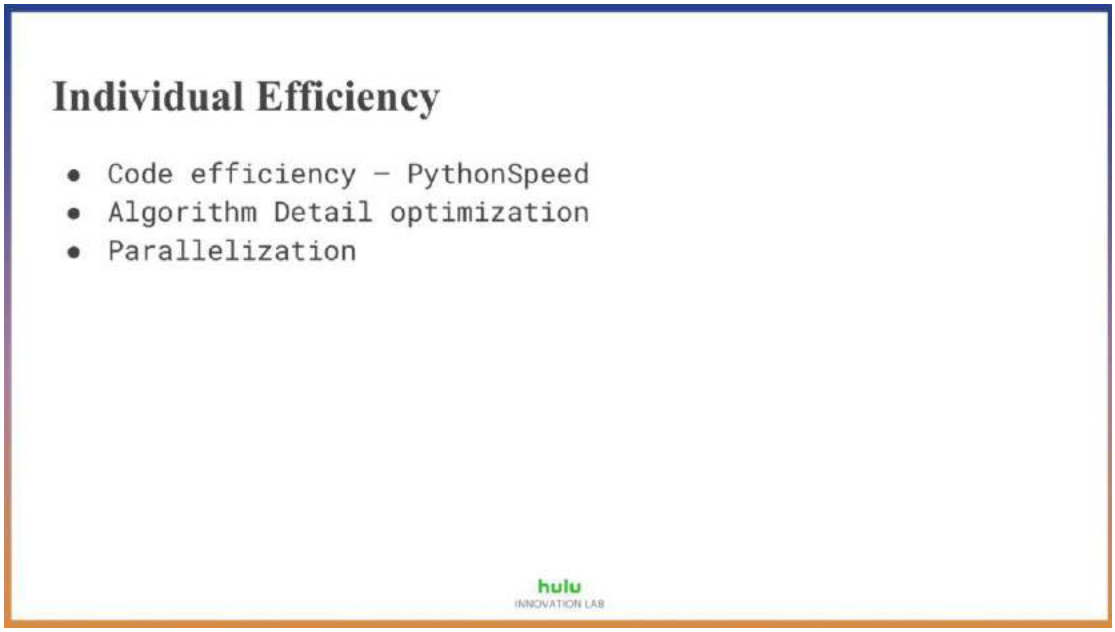


AI Platform 分为三层，第一层为基础架构层，第二层为机器学习数据层，第三层为 AI 服务层。基础架构包括对原始数据的存储、分布式的计算资源 GPU 的调度，分配不同节点进行机器学习、深度学习的任务等。ML 数据层存放加工之后的机器学习原材料，比如 Content Embedding 就会存放在这里，以便日后各个 team 的使用和模型的迭代优化、版本控制。第三层服务层的构建是为了让模型更好地服务于线上产品，使团队能更好的管理、发布和监控模型线上的运行情况。



在 Hulu 的整个机器学习任务周期中，一般流程为：建立数据流 pipeline，自动化将数据取出放入特征仓库，然后训练模型、进行评估，将稳定的模型版本存放在在模型仓库中，之后部署到线上通过 AI 服务层进行预测和监控等。

DataFunTalk



2.个人开发效率：

在公司中个人的工程实践能力也非常重要。个人工程化能力的提升是一个日积月累的过程：

A. 代码效率：在开发过程中我们需要考虑编写的代码是否高效，比如在 python 中选择哪个库？用哪种形式、哪种写法的函数会使程序效率更高。

B. 算法优化：从算法角度提高代码效率，类似 Word2vec 相对 NNLM 采用的 Hierarchical Softmax 等方式对算法进行优化，极大提高了训练效率。

C. 并行化：在公司进行项目开发，要灵活地运用大数据基础架构等底层服务对代码进行并行化，比如数据读写、训练模型时都需要考虑能否以及如何进行并行化，这样可以大幅度缩短项目的开发和迭代周期。

人机交互式机器翻译研究与应用

嘉宾：黄国平 编辑：赵文娇

这个题目看起来比较小众，希望大家听我的讲座没有白听，可以学到一些东西，就是AI 技术落地过程中的一些问题，以及解决策略和方法，技术、产品、行业问题的大杂烩。

我的分享主要包括以下几个方面：

1. 机器翻译现状简介；

2. 翻译需求与人工翻译行业；

3. 人机交互式机器翻译技术；

4. 人机交互式机器翻译应用；

5. 人工智能落地的一些思考。

首先会过一下机器翻译的现状，看一下翻译需求是怎么回事，然后就是我本身做的事情，还有做这个事情过程中遇到的哪些问题，以及这些问题带来的思考。

机器翻译现状简介：

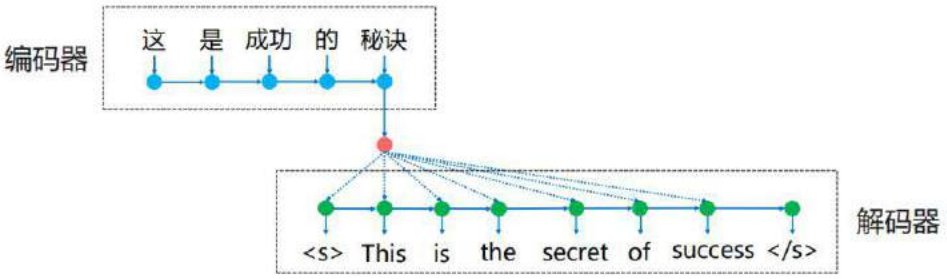
机器翻译在 2018 年不想听到这几个字也很难，有的公司说起人工智能好像也不知道做什么，那就做一个翻译机或者同传吧，如果没有技术怎么办，没有技术的话，你也可以做机器翻译，怎么做呢？做人机耦合吧，把人关到同传箱里面，然后翻译内容不由机器来生成，由人进行生成，把人工翻译结果打印到屏幕上。翻译其实是比较低频的需求，但是在 2018 年很多公司希望在翻译上展示出自己不一样的能力 ,表现出很饥渴的样子，

大家很多时候或多或少有这样的疑问，当大家真的需要用机器翻译的时候，往往错误百出，在同传的时候，在博鳌论坛上，翻译连续翻译出 10 几个神奇的句子，这种神奇诡秘的地方，对于机器翻译来说，这不是伪需求吗？谁会花几千块，大几百块买那么一个破玩意儿，但是各家说自己出货又很高，我一直没有想明白，为什么那些翻译机的销量能有几百万，几千万台，然后各家的数据一综合的话会超过亿，可是在我身边却看不到呢，我们 AI 行业各家的数据都是谜一样的存在。每家都在宣传自己掌握核心技术，通过开源软件知道了机器翻译的原理，出来的东西也都差不多，一句诗来形容就是“藕花深处田田叶，叶上初生并蒂莲”。

国内最近 20 年，长期投入的机器翻译的单位大概有这么几家，有的资深一些，有的比较新一些，比如说腾讯在机器翻译上的投入没有超过 3 年，超过 10 年快 20 年的是前面几个单位，比如说中国科学院自动化研究所（也是我读博的地方），还有中国科学院计算机技术研究所，除此之外，清华，哈工大，这些时间也比较长，苏州大学比较年轻的，最近几年在人工智能上的成绩是有目共睹的，公司的话应该是百度和网易有道，其他的相对晚一些。机器翻译的从业人员，应该都是从这些机构走出来的，毕业的或者跳槽的，思维趋同，做出来的产品大差不差。

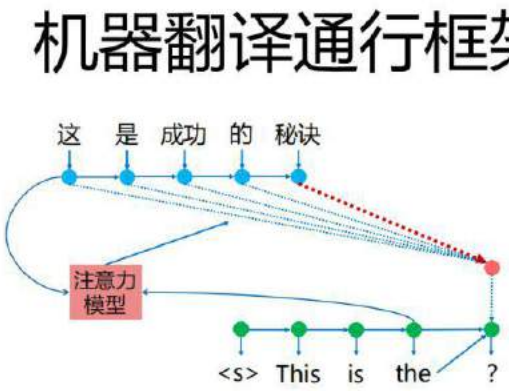
DataFunTalk

机器翻译通行框架



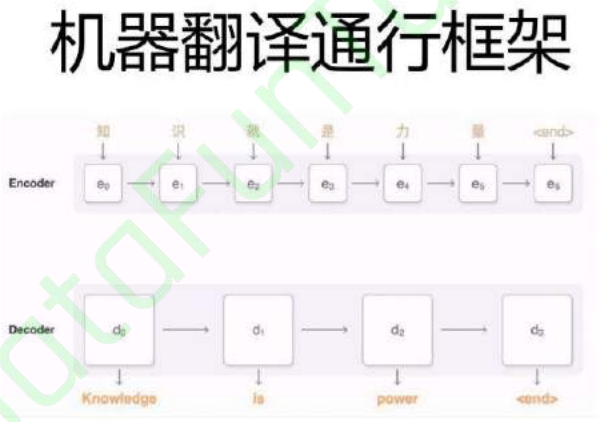
机器翻译“编码器-解码器”结构

机器翻译用的框架还是雷同的，主要是细节处理或者是针对一些问题本身的改进。通行的结构就是编码器解码器结构，做深度学习的对这套框架应该是很熟悉的，最难用的就是先把原文句子经过每个字词通过 RNN 之类的手段对它进行编码，最后形成一个句子向量，在解码端，从句子开始根据句子向量，生成空格段的词，最难用的框架，问题可能是比较多的：比如它的基本原则很简单，在解码器一端从句子开始到句子结尾处，它才停止翻译的过程，如果没有遇到句子开始和句子结尾的话就一直进行翻译过程。大家很容易想到有些问题：比如在这种框架中，有些片段没有翻译到，形成丢失，或者是生成的译文和原文没有什么关系。大家实际中在使用线上翻译产品中会有这样的感知，虽然现在的机器翻译和三四年前相比已经有了质的飞跃，但还是不合格的，仍然有很多问题，这时如果需要弥补这些问题，一般都是加模型。



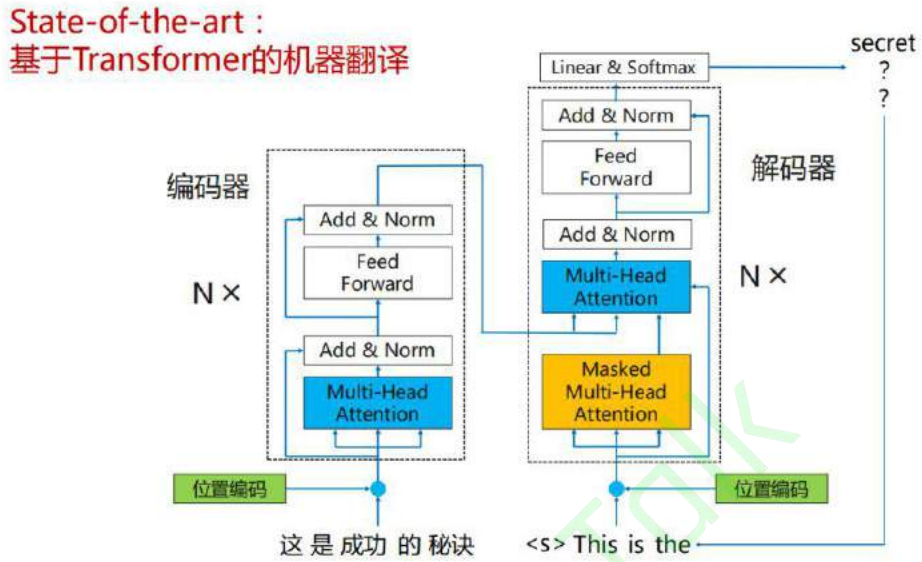
采用注意力机制的机器翻译

最简单的就是注意力模型，加注意力模型的本质需求是希望我生成的下一个目标词的时候能与原文的某些词或者某一个词建立比较强的联系，并通过注意力模型，来生成随翻译过程不断变化的上下文向量。它根据原文编码，加注意力模型之后呢，它会随着前面已经生成的词，进行变化，这个时候翻译效果会上升一大截。



采用注意力机制的机器翻译

如果用固化的方法来讲，注意力模型怎么去解释它的物理作用呢？在翻译这个问题下，它看起来有点像翻译的概念，就是词到词之间的翻译，但是好多词又不一定，在使用线上翻译的时候，输入原文句子里面多一个标点，少一个标点可能引起译文端的剧烈变化，这个时候，这个时候注意力就是学偏了，它把好多东西都压到了原文最后那个标点上面。所以我们人工单相思式的去解释翻译概念是不对的，既然注意力机制与翻译概念不是那么有关系的话，那就换一下，比如谷歌发表了 transformer 的架构。

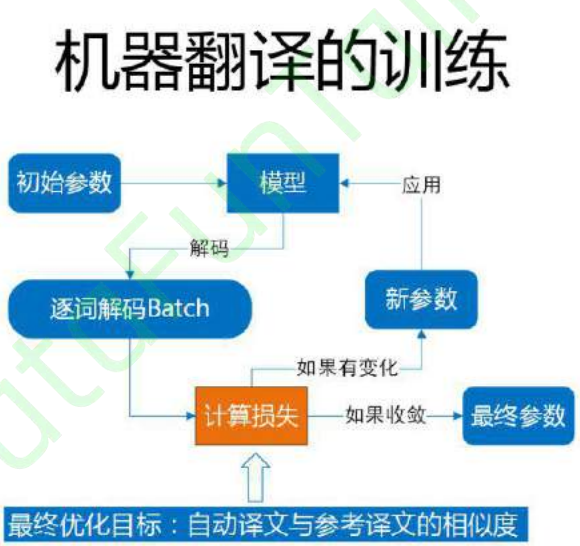


参考文献: Attention is all you need

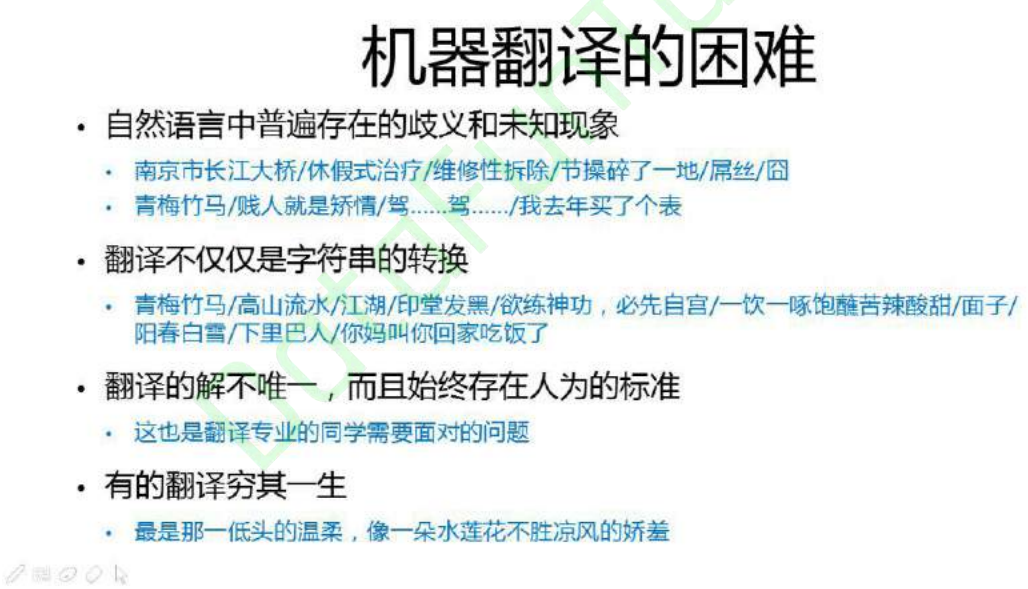
大家如果做这个方向应该是比较熟悉的，不做这个方向的，要理解的东西也比较多，相比 RNN 结构，transformer 这个结构相比 RNN，对原文信息的编码是更充分的，比如说是在 RNN 的时候，对原文的编码都是从左至右或者从右至左，但是在 transformer 这一套框架中，它对原文中每个词的编码都会参考整个句子的词，而且编码还不止一层，有个 self-attention 机制，一般我们用 attention 都是原文和目标端文字之间的关系，但是在 transformer 关系中它会计算原文和原文之间的关系，然后译文与译文之间的关系，这样就会更充分了，但是参数规模会爆炸，所以也出现了比较尴尬的事实，

transformer 在比赛中，如果 GPU 状态比较差的话，是跑不了实验的，比如我们实际场景上线，就是训练语料是以亿句为规模的，想通过 1080，或者 2080 那是肯定跑不出来的，因为要跑好几个月，就算是 P40，一块 P40 也没什么用，就是它的单个显存比较大一些，比如说 24G 左右，因为在面对海量数据的时候，我们一个模型，跑一个模型 48 块盘，48 块 P40，如果大家是玩一玩或者创业公司来做这个事情的时候，会有很多的困难，也许以后会随着云计算业务的普及，大家会受惠，但是短期而言的话，在做 AI 技术落地的时候，硬件对于创业团队容易造成很大的经济压力。

transformer 对语言的编码不是单向的，每个词每一轮的编码都会参考上一轮的结果，这是 transformer 显著区别于 RNN 的区别，编码器如之前所言考虑句子中所有词，解码器默认是从左到右，从句子开始生成下一个词，在生成下一个词之前，对之前解码的内容进行多轮解码。



机器翻译的训练和大家做其他的训练也没有什么两样，都是随机初始化，或者按照某种预设方式初始化参数，然后把它加载到模型中，对训练语句进行 batch 数据，计算损失，调参数，重复这个过程，直到最后收敛，把参数保留下来，没有什么不一样的地方，只是需要注意计算损失这一步。因为一般而言在翻译领域，如何去评价一个自动译文好不好呢，我们的评价指标叫 blue，它是和参考译文去比对，一元的串匹配的有多少，二元的串匹配的有多少，三元四元有多少。但是在神经网络翻译训练当中，我们是没有办法用 blue 这种方式的，这时候就用 PPL，用的比较广泛，虽然在统计规律中讲 PPL 越低，blue 值会增的，但是好多时候，在一些比较狭小的空间，它会有个反馈，所以这个时候会有个模型选择。这个模型选择一般是构建一批比较大一点的人工测试集，就是面向开放领域，或者把大家的高频 query 抓下来，埋点，然后中间会生成一堆 checkpoint，然后看哪个更能体现线上实际要求，这个过程和做其他产品没有特别大的区别，只是做数据处理充满各种小的技巧。



比如说在做产品的过程中产品还没有发布，买数据的钱花了好多，然后人工评估体系的构建又花了好多钱，最后都不知道这个产品能赚几个钱。（注：当然我是很有自信的，我的产品能把产品的成本赚回来了。）现在各家都在炒作自家产品，其实用的技术没有差太多，如果用相同的语料去训练各大公司的机器翻译引擎的话，大家 blue 值的差别不会超过 3 个，区别其实会很小很小，大家为什么又拼命强调其中的差异化呢，其中最主要的原因就是：目前，翻译的问题还没有被彻底解决，大家还可以找出很多差异，但是大趋势是趋同的。机器翻译的困难呢，大家在做的过程中会有一些体会，我这里总结了四个方面：

第一个方面就是歧义和未知现象：歧义的，像南京市长江大桥这种都老掉牙了，未知现象的，比如是在武侠小说里面驾。。。驾。。。（模仿骑马的这种）， 再比如新产生的词不太容易被翻译出来。

第二个方面是翻译不仅仅是字符串的转换：直接翻译不能解决所有问题，比如，青梅竹马，直接翻译成英文是“发青的梅子，竹子做的马”；特定情况下的江湖（比如有人的地方就有江湖），直接翻译过去就是“江和湖”，其实并不能表达中文中的语义；再比如特定场景下的“你妈妈叫你回家吃饭了”，都没法直接逐字进行翻译。

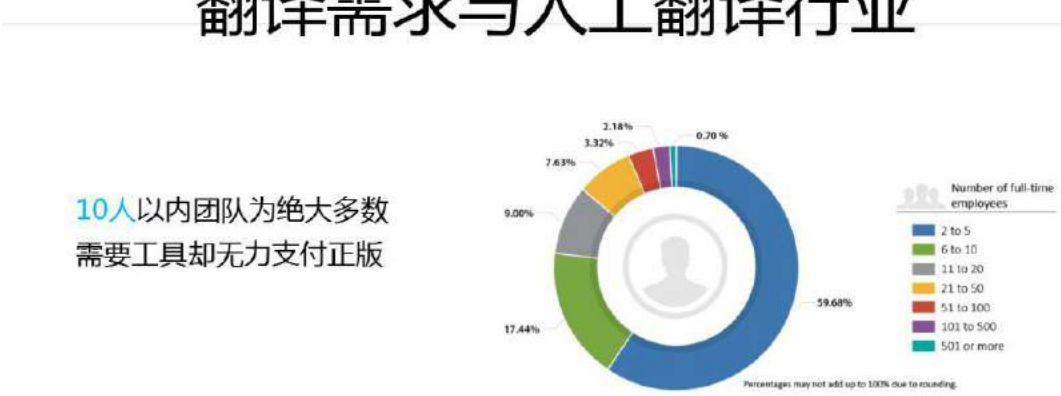
第三个方面就是翻译的解并不唯一：缺乏量化标准，并且始终存在人为标准，这个是精准层面的，把一句话表达清楚，其实有很多方式。

第四个方面就是翻译的高度：有的语句意境很高，上升到文学层次了，比如“最是那一低头的温柔”，翻译的时候涉及到意境的体会和把握，对于人工翻译来说困难，对于机器翻译来说，也困难。

翻译需求与人工翻译行业：

翻译行业全球的产值还是很高的，大约四到五百亿人民币，主战场在欧洲和北美洲，最大增速在非洲，中国占 10%左右，国内主要的语种是中英互译，利润主要来自于外单。

独角兽不独，充分多样性。其他领域独角兽往往比重很大，占据 70%左右的市场，但是翻译市场不是这样。从 2005 年到 2016 年头部公司的排名和占比区别不大。



10 人以上团队需要工具却无力支付正版。

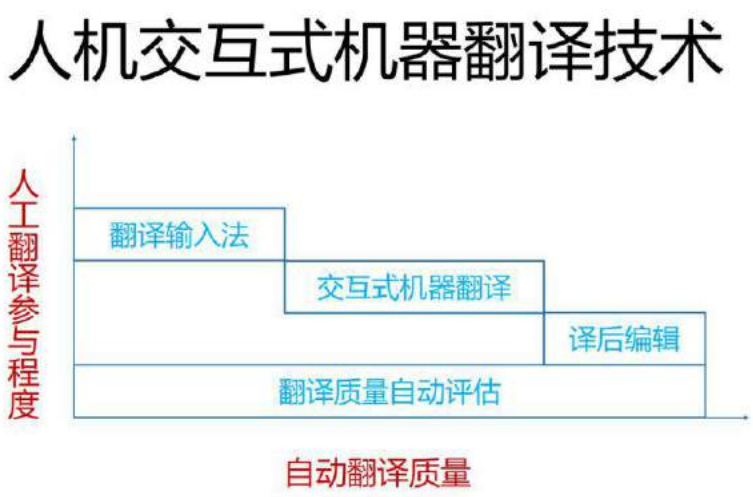
人机交互式机器翻译技术：

背景：全自动输出翻译结果无法保证译文质量，这时候就需要人机交互，实现下面三个任务：

1. 在翻译过程中接受用户提供译文干预，能根据干预结果生成更好的结果；

2. 及时学习用户的修改反馈，不能一个愚蠢的错误反复出现；

3. 实时提供翻译辅助信息。



人工翻译如何和机器翻译产生交互？人工翻译参与程度与自动翻译的质量相关，如果自动译文的质量较高，那就机器翻译直接发挥作用了（人工只需要在自动译文之后进行校验），如果自动译文的质量比较低，就需要比较高的人工介入，像翻译输入法（相当于比较多的人工标注），如果主动译文的质量不好不差的情况下，可以用交互式机器翻译来解决翻译质量的问题（人工标注比较难的部分）。

机器翻译在翻译中的角色

在自动翻译过程中，机器翻译可以扮演的角色：

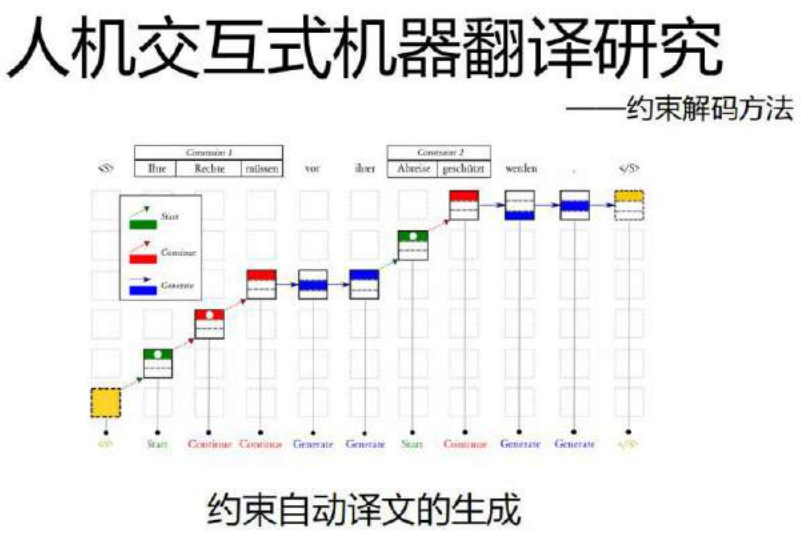
整句的更新（高精度的翻译整个句子）、片段提示（在无法保证整句精准的情况下保证重点片段的精准）、翻译输入法、在线学习、语义理解、快速解码。

在整个过程中，除了 AI 技术，还涉及到产品设计能力。需要产品经理和技术人员之间的协调和共识。

DataFunTalk

最简单暴力的使用机器翻译的一种，就是译后编辑。就是首先机器翻译，然后人工编辑。

机器翻译不差不好的情况下使用交互式机器翻译，就是人工翻译片段和机器翻译片段交替进行。



约束解码方式，就是强制在翻译中必须含有一部分术语，比如说一句话中必须把“深圳腾讯”，“社区”包含在内，这一点是目前难以做到的，主要是卡在翻译速度上，时间消耗严重。

我们对算法进行了一些改进，如上图所示。

人机交互式机器翻译技术

——翻译输入法

源文句子:

China mulls change to officials' welfare system

最终译文：

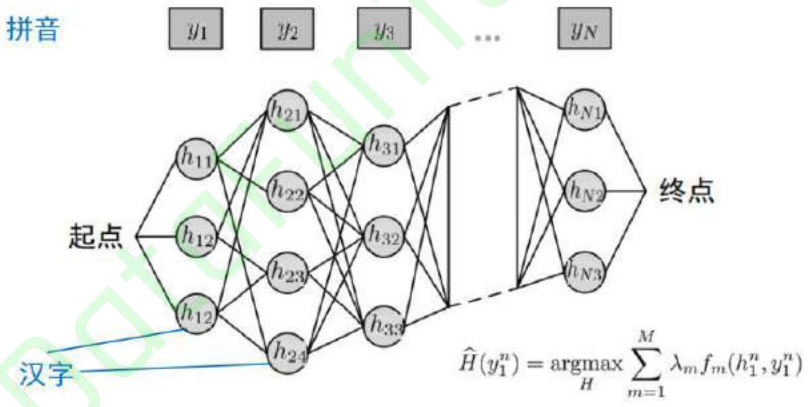
中国考虑改革公务员福利制度

谷歌输入法

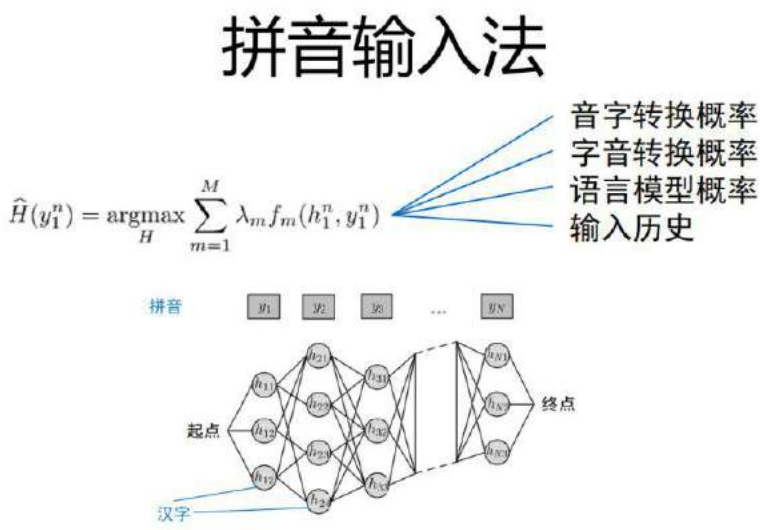
微软输入法

我们希望用户打完这句话用的按键数越少越好，用谷歌和微软输入法结果如下，因为一般的输入法去学习的是人的输入历史，我们引入机器翻译的东西可以改进过于依赖输入历史的情况。

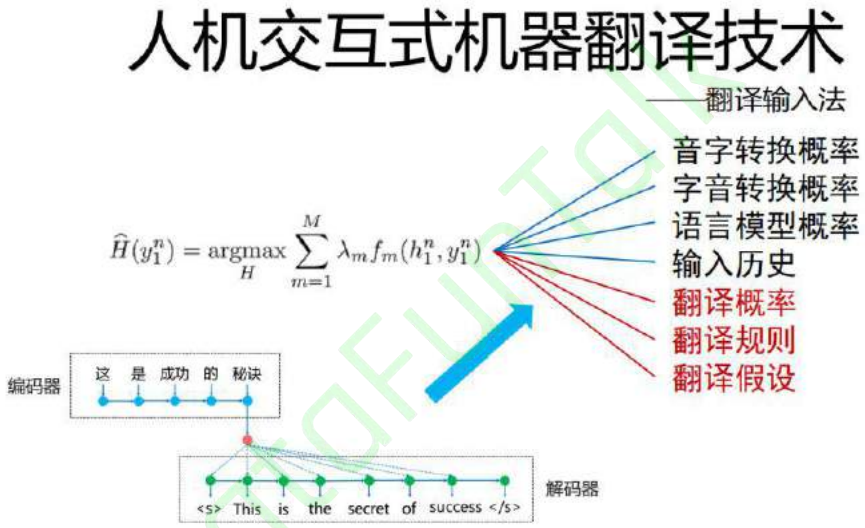
拼音输入法



DataFunTalk

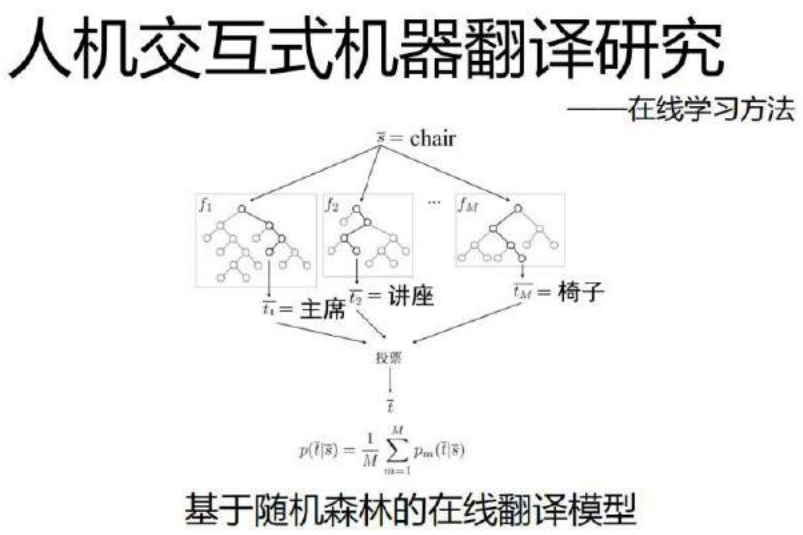


拼音输入法：是寻找最优解的过程，需要对过程进行打分，之前主要是音字转换概率，字音转换概率，语言模型概率，输入历史。这里我们可以进一步引入翻译技术。



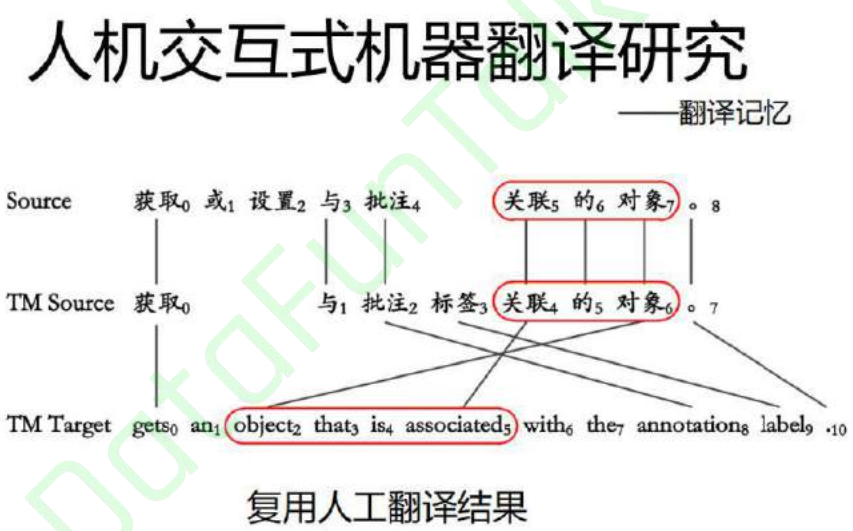
人机交互式机器翻译研究-在线学习方法

在线学习，就是 online learning，即时学习用户反馈，一般强化学习，增强学习比较多，我们采用的是基于随机森林的在线翻译模型，是一种投票机制。



基于随机森林的在线翻译模型

根据信息增益决定决策树上某一支是否要分叉，同时根据当前的错误率决定是否删除某棵树，方法比较简单，主要是基于解决实际问题以及当前可获得的有效资源的角度进行考虑，要找到适合当前环境的方法。



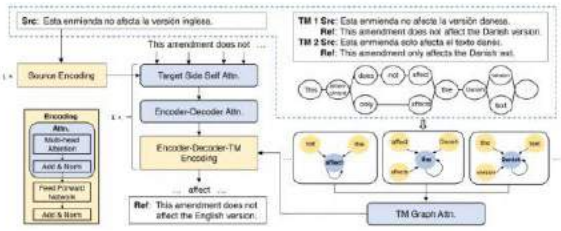
复用人工翻译结果

比如说需要翻译的句子 A 如下：“获取或设置与批注关联的对象”但是之前翻译过这样的句子包含了与 A 的公共子串“关联的对象”，这样就是如何复用以前的翻译结果？这

就是我们需要找到简单、高效、适合当前团队实现能力的方法，我们的方案是容易翻译记忆的 NMT（Transformer）如下图：

人机交互式机器翻译研究

——翻译记忆



融合翻译记忆的NMT（Transformer）

Similarity	Sents	Percent	TFM	P-TFM	G-TFM
[0, 0.1)	2	0.19%	54.17	68.43	58.22
[0.1, 0.2)	138	11.40%	36.08	36.46	37.49
[0.2, 0.3)	462	20.76%	44.94	45.99	45.30
[0.3, 0.4)	305	14.56%	51.19	51.98	51.75
[0.4, 0.5)	272	12.83%	60.72	62.86	62.10
[0.5, 0.6)	206	7.82%	66.31	66.41	70.66
[0.6, 0.7)	203	7.40%	71.38	73.73	76.17
[0.7, 0.8)	188	7.05%	77.48	78.78	83.37
[0.8, 0.9)	377	10.90%	83.81	85.52	88.71
[0.9, 1)	432	7.09%	87.81	88.02	93.21
[0, 1)	2585	100%	62.68	64.30	66.21

翻译质量对比（BLEU）

不一定最新的论文就能解决当前团队下我们的问题，很多东西还是要结合实际去思考适合自己的，最后的效果不一定差。

人机交互式机器翻译应用：

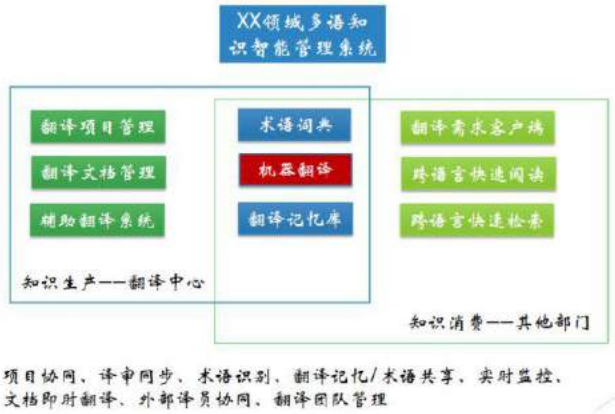
DataFunTalk

人机交互式机器翻译在腾讯

这里主要说人机交互式机器翻译在腾讯的主要应用场景，分为内部场景和外部场景，腾讯不同的方向有不同的翻译团队，进行赛马机制。如下，同声传译，拍照翻译，辅助翻译。技术问题落地到实践是一个问题，大公司和创业公司的方向可能不一样，大公司可以内部落地也可以外部落地，创业公司只能外部落地，内部落地不一定意味着产品能够解决问题，外部落地就很现实，所以如何外部落地，能揽到更多的外部项目是一个值得思考的问题。

DataFunTalk

人机交互式机器翻译应用



工业场景对翻译的需求还是很大的，比如华为、中兴、南车、北车，他们一般有自己的翻译部门，他们是有这个需求，实际上他们需要根据他们自己的业务背景，进行一些技术定制，客户就是大爷，我们听他们的，虽然有时候感觉这些需求和高大上算法没什么关系，但是没有客户的肯定，算法是没有办法用的，最后经过挺长时间的行业调研，一开始的时候定制的比较麻烦，后来就习惯了。

如何构建人机交互翻译系统？

- 明确的需求
  - 语种、领域、用途、业务系统接口
- 最相关的语料
  - 200万句对以上
- 集成可用的state-of-the-art技术
  - 输入法、术语抽取、翻译片段挖掘等
- 调试GPU集群、并行加速训练
- 部署上线并迭代

如何把技术变成产品，这里有许多路要走，因为用户往往也不知道自己需要什么，为了避免一系列莫名其妙的需求：

第一步要做的就是需要有明确的需求：比如语种、领域、用途、业务接口。

第二步就是确定合适的语料，从哪里买，语料质量怎么样，价格合不合适，能不能压价等，一般都要 200 万句对以上。

第三步是集成可用的 state of the art 技术：输入法，属于抽取，翻译片段挖掘等，很多时候是有开源工具的，但是开源工具基本上都是有坑的，很多时候面临着自己开发和用开源工具的问题，目前我们是自己开发，因为熟悉开源工具的时间比我们自己开发还要久，但是不同的团队有不同的情况，开源工具容易上手，能拿过来就用。

第四步就是调试 GPU 集群，并行加速训练。第五步是部署上线并迭代。

难点在哪里？

人们对于自动译文质量过高的期望、开源系统的坑远比计划的多、如何甄别并且集成最新技术、如何拉通目标场景技术链条，关于最后一点拉通目标场景的技术链条，因为发论文是将问题聚焦到一个点上，而做一款产品的时候，流程和目标设定上环节比较多，链条比较长。

人工智能落地的一些思考：

团队需要什么样的人？需要调参大师、论文机器、代码工匠，说起来似乎论文机器发挥的作用最小，其实也不是这样，如果是创业公司，老板需要拿最新的论文去忽悠钱，没

有论文机器是不行的，调参大师和代码工匠是必须的，三者的关系需要高超的管理技能进行协调这是内部的问题。

外部的问题有“数据鸿沟”和“工程壁垒”，纯技术团队是没有业务数据的，业务数据又特别宝贵，如何通过沟通把宝贵的数据弄到手是一个技术问题；另外一个是在算法技术同质化的情况下，需要很细致的步骤安排大量的数据清洗和工程问题，这些都是大家比拼的实力。

最后在设计产品的时候，理念选择上，设计一个“人工智能产品”还是一个“用人工智能的产品”，未来的趋势一定是人工智能产品，但是目前我们还是要帮助产品设计出更合理的人工智能的产品。

DataFunTalk

百度中文纠错技术

付志宏 李润顺

本次分享聚焦于自然语言处理的一个传统问题 - 文本纠错，首先介绍文本纠错的相关背景及目前的主流技术，然后重点分享百度在文本纠错方向的主要工作，最后结合具体应用场景，展示纠错技术带来的产品体验升级。

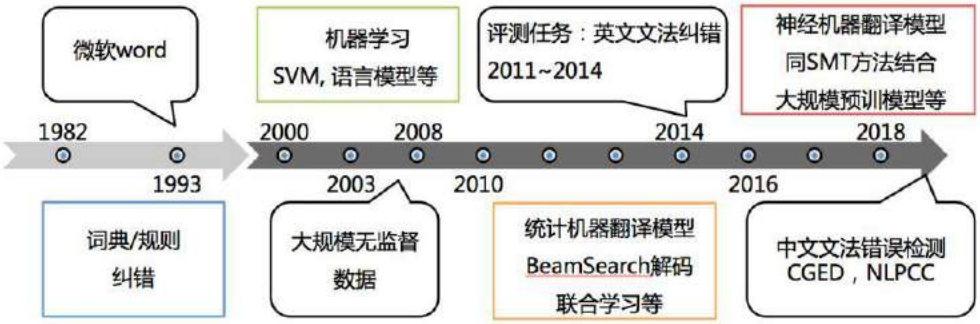
一．纠错技术概述

语言是复杂的。每一门语言都经历了几百年，甚至几千年的长期演变和发展，形成了一套复杂的文法和句法规则。这些文法和句法规则复杂多变，例如一些词或者短语存在多音、多意、多用等现象，对语言的使用者提出了较高的要求；一旦语言使用者对语言掌握不够或者粗心大意时，则很容易发生用词不当、张冠李戴等错误。虽然这些错误看起来微不足道，但正所谓“差之毫厘，谬以千里”，特别是在某些场景（比如外交场合），即使很小的语言错误也可能带来非常恶劣的影响。

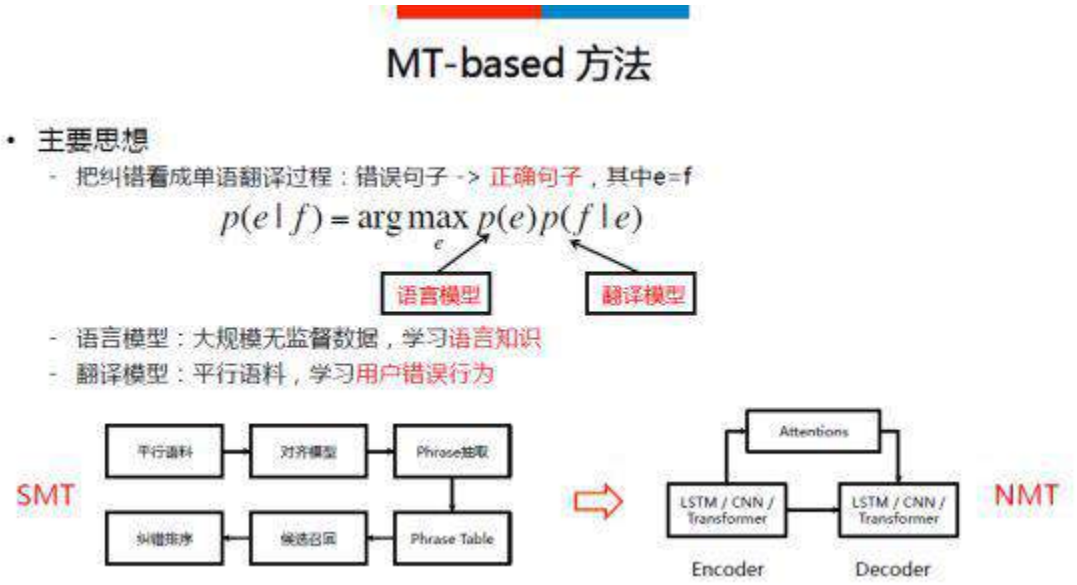
自然语言处理常见的任务包括词法分析，句法分析，语义计算等，这些任务要取得理想的结果，输入数据的准确性是基本前提，因此从 NLP 的整体技术角度来说，文本纠错起着保驾护航的作用。

DataFunTalk

纠错 - 发展历史



纠错技术相对于词法分析，句法分析等受到的关注一直较小，一方面是因为文本出错的比例比较小，在一些重要场合，也有专门人员进行校验；另一方面本身问题也相对较难，其要求计算机对语言规则以及文本语义有深刻的理解。在 2000 年以前，业界主要依靠长期积累的纠错规则和纠错词典来进行纠错，比如微软的文档编辑产品 WORD 即采用这种方法。随着机器学习技术的发展，纠错问题受到了学术界和工业界越来越多的关注，其中有两大主流方法：一种解决思路是将语言错误归类，然后采用 Maxent、SVM 等分类方法对这些类别进行重点识别；另外一种思路是借鉴统计机器翻译(SMT)的思想，将语言纠错等价为机器翻译的过程，即错误文本翻译为正确文本，并随之出现了一系列的优化方法。最近几年，随着神经机器翻译(NMT)技术的快速发展，人们逐步将 SMT 与 NMT 技术结合起来解决纠错问题。最近几年中文纠错的研究也得到较多的关注和发展，并陆续举办了几届中文纠错评测，例如 CGED 与 NLPCC 等。



目前，基于机器翻译的方法已经成为文本纠错的主流技术，主要思想是把纠错看成同种语言中错误句子翻译为正确句子的过程，如上图中的公式所示，其核心由语言模型和翻译模型组成。语言模型学习语言规则、语言知识；翻译模型从平行语料中学习用户的纠错行为。SMT 纠错一般的典型做法：首先基于平行语料训练对齐模型，得到多粒度（字、词、音、形、短语）的混淆矩阵（Phrase Table）；针对具体的纠错实例，基于句子中的错误点从 Phrase Table 中召回可能的正确候选，然后基于句子语义理解，对这些纠错候选进行排序，从而得到正确的结果。

而 NMT 方法主要依赖于大规模的监督语料，忽略掉中间的各种步骤，直接实现端到端的学习。NMT 方法相对于 SMT 方法的优势在于避免了 SMT 每一步过程中导致的错误传递，同时具有更强大的学习能力。

二． 百度中文纠错

▪ 技术背景

近年来，随着新媒体行业的快速发展，中国自媒体从业人数逐年增长，至 2017 年有近 260 万。但是相对于传统媒体，其缺少人工校稿环节，编辑好的文章即刻发表，导致文章的错误比例较高。比如一些新媒体平台的正文错误率在 2%以上，标题错误率在 1%左右。同时，语音智能硬件产品的兴起，也暴露出语音识别技术的错误率高企问题，在某些场景语音识别中，错误率可能达到 8%-10%，影响了后续的 query 理解及对话效果。因此研发优质的中文纠错技术，便成为了必须。

▪ 技术目标

为了满足以上的需求，百度中文纠错一方面需要支持多种类型的错误识别，另一方面需要支持不同模态的输入数据，同时还需要提供快速的场景迁移以及深度定制能力。

我们把中文常见错误总结分为三类：

❖ 用词错误，由于输入法等原因导致的选词错误，其主要表现为音近，形近等；

- 文法/句法错误，该类错误主要是由于对语言不熟悉导致的如多字、少字、乱序等错误，其错误片段相对较大；
- 知识类错误，该类错误可能由于对某些知识不熟悉导致的错误，要解决该类问题，通常得引入外部知识、常识等。

▪ 整体架构

百度中文纠错的整体架构如下图所示。整体上，将纠错流程，分解为错误检测、候选召回、纠错排序三个关键步骤。通过引入语言知识、上下文理解和知识计算的核心技术，提升不同类型错误的解决能力。最后，支持 SMT based 和 NMT based 两套 Framework，形成完整的系统架构。下面分别进行详细介绍。

整体架构

▪ **关键步骤**（错误检测->候选召回->纠错排序）

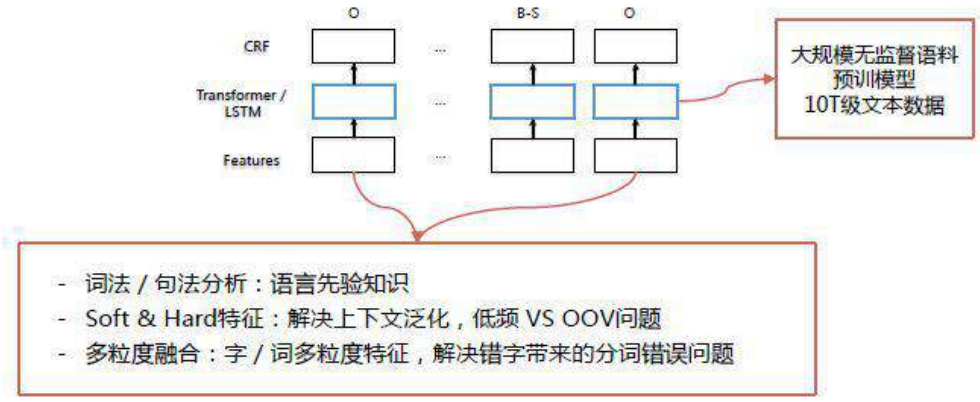
错误检测的目标是识别输入句子可能存在的问题，采用序列表示(Transformer/LSTM) +CRF 的序列预测模型，这个模型的创新点主要包括：

- ❖ 词法/句法分析等语言先验知识的充分应用；
- ❖ 特征设计方面，除了 DNN 相关这种泛化能力比较强的特征，还结合了大量 hard 统计特征，既充分利用 DNN 模型的泛化能力，又对低频与 OOV（ Out of Vocabulary ）有一定的区分；
- ❖ 最后，根据字粒度和词粒度各自的特点，在模型中对其进行融合，解决词对齐的问题。

DataFunTalk

错误检测

- 序列预测：序列表示 + CRF



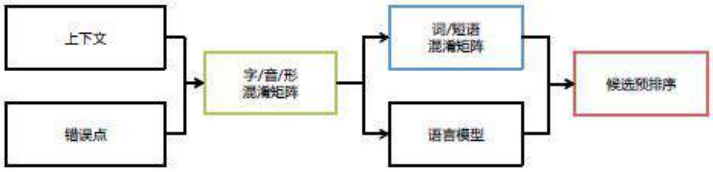
候选召回指的是，识别出具体的错误点之后，需要进行错误纠正，为了达到更好的效果以及性能，需要结合历史错误行为，以及音形等特征召回纠错候选。主要可分为两部分工作：离线的候选挖掘，在线的候选预排序。离线候选挖掘利用大规模多来源的错误对齐语料，通过对其模型，得到不同粒度的错误混淆矩阵。在线候选预排序主要是针对当前的错误点，对离线召回的大量纠错候选，结合语言模型以及错误混淆矩阵的特征，控制进入纠错排序阶段的候选集数量与质量。

DataFunTalk

候选召回

- 候选挖掘：多来源多粒度

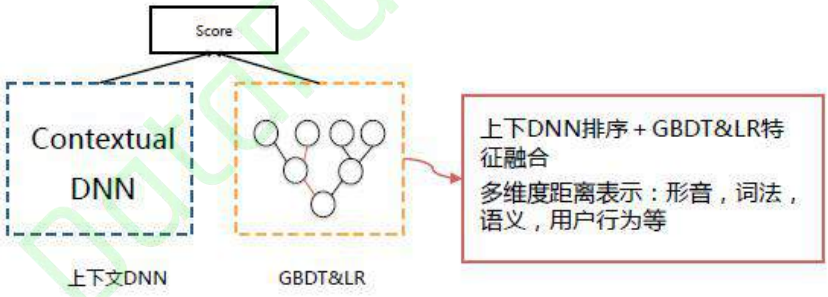
- 候选预排序：宽进严出



纠错排序解决的是，由于纠错的正确结果具有唯一性，如何在召回的纠错候选中将正确的结果排在第一位。百度中文纠错采用的是 Deep&Wide 的混合模型结构，Deep 部分学习当前错误点上下文表示，Wide 部分基于形音、词法、语义、用户行为等特征学习原词与候选词的多维度距离表示，另外通过 GBDT&LR 模型学习到更好的特征组合。

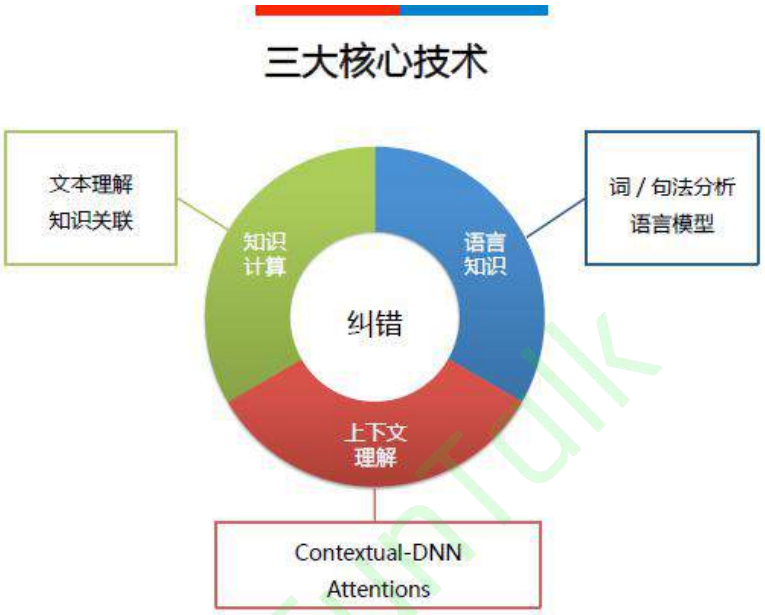
纠错排序

- 混合模型：Deep + Wide

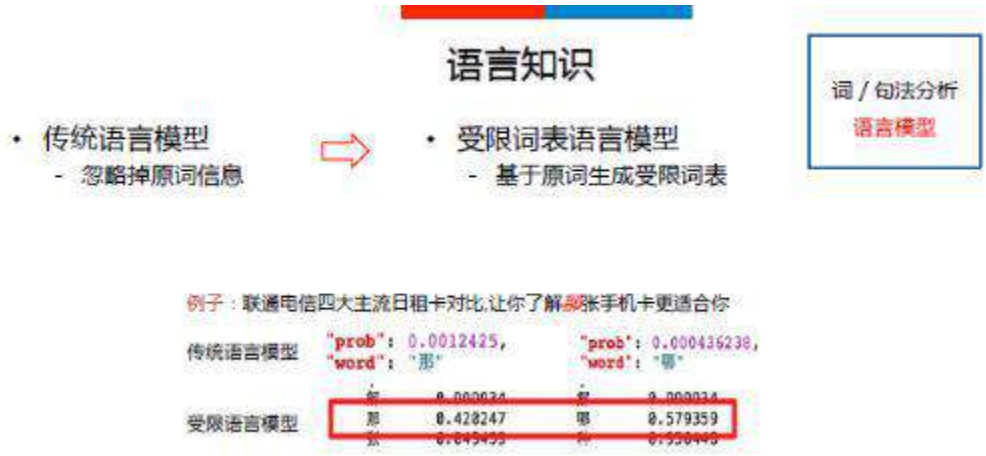


- **核心技术**（语言知识->上下文理解->知识计算）

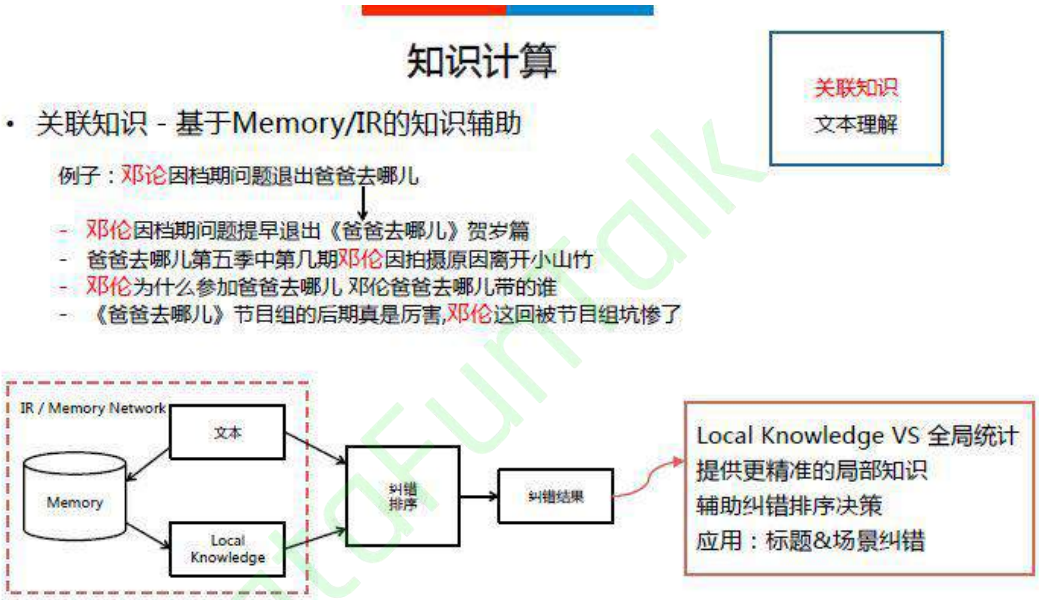
中文纠错需要在错误检测、候选召回、纠错排序的策略层面上取得较好效果，最关键的因素仍然在于解决最基本的自然语言处理问题：语言知识、上下文理解、知识计算。语言知识完成对语言规则的学习，对语言结构的理解，具体包括词法分析、句法分析以及语言模型；上下文理解是指需要理解错误点上下文所表示的内容或语义，百度中文纠错主要通过 Contextual-DNN 模型来学习 ,并通过 AOA attention 机制解决长依赖问题，帮助错误纠正。知识计算方面，从知识（客观规律）的维度考虑问题，重点是做好文本理解与知识关联。



语言知识方面，词/句法分析不详细展开，主要介绍语言模型的训练和使用。传统的语言模型，主要是根据前面一串词预测下一个词的概率，直接应用于纠错的主要问题在于忽略了原词的信息。因此，我们提出了一种受限词表的语言模型，该受限词表基于原词生成，通过基于受限词表的训练和预测，能够对易错词进行更好的区分，具体例子如下图所示。

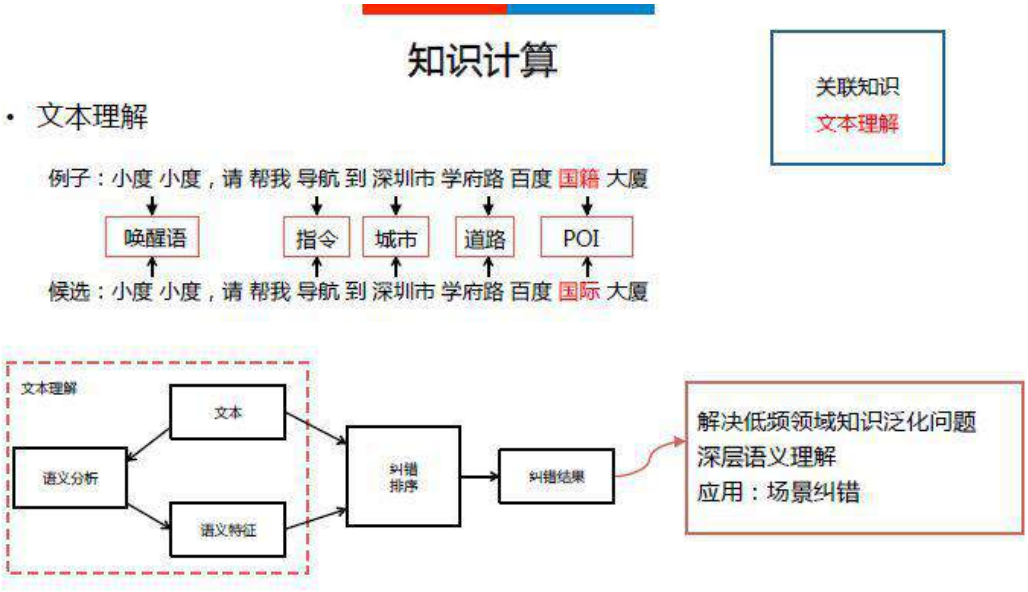


关联知识方面，看下图中的例子“邓论因档期问题提出爸爸去哪儿”，可以通过原始错误标题在标准语料中基于检索（IR）或者上下文 context Memory 的方式，补充到大量同原始标题相关的精准局部知识。利用这些精准的局部知识来辅助纠错排序。



文本理解方面，以用户与智能音箱的交互句子“小度小度，请帮我导航到深圳市学府路百度国籍大厦”为例，可能会出现低频的道路或 POI，如果采用统计得到的语言模型来纠错而不去理解句子表达的内容，显然是不恰当的。需要从全局理解句子内容以及理解

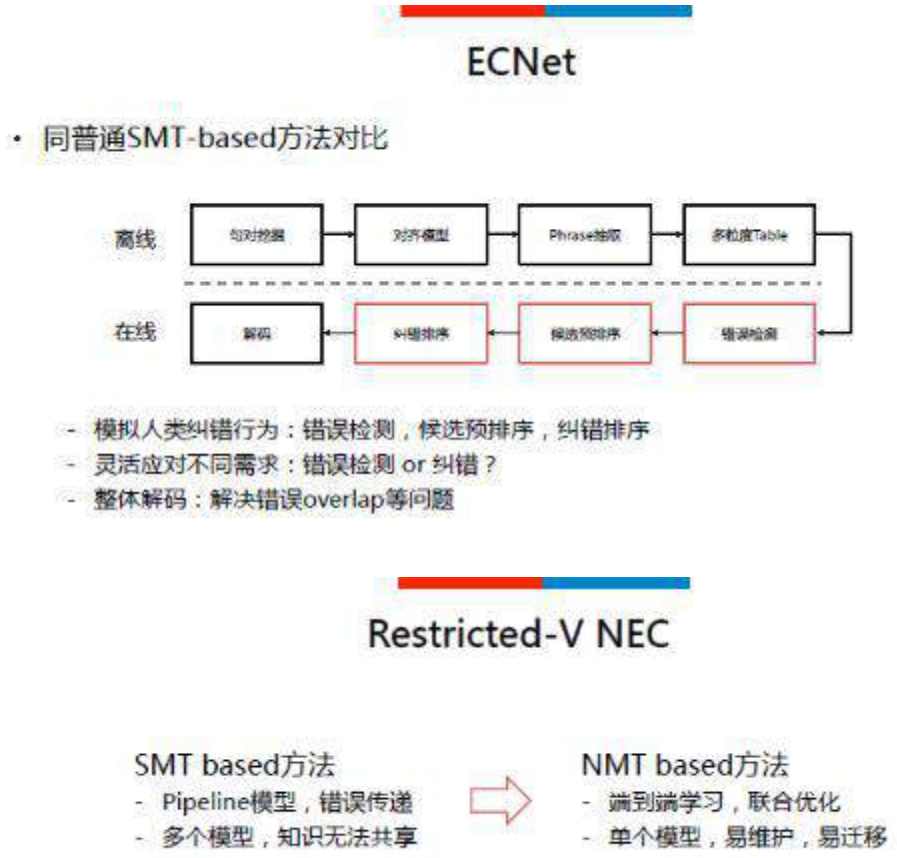
句子的每个成分，解决低频领域知识的泛化问题。具体做法是通过对文本进行语义分析得到语义特征，应用到纠错排序模型中，得到更好的纠错结果。



▪ 系统框架

中文纠错在策略层面涉及错误检测、候选召回、纠错排序，核心技术层面涉及语言知识、上下文理解、知识计算，那如何来构建一个完整的中文纠错系统呢？百度的中文纠错系统支持两种纠错系统框架：ECNet 和 Restricted-V NEC。ECNet 系统把纠错任务分成很多步骤、多个模型，每个模型解决具体的特定问题，然后通过 Pipeline 的方式串联，得到最后结果；这种方式的优点在于方便分析问题，能够针对各个问题进行重点突破，但是也存在两个明显问题：

- ❖ 错误逐级传递；
- ❖ 每个模型单独学习，模型之间知识无法共享。Restricted-V NEC 系统则采用端到端的学习、联合优化，单个模型就能完成纠错任务。



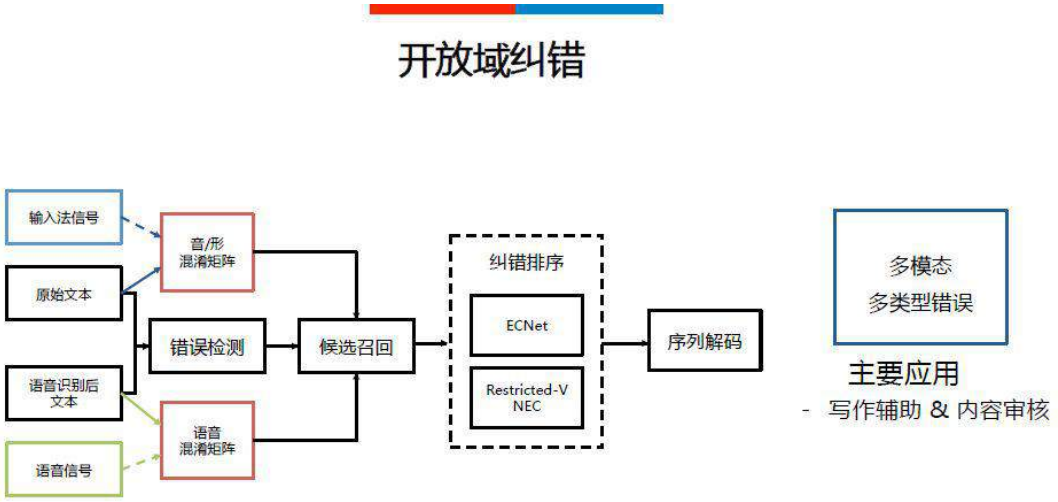
三． 纠错应用

▪ 开放域纠错

开放域纠错没有场景限制，支持多模态输入，可以是文本或者语音，不同的输入形式对应不同的混淆矩阵。整个纠错过程包含错误检测、候选召回、纠错排序和序列解码，支持多种类型的错误纠正。开放域纠错的典型应用场景比如写作辅助和内容审核。

写作辅助：在用户编辑文章的过程中，纠错服务能够及时发现用户错误行为，提升内容创作者的创作质量和效率。

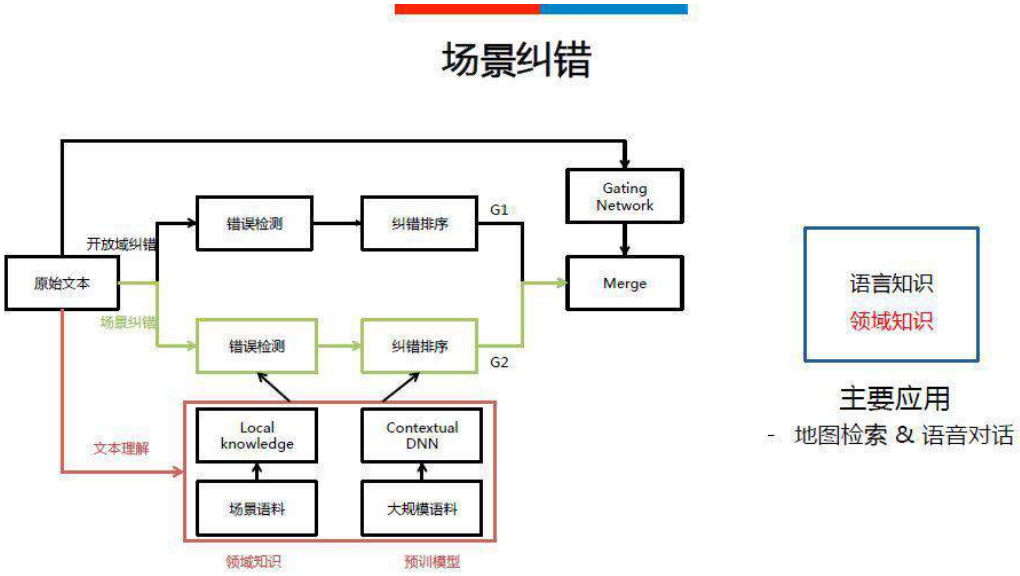
内容审核：对于完稿的文章，纠错服务会对其标题和内容进行错误检测，由专业人员进行二次审核，保证文章质量，提升用户的阅读体验。



▪ 场景纠错

场景纠错，与开放域纠错的主要区别在于领域知识的使用，场景纠错除了做好语言规则的刻画和上下文理解外，还需要对场景中的领域知识有充分的学习。场景纠错的重点是针对输入数据做文本的理解、基于场景语料获取关联知识、基于大规模语料学习语言规则。场景纠错的应用点比如地图检索和语音对话。

在地图检索业务中，通过充分利用 POI、位置距离特征及文本理解进行场景纠错，可以协助用户更好的找到目的地，改善用户体验。另一个场景纠错的应用场景是语音产品，语音的内容应该与当前环境场景相吻合，基于文本理解进行纠错。



▪ 纠错开放平台

百度中文纠错基于百度十几年在自然语言处理领域的技术积累，并有效融合了丰富的各类知识库、文本理解等特征，通过互联网用户行为挖掘海量训练样本，结合树模型和神经网络模型的优势，保证了对海量数据的高效利用，因此，百度中文纠错具有算法识别精度高、效果稳定性强的特点。

以上技术特色已通过百度 AI 中文纠错平台对外开放，支持短文本、长文本、语音识别结果等多种文本内容，在搜索引擎、人机对话、语音识别、内容审核等方面有广泛的应用，能显著提高这些场景下的语义准确性和用户体验，欢迎了解和试用。

DataFunTalk

纠错开放平台

多类型 - 覆盖多种类型错误
多模态 - 支持文本，语音等不同输入形态纠错
场景迁移 - 快速灵活，可配置的深度定制

DataFunTalk

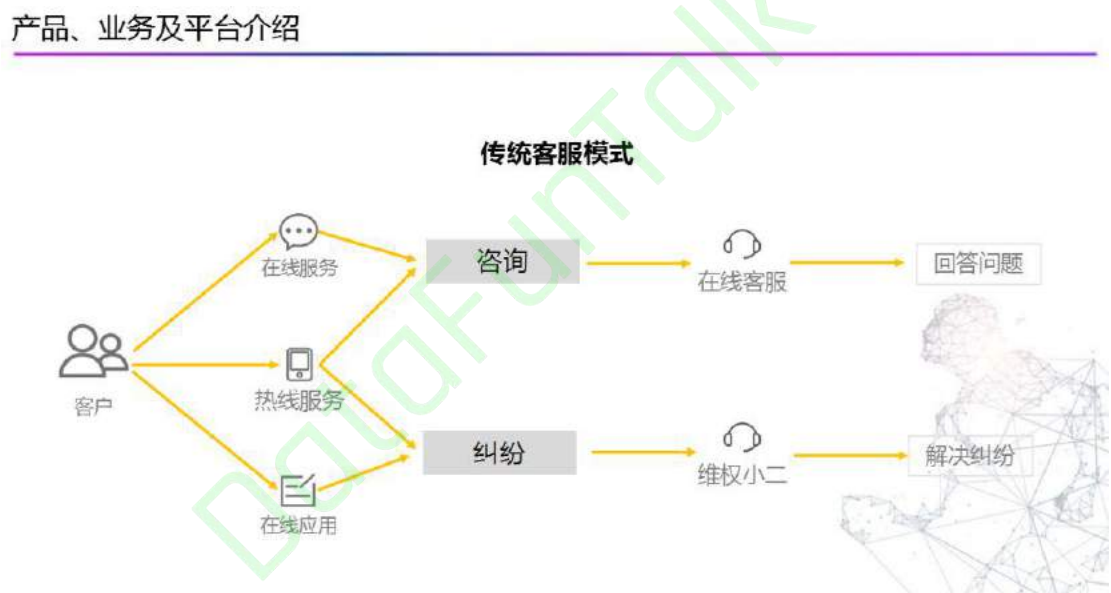
阿里小蜜：智能服务技术实践及场景探索

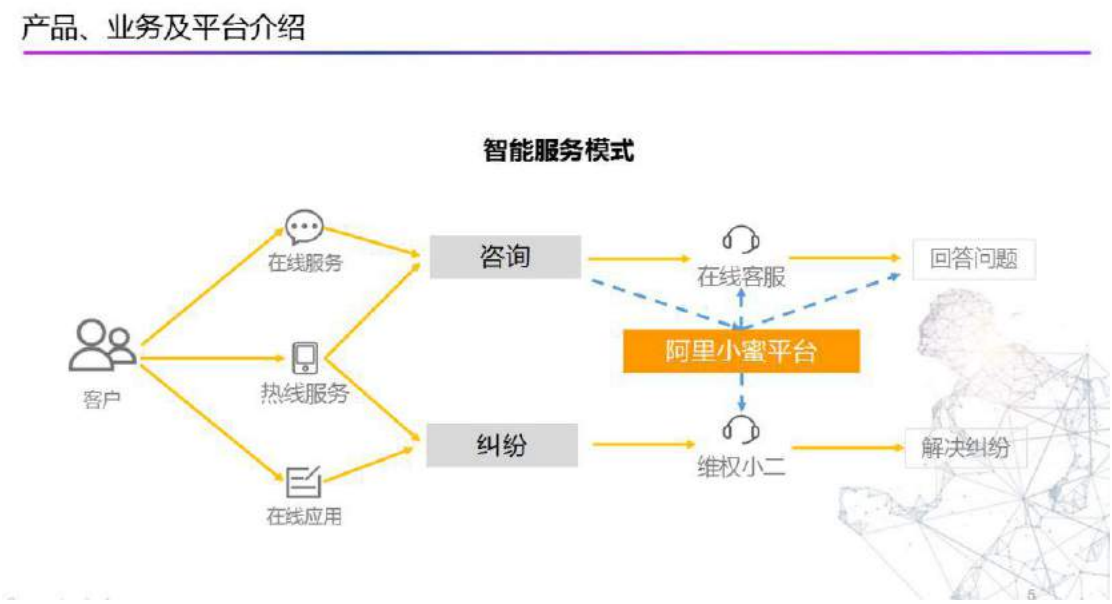
嘉宾：张佶　编辑：赵静宇

在 2017 年双十一期间，阿里小蜜所提供的智能服务机器人已经成为双十一服务主力。其中阿里小蜜的智能服务占比达到了 95%，智能解决率为 93.1%，为商家提供服务的店小蜜在 1 天内提供了 1 亿轮次的对话服务。在本次分享中我们将介绍阿里小蜜整体的产品业务、相关场景中的自然语言处理技术的运用以及阿里小蜜如何从智能客服逐渐升级为具备“泛服务”能力的咨询顾问。

阿里小蜜平台介绍

在人机交互行业生态领域发展的大背景下，面向智能服务领域，阿里小蜜在这两年时间同样在不断的升级与变迁，下面我们来对比传统客服与智能服务模式的区别：





阿里小蜜赋能各领域生态圈：

如图所示是阿里小蜜的产品功能，很多咨询业务都会通过小蜜来回答，作为一个人工智能助理，除了可以解决用户的业务咨询类问题，缓解人工压力之外，也兼备了其他助理业务，包括：查天气、买机票、充话费等。同时，还可以通过多轮会话完成一个导购助

理的工作，通过主动推荐引导的方式推荐商品标签，搜集用户意图，最终为用户推荐出满足他们需求的商品，其功能覆盖如下：

第二个是店小蜜的产品功能，店小蜜已经为海量的商家提供了智能服务能力，帮助商家更好的服务他们的用户。在店小蜜中，首先要解决的是每个商家都有的通用问题，如物流。其次，每个行业商品都有不同的特点，比如手机行业可能会有一些手机硬件和参数的问题，服饰行业可能会有他自己特有的问题。所以对不同的行业会制定一些相应的本体模型，然后针对自定义的 QA 模型做一些配置和输入。

第三个是是通过企业小蜜支持整个企业的生态圈，现在国内和海外的企业，以及第三方 ISV 和开发者可以通过阿里云和钉钉等平台开放服务。所以在企业方面，我们不仅有电商小蜜，在政务、金融、办公等领域也有小蜜的支持，目前我们有了中文、英文、越南语、印尼语、泰语、西班牙语、俄语、葡萄牙语等多种语言的支持。第一个例子是东南亚最大的电商平台 Lazada ，第二个例子是为阿里巴巴内部为企业员工服务的内外小蜜，目前也有大量的钉钉企业用户采用了这种服务模式。

DataFunTalk

阿里小蜜技术实践与场景探索

技术方面从以下四个方面来介绍小蜜：

- 意图识别
- 机器阅读理解
- 多轮增强式导购
- 迁移学习

先看下小蜜整体的对话流程，当用户输入一个问题的时候，我们不仅会对这个 query 解析，也会结合上下文对用户 query 进行理解。意图识别就是为了挖掘用户真正的想要做的操作。然后在对话管理系统中进行多轮式控制，在理解用户后，会选择三大类（QA Bot，Chat Bot，Task Bot）中的一类来回答用户的问题。这三大类有不同的技术去支撑：

- 第一类是QA Bot ,他使用IR建立自己的知识图谱 ,还有Machine Comprehension 从一篇文章中读取理解它等技术。
- 第二类是 Task Bot :主要是做任务型工作 ,比如订一张机票。这些使用 Slot Filling，Bot Framework 和深度强化学习等技术来支持。
- 第三类是 Chat Bot：纯粹的闲聊，比如“用户问你今年多大了？”等问题。使用 IR+S2S（检索式和生成式的方法）技术支持。

意图识别

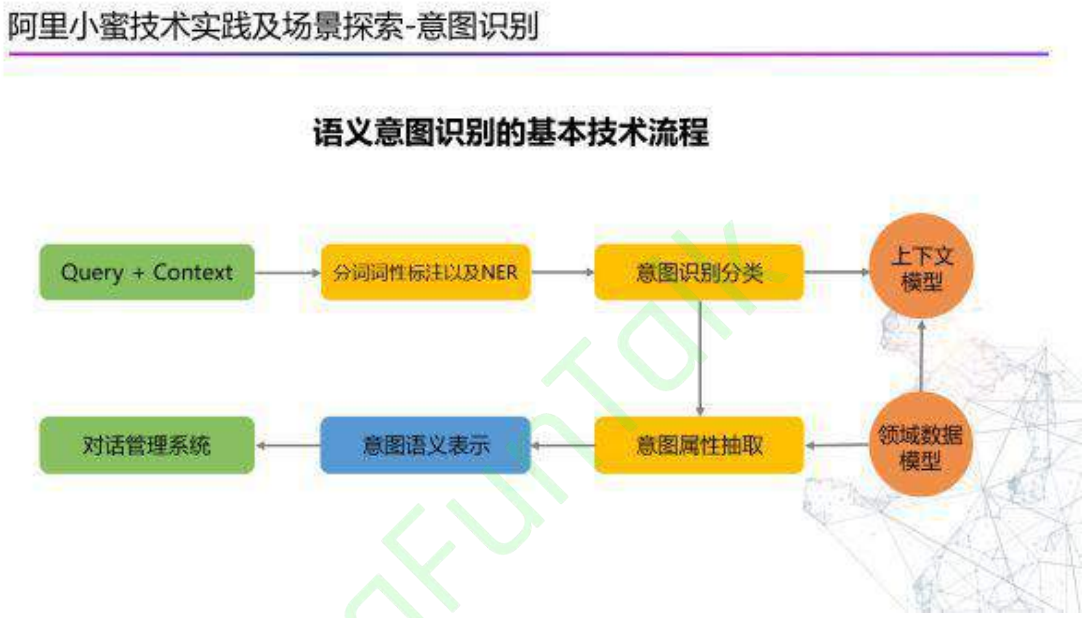
意图识别是机器人识别的第一个环节，阿里小蜜有三大场景。

- 意图场景识别：比如第一个用户说充值，其实是要给手机充话费。
- 多轮意图继承：比如用户第一句说上海天气怎么样，第二句说杭州呢？那第二句就是继承了第一句天气的意思。

- 动态障碍预测：当用户还没有输入问题的时候，阿里小蜜需要根据用户之前的询问历史来猜测推断出用户想要咨询的问题。

整体的技术流程

- **对用户的 query 首先进行 NLP 处理，分词词性标注以及 NER 处理。**
- **然后输入到意图分类的模型，也需要根据上下文抽取一些意图的属性，结合领域的模型判断出属于哪一个领域的问题。**
- 最后通过意图语义表示传回对话管理系统中。



分类

传统机器学习：

1. 多分类模型(有监督的分类算法 ,依据具体场景进行选型 Bayes\SVM\最大熵\...) : 适用于相对简单场景且分类数稳定领域；

2. 二分类模型（按照意图领域做成多个二分类模型）：适用于领域分类相对独立，并且经常需要新增修改的场景，能做到相互独立。

深度学习：

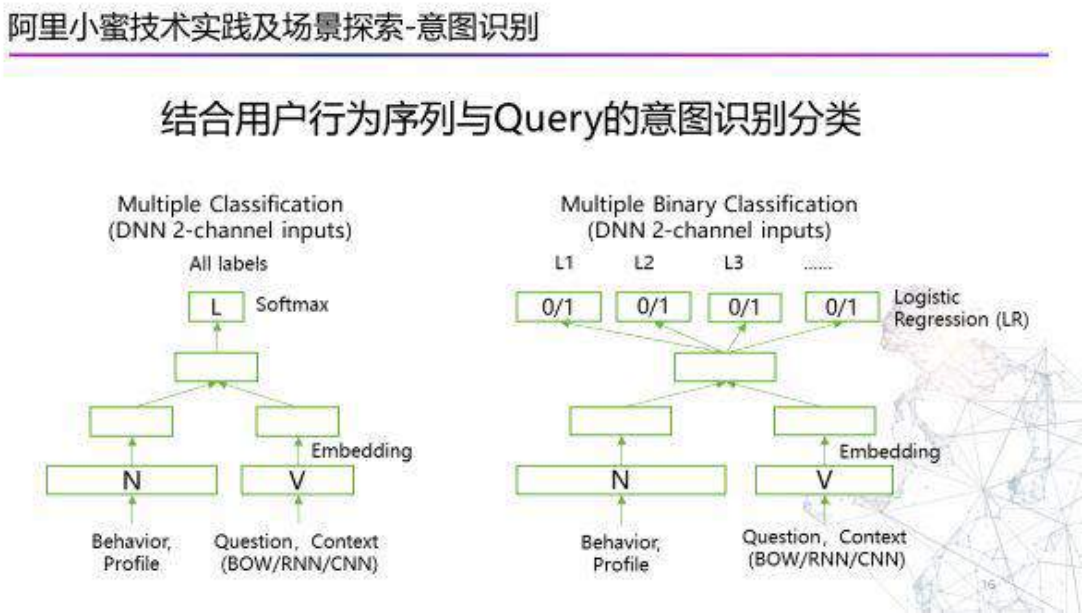
3. 深度学习多分类模型（CNN\DNN\LSTM\...）：适用于较大数据量积累场景；

4. 结合用户行为特征的深度学习意图预测模型：由于文本缺失、不明确或者不完整的情况下，增加用户行为特征进行意图分类预测。

深度学习分类模型结构

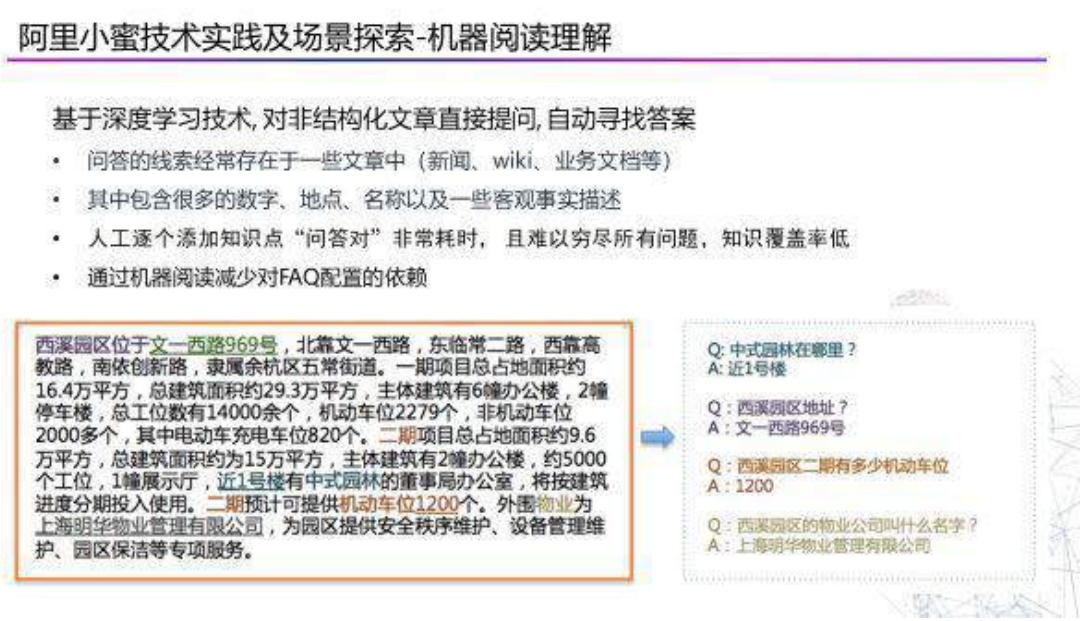
神经网络中会有两大输入，左边 N 会输入结构化数据，比如个人属性以及浏览操作历史纪录，右边 V 会输入一些非结构化数据，比如前几轮问的问题和序列，对于这些非结构化的数据我们会有句子编码器解析这些数据，当需要考虑到句子的语序关系的时候会使用 CNN 或者 RNN 网络结构；上层的话，会结合用户的 Embedding 和句子的 Embedding 去输出。但是工业真正的场景中，用户问题的问题个数是不固定的，所以会把最后一层 Softmax 更改为多个二分类模型。

DataFunTalk



机器阅读理解

机器阅读理解是目前一种新的 QA 方式，我们发现问答的答案线索往往已经存在于一些文章当中，如果能从现有的文章中简洁有效的抽取出有用的问题并且回答用户的问题，那效率会大大提高，省去了过去知识运营人员大量配置 FAQ 的工作。下面文章是阿里巴巴 wiki 的描述，这里面会有很多数字，地点，名称。如果传统的知识抽取 FAQ 的方法，这一小段能抽取二三十个问题，代价很大，而机器阅读目前能直接阅读回答这类问题。



机器阅读理解领域已经成为 NLP 研究的热点领域：

除了在学术领域取得一些成果之外，对于阿里小蜜来说更重要的是怎样通过机器阅读来真正解决真实场景中的问题。例如，在双十一等大促期间我们往往有大量的电商活动上线，而这些活动已经有一些规则文档，我们可以对这些文档进行阅读并回答用户关于双十一的活动问题，大大提升活动问答的上线效率。

第二个场景是企业小蜜在支持外部的客户税务法规的解读。过去需要税务专业毕业的专业人员才能回答，现在我们可以通过机器阅读税务法规文档来回答问题。

第三个场景用在海外的企业，Lazada-政策条款解读，需要解决的是如何让机器阅读运用到更多的语言，例如像印尼语这样的小语种，但印尼语在 Lazada 中是市场非常大的。

第四个场景是店小蜜中，我们推出了商家的活动专区功能，可以让商家直接以文本的方式来描述活动规则，机器自动阅读并回答用户问题。

机器阅读理解技术细节

一共分为四个步骤：

- 文章片段定位：针对用户问题，召回候选文档段落集合，借助文本分类、检索或者问题模板辅助；
- 输入预处理：格式归一，特征预计算问题，及相应段落向量表征，生成文档结构标签；
- 在线预测服务：GPU-Based 模型加载及服务驱动，预测段落中词或符号得分；
- 后处理机制：基于动态规划选取最佳文本短语作为输出，拒识：判断是否拒绝回答。

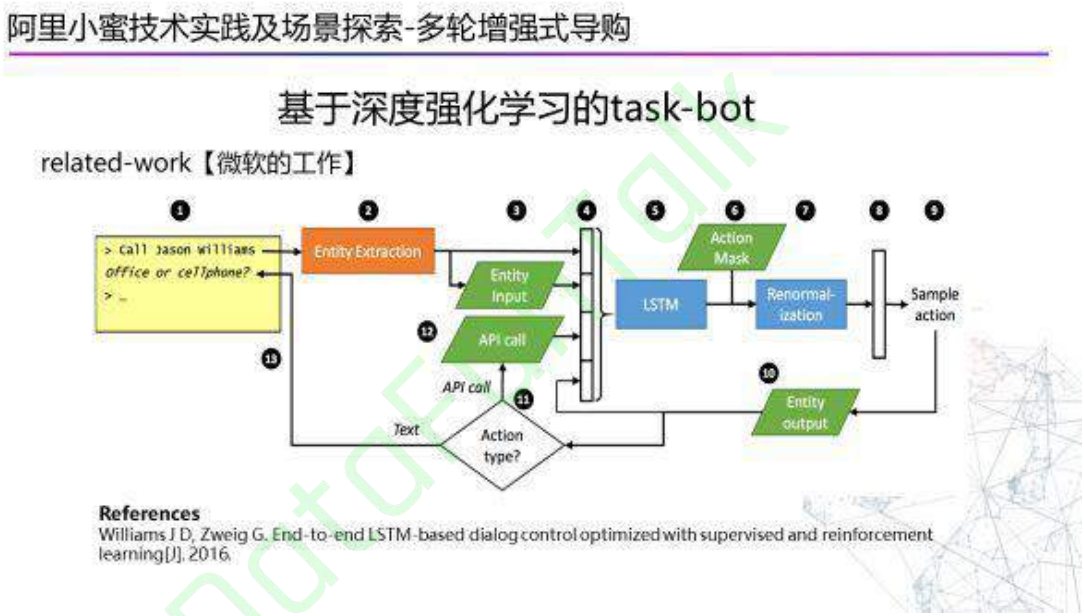
机器阅读理解模型结构

- **Embedding Layer 问题及篇章中词向量表示，RNN 网络捕捉语序间依赖；**
- **Attention Layer 对齐问题和篇章，语义相似性计算，引进注意力机制，带着问题找答案；**
- **Modeling Layer Question-Aware 篇章建模，充分利用问题中信息；**
- **Output Layer 基于问题和篇章匹配预测答案位置。**

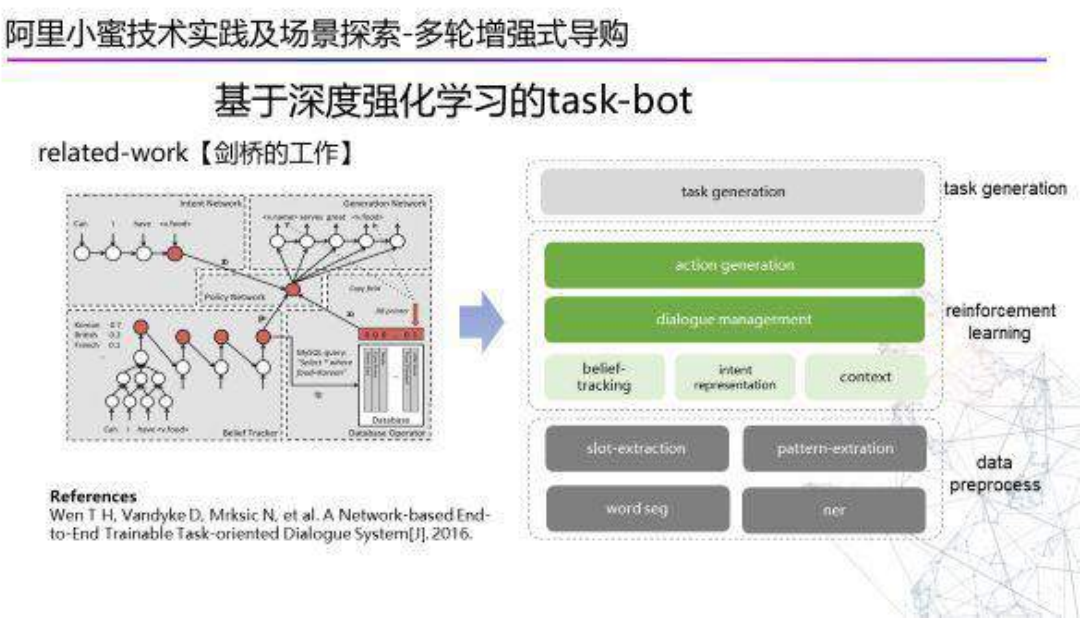
多轮增强式导购

对于电商类机器人，主要功能是能否帮助客户快速的定位到自己想购买的产品，下面是几个例子，这里面会有一个问题， 在每一轮对话过程中，什么时候该出推荐商品，什么时候需要继续反问？因为反问的太多会导致用户离开。这个问题使用强化学习来解决的。

下面介绍一些微软在这方面的工作，他会先输入一些少量数据，做一个有监督的学习，再用 Bot 和用户做一个真实的交互，通过的方式做筛选，在这个基础上做一些调优。

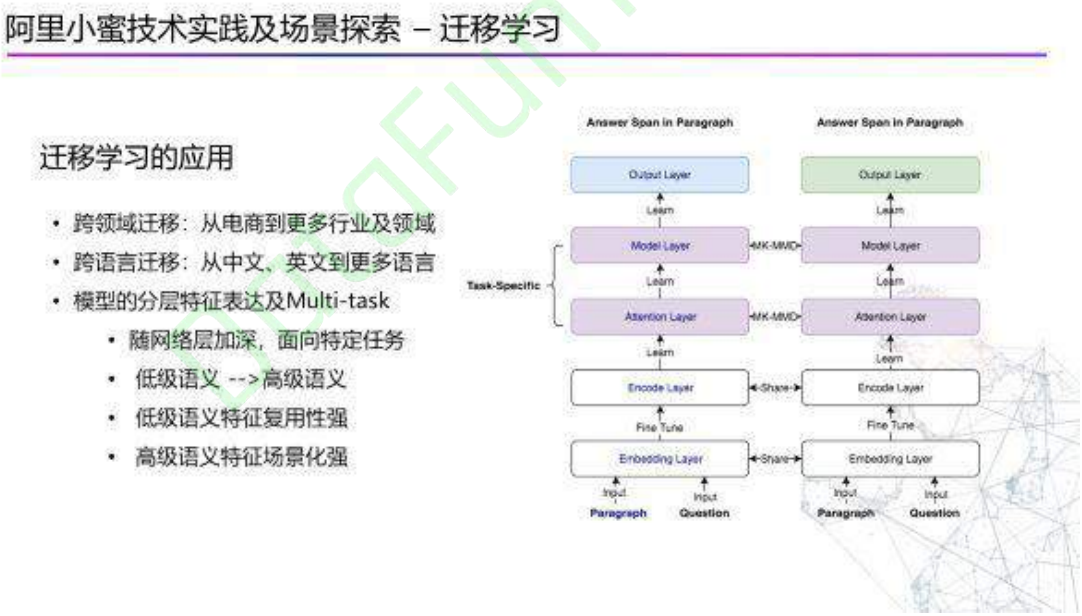


第二个是剑桥在这方面的工作，它的 network 来生成一个策略，他的结果在推荐问题和给出结果之间选择。



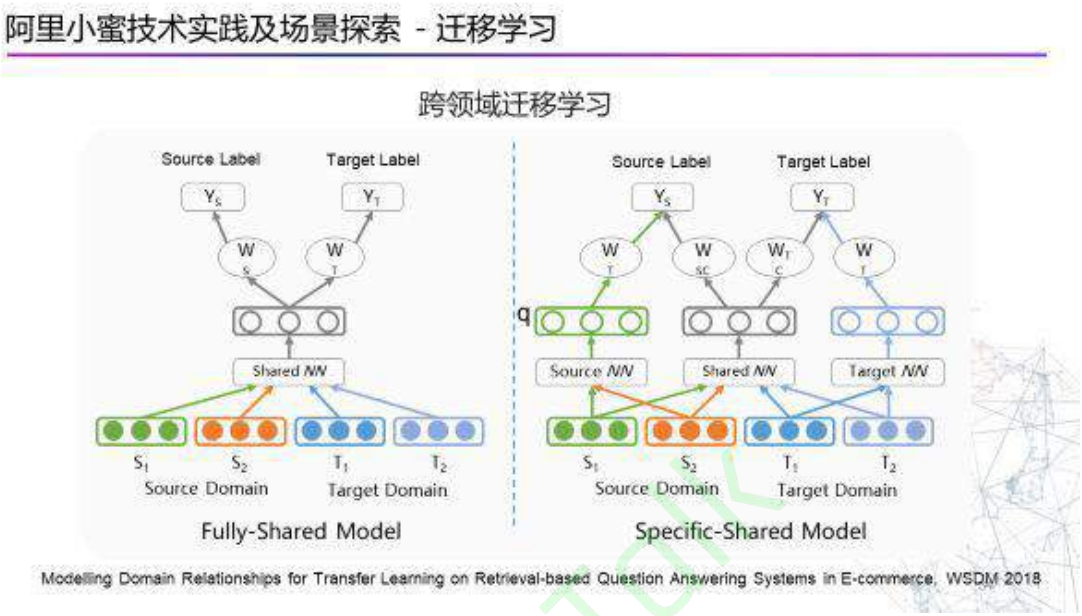
迁移学习

迁移学习对于企业小蜜有非常大的意义，因为需要从电商领域走向更多的行业，还有需要做跨语言的迁移学习。模型的迁移会做一个分层的表达，大部分的 NLP 模型都可以这样拆分，我们认为越是底层的层次越偏向于语言，他的复用性越高，越是上层，越偏向于具体的任务，复用性底。



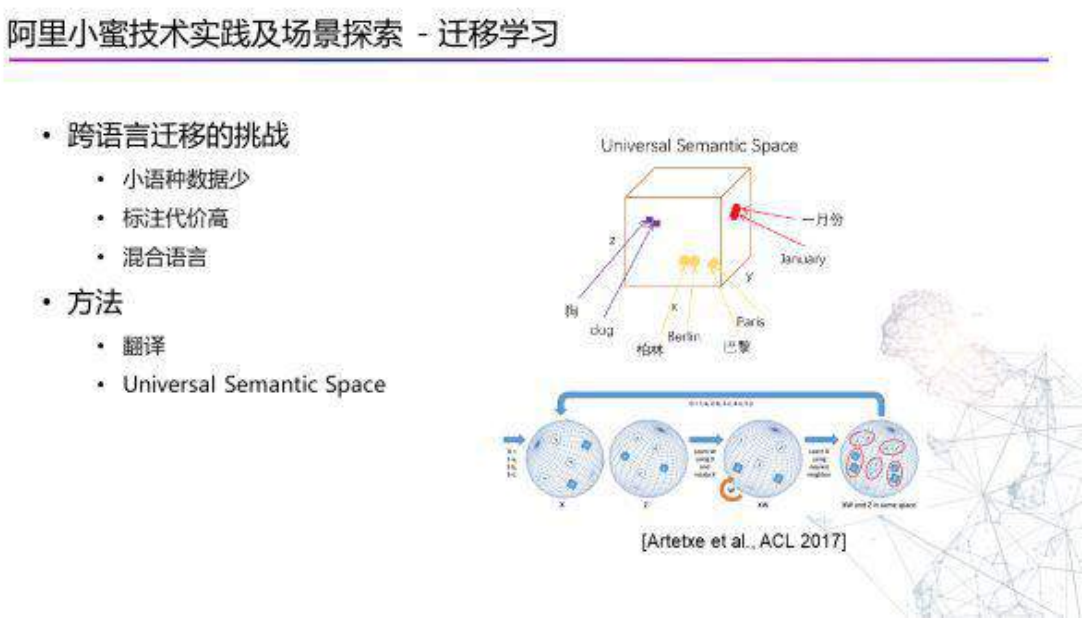
跨领域迁移学习

有两种方法，首先 Fully-Shared Model 是比较相似的两个领域，输入两个领域的知识，然后学习到两个领域公共的部分和表达。然后是 Specific-Shared Model 这种模型是采用摸着石头过河的方式，虽然两个领域相差比较远，但是也可以很好的过度过去，中间 WSC 和 WTC 会采用对抗的学习方式来训练。



跨语言迁移学习

这部分最大的挑战是小语种的资料比较少，标签化的数据更少，所以面临比较大的挑战。我们现在有两种方法去解决：一是翻译，二是对用户的每次 query 进行 Universal Semantic Space，右边的图是一个基础的思路。



挑战与思考

- **智能服务机器人将以多模态的方式，不仅提供自动服务模式，同时提供更好的人机协同模式，为用户和客服人员提供更复杂问题的解决能力和决策支持能力；**
- **持续探索的技术方向：生成模型、强化学习、迁移学习、机器阅读理解、情感分析等。**

DataFunTalk

阿里小蜜中的情绪回复能力

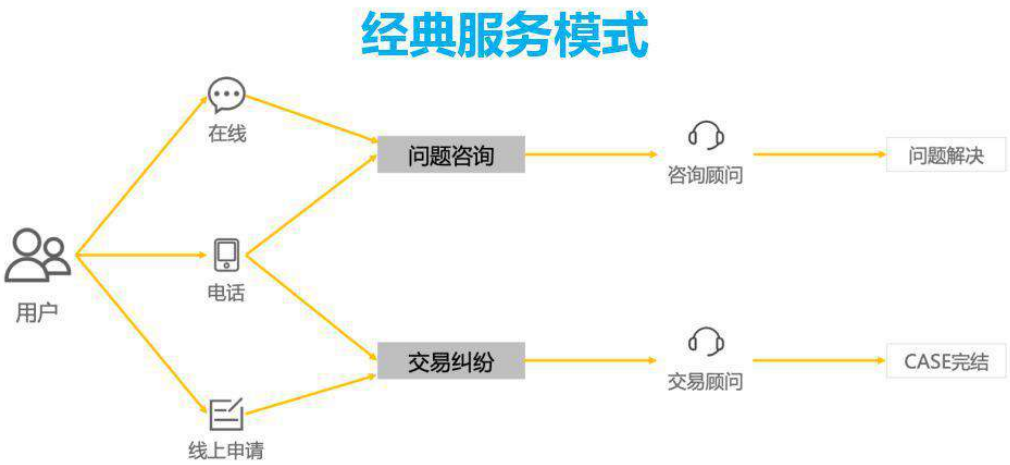
嘉宾：宋双永　编辑：赵文娇

本次分享内容提纲：

- 阿里小蜜介绍
- 情绪回复能力介绍
- 客户情绪安抚
- 客服质量检测
- 情绪生成式语聊

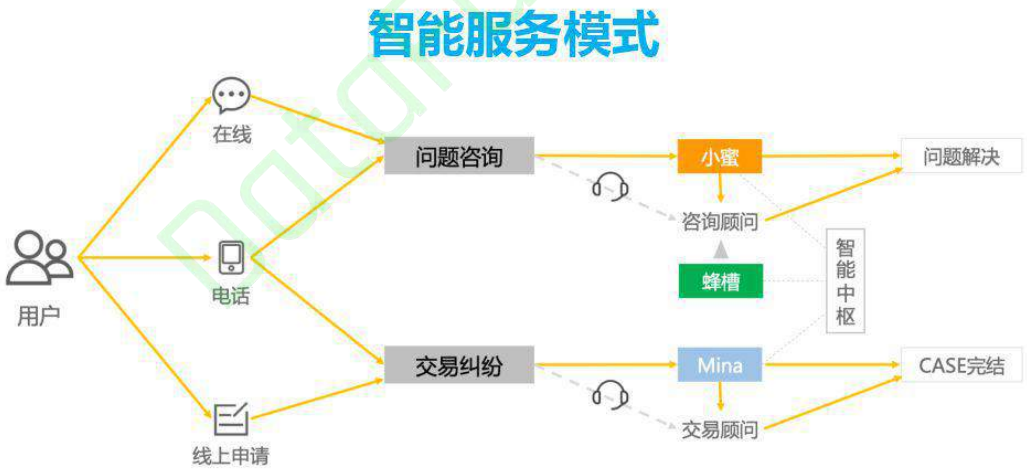
一、阿里小蜜介绍

传统的客服模式始于电话客服，会有专业的咨询顾问帮客户解答问题。之后有了在线客服和线上申请，在线客服类似于文字聊天，线上申请是非实时的通讯方式，可以理解为类似留言或者邮件的形式。



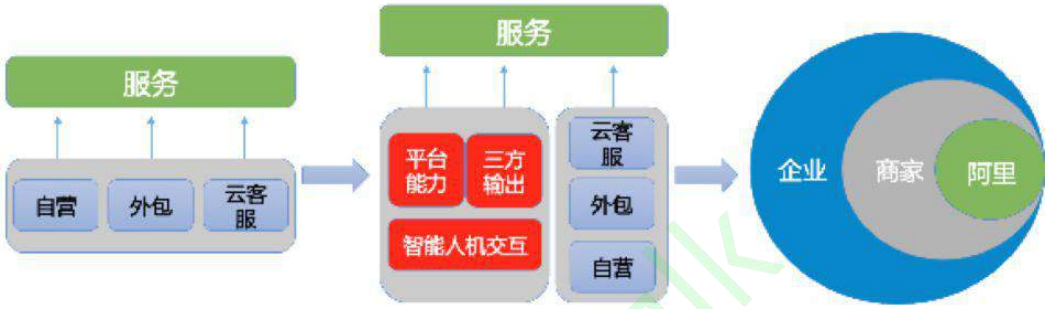
1. 智能服务模式

在阿里场景下我们对问题进行了区分，将其分成问题咨询和交易纠纷，分别有专门的顾问进行解答。在智能服务模式下，在问题咨询端建立了小蜜这样的产品，如果问题比较复杂，小蜜解决不了，还是会把问题转到人工顾问这一端，为用户提供更全面的服务。即便这样，在人工客服一端，也有对应的智能辅助产品，帮助提高人工客服的服务效率，比如检索是否有历史相似答案提供给人工客服，帮助他们快速完成解答。

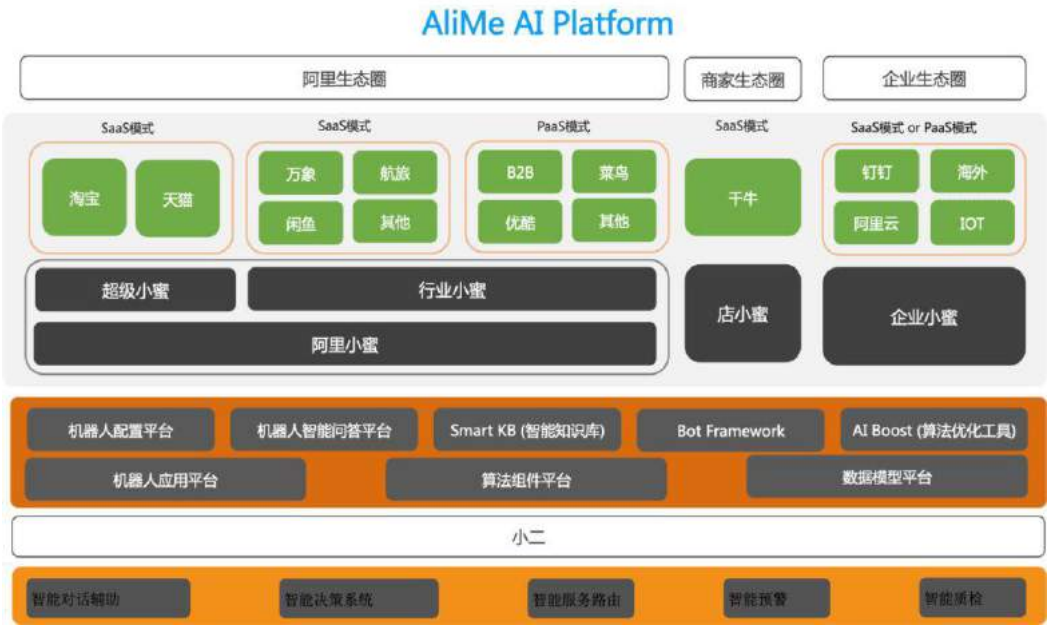


2. 模式的升级与生态圈拓展

智能客服能力建立之前，人工服务的能力来自于自营客服、外包客服和云客服。模式升级之后，利用智能服务的能力，形成了平台能力、三方能力、智能人机交互这三种服务模式。服务的对象可以理解为三个层次，其中，阿里是包括指淘宝、天猫、咸鱼、淘票票等阿里巴巴内部产品平台的范围；商家是指依托于阿里巴巴淘宝和天猫等平台之上的外部商家；企业则是指纯粹外部的企业。



如图，最下面这层其实是为小二进行服务的，小二是阿里客服的简称。在上面机器人配置平台这一层是为机器人服务提供一些能力。再往上一层是按照产品进行划分，分成阿里小蜜、店小蜜和企业小蜜。阿里小蜜主要服务阿里内部的淘宝天猫这样平台，店小蜜服务阿里平台上的商家，企业小蜜服务于外部企业。



二、阿里小蜜情绪回复能力

我们今天介绍阿里的小蜜产品在情绪回复能力上的技术发展。情感机器人的两个主要发展方向：

- 类人：就是情感越像人越好。
- 多模态：就是情绪的表达其实可以有多少手段，对人而言，可以是语言、表情神态、肢体语言等。

从情感处理能力上来说，可以把机器人分成三类：

第一类是机器人没有情感处理能力，阿里小蜜最开始的版本确实是没有情感处理能力的，只是对高频场景中的问题进行解答，比如用户说我要退货、如何退货、赶紧给我退货，得到的答复都是阿里平台上如何退货的文字描述，

但其实第三句是有强烈情绪表达的，但是初期的阿里小蜜没有这样的理解能力；

第二类机器人是有完整的情绪识别能力的，目前体现最多的是在一些闲聊场景下，比如小冰，比如在对骂场景下，如果客户骂机器人，机器人虽然没有直接的对骂，机器人也会有婉转的方式，比如 " 180 度反弹 " ，虽然没有直接对骂，但是也表达了对自己辱骂的情绪，这是一种情绪比较完整的机器人；

第三类机器人产品，客服机器人，客户可以骂人，但是人工客服和客服机器人是绝对不能有这种情绪的，辱骂、讽刺、挖苦是机器人绝对不能有的情绪。但是有一些，比如高兴、委屈是机器人可以有的，所以小蜜的定位是部分情绪能力缺失的客服机器人。

从多模态角度来说，阿里小蜜目前只考虑了文本，和少量语音。

情绪回复，今天会讲两个方面，一个是理解情绪，另外一个是表达情绪。

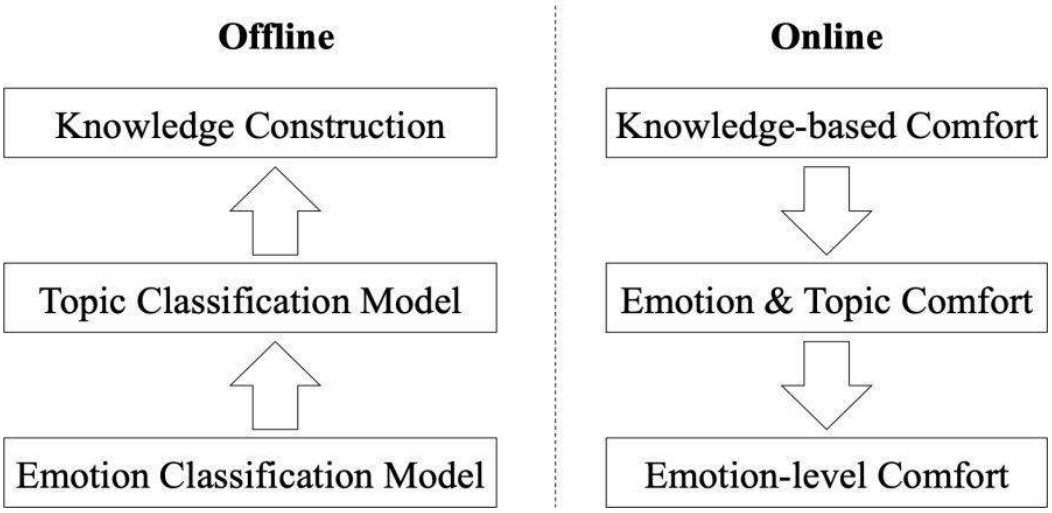
三、客户情绪安抚

DataFunTalk

从三个方面进行介绍：服务质量检测，客户情绪安抚和情绪回复生成。

先看客户情绪安抚，针对常见场景，我们不仅仅只是告诉客户一些流程，比如退货流程，还是有一些安抚在里面。然后看右面的情绪回复能力，这个情绪回复能力，和情绪安抚的主要区别是，情绪安抚是业务专家之前预设好的各种答复，但是情绪回复采用纯生成式的模型，用在闲聊场景，内容不是提前配置好的；最左边的服务质量监测，如果从小蜜转到人工客服，小蜜也是继续服务的，针对服务质量不好的时候，会对人工客服做一些警示，就是提示他，注意自己的服务态度。下面针对这三块进行更为详细的介绍。

针对客户情绪安抚，我们分成离线端和在线端。

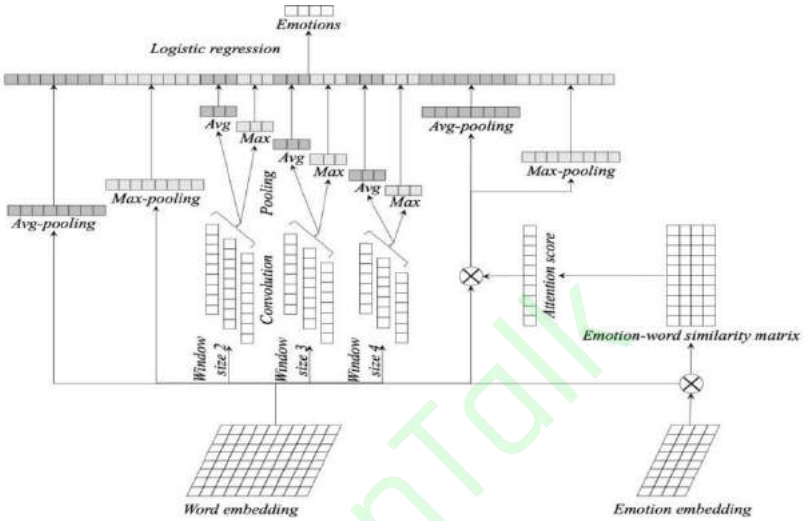


离线端，从下往上是如下几个离线处理，首先是情绪分类模型，这是整个流程最基本的东西，我们要去识别客户交流过程中体现的情绪；然后是主题分类模型，也就是说我们不仅要识别出情绪，还要知道聊的是哪方面的内容；第三步是知识构建，这里的知识构建应该就是一个问答对，做这一步的原因是，因为情绪分类和主题分类都比较粗，针对一些高频的问题，希望给用户更具体的回复方式。

在线端，其实是一个相反的过程，首先识别用户所说的是不是和之前总结的知识点比较接近，如果有就拿出来进行回答，如果没有就看是否可以归结到某种主题这样的情况上，如果再没有的话，我们就监测，它是不是仅仅是属于一种情绪表达，然后给更出更宽泛的情绪安抚。

其中的情绪分类模型：

通过数据分析以及参与常用的情绪字典，我们将情绪划分为 38 类（ 感激、惊奇、失望、抱歉、期待、疑惑、尴尬、高兴、着急、怨恨、喜欢、抑郁、委屈、轻视、害怕、孤独、愤怒、悲伤、满意、无聊、同情、平静、烦恼、激动、嫌弃、懊悔、羞愧、解恨、犹豫、思念、感动、敬佩、心慌、低落、骄傲、心虚、羡慕、辱骂 ），但是其实我们针对最常用的 7 类（ 委屈、恐惧、着急、失望、愤怒、辱骂、感谢 ）模型训练出单独的分类模型，这样对这 7 中情绪分类的更准确。

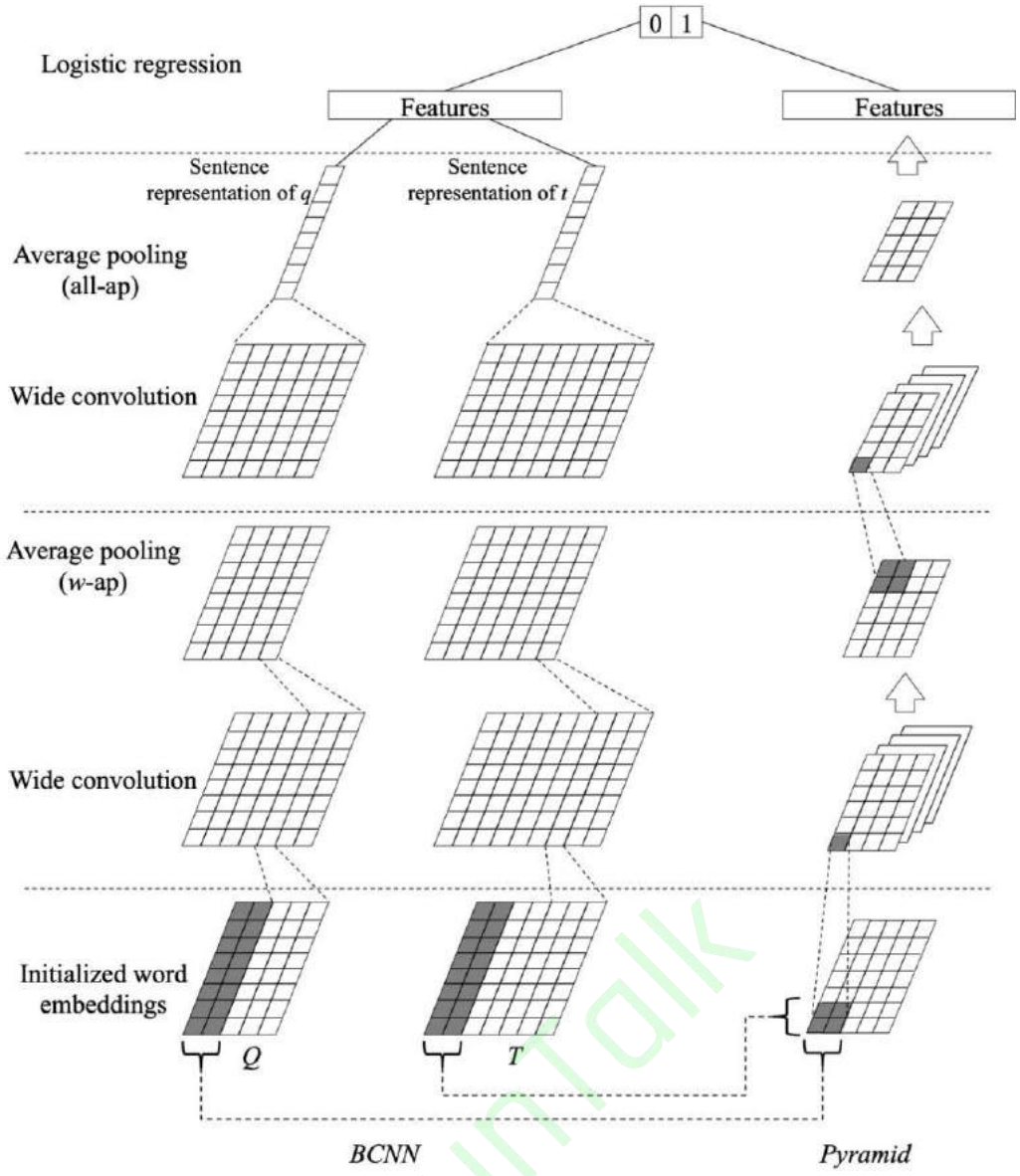


上面是模型图，最左边的两个 poolling 是在做句子级别的语义特征抽取，这个实际上用的是 swem 算法；中间是 n-gram 多元的特征抽取，用的是 cnn 的模型，我们提取了两元、三元、四元这样的信息形成特征；最右侧的一块是 emotion embedding，用到了 18 年发表一篇文章的思路。在我们的场景下，label 就是 emotion，所以这一块叫 emotion embedding，这里将 word embedding 和 emotion embedding 结合起来其实是算某个

词在某个 emotion 下面的 attention score，attention 可以理解为权重。这样更直接的体现了词级别的语义特征，整个句子从左到右，就是将词级别的、n-gram 级别的和词语级别的语义信息结合起来，才能获得比较好的语义分类。因为在线端用户打的句子都比较短，用这种方式才能实现比较好的语义识别的结果。

第二部分就是主题分类，我们定义的时候称呼为 " 情绪主题类别 " 。比如阿里小蜜有查天气的功能，并且被高频使用，这部分有没有必要放到这个模块呢？其实是没有必要的，一般是通过点击按钮引导操作，只是想看看天气，几乎没有情绪表达的。而这里提到的模型是用于识别情绪主题类别，从情绪的角度，归结常见的主题，而其它不带情绪的主题是没有归到里面。下面是针对 7 类情绪，做了 35 种情绪主题分类，主题分类架构和刚才的情绪分类的架构一致。

DataFunTalk



最后，基于知识的安抚，就是看用户说的话是不是和某个知识点很相近，这时候我们就用到匹配回复这样的模型能力，在文本匹配这一块儿可以分成两个功能能力。首先是分成两部分，最左边两块是一部分，最右侧是一部分，最左边的两块是两句话，他们分别在提取特征，最后把特征合并到一起做分类，最右边的部分是把两个句子从一开始就进行交互，把交互的结果，一层层做特征抽取，这相当于一个交互时间点的不同，一个是最后交互，另一个

是一开始交互，我们把两种交互抽取特征的结果结合到一起，做一个准确率更高的文本匹配模型，来实现在线匹配问答。

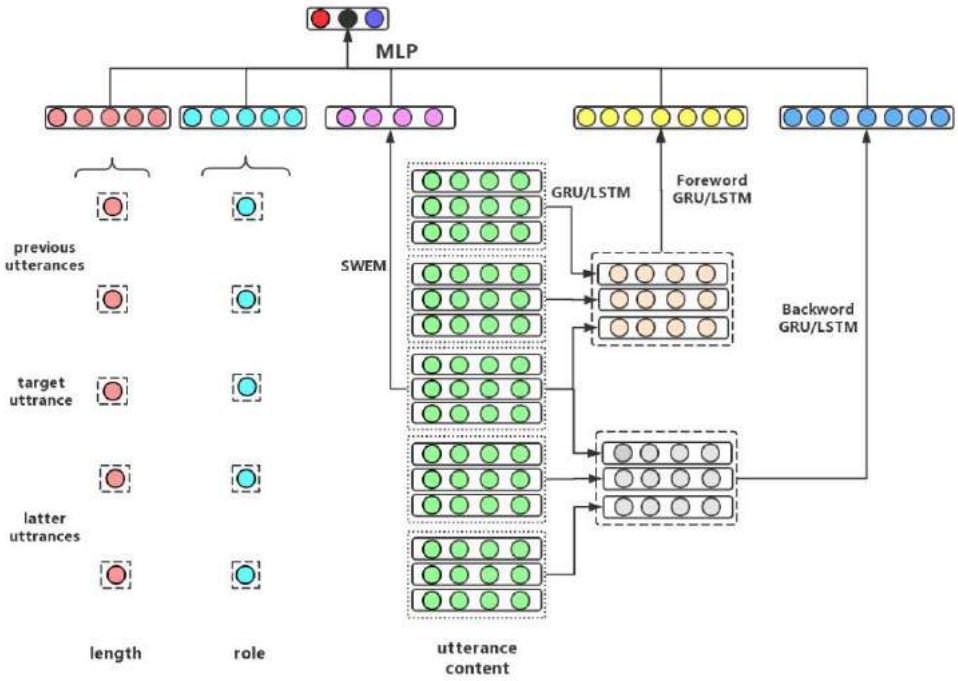
四、客服质量检测

我们只探讨两种服务问题，一种是消极，一种是态度差，消极是指爱搭不理的态度，态度差是指客服虽然给了客服充分的回复，但是态度不好，比如可能有反怼客户，讽刺客户的现象发生。这套服务提供给平台端和商家端，平台端就是比如淘宝，天猫这样阿里自家平台，商家端是商家自己的店铺的客服检测，这是两个不同的模型，因为两种场景在服务质量的要求上存在不同的衡量标准。

在机器人端典型的对话方式是一问一答，而在人工客服端往往出现多问多答的情况，比如客户连着说几句，客服是连着回答几句。这时候我们对客服服务质量进行评价就需要非常关注上下文，并且上下文每句话是谁说的，等等这些信息。

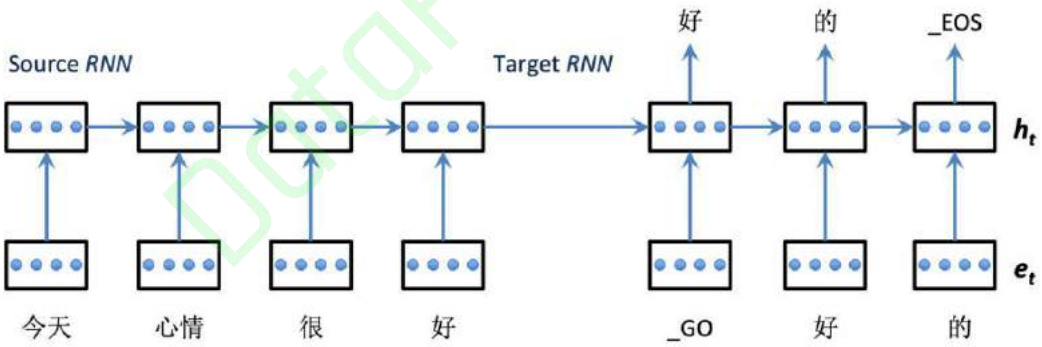
模型如下图，考虑了句子长度、说话人角色，以及内容的语义信息等等特征。

DataFunTalk

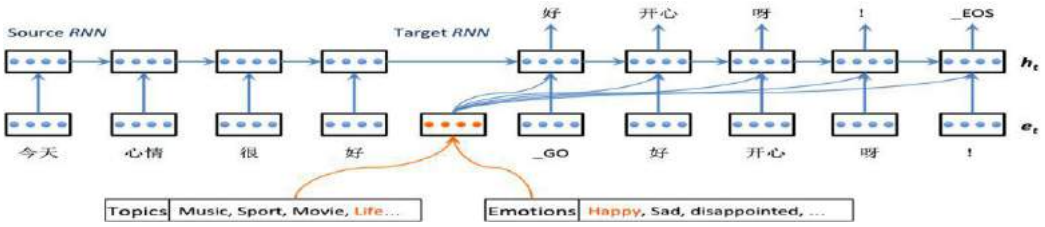


五、情绪生成式语聊

下图是比较通用的语义生成模型，这种传统的生成模型存在的一个问题就是 ' safe response ' 的产生，就是很泛泛的一个回复，很少有情感传达在里面，比如好，哦哦，可以这样的回复。



我们的目的是让机器人产生带情绪的回复，另外希望回复更具有针对性一些，而不是全部都是通用的 ' safe response '，在下面的模型里，除了情绪，我们还可以添加 topic 相关的信息，分析出聊天的主题。下面的例子里，客户说今天心情很好，聊的是生活化的主题，表达的是高兴的情绪，这时候我们生成 ' 好开心啊 '，回复用户。



针对上述的用户和机器人的情感对应关系，可以进行预设。比如客户在表达高兴的时候，我们也要表达出高兴，用户在辱骂我们的时候，我们要表达出委屈。

六、未来工作

未来，我们要做一个 session 满意度预估，这里的 session 就是一个完整的对话，目前是通过人工用研分析，是设计一个调查问卷的形式，随机抽取每天的用户，然后让用户打分，最后的分数就是满意用户的占比。

存在的问题就是：一个是耗费人工；二是每天的统计量是不足的，因此会产生每天统计结果比较大的自然震动。

通用领域对话问答

嘉宾：李亚楠　编辑：HohXil

通用领域对话问答通俗来说就是智能助手，今天主要分享下通用领域的应用技术和相关难点，还有就是我们在这方面所做的探索和实践。

首先讲一下对话领域发展过程，对话领域发展已经有半个世纪的历程，上个世纪 60 年代最早的对话机器人 ELIZA，通过规则和剧本的方法可以像心理咨询师一样和人对话，而后又出现了 Alice、Mitsuku 等基于更好剧本引擎的对话机器人。近年来，各大巨头公司都投入到了智能对话问答领域，出现了 Siri、Watson、Google Assistant、微软小冰、天猫精灵、小度等，这些都可以看做已有搜索引擎的下一代演化，这里我会从工业界搜索引擎的技术来讲解。

第一代搜索引擎的发生是人连上了互联网，互联网能给人带来很多信息。第一代互联网就是杨致远和他的同学仅靠人工编辑就能将好的网站收录进来，我们称为第一代互联网搜索引擎。随着互联网信息的增多，两个人的搜索是不够的，甚至 Yahoo 这样最大互联网公司的编辑力量都不足，后来就有了 Google 的网页搜索。

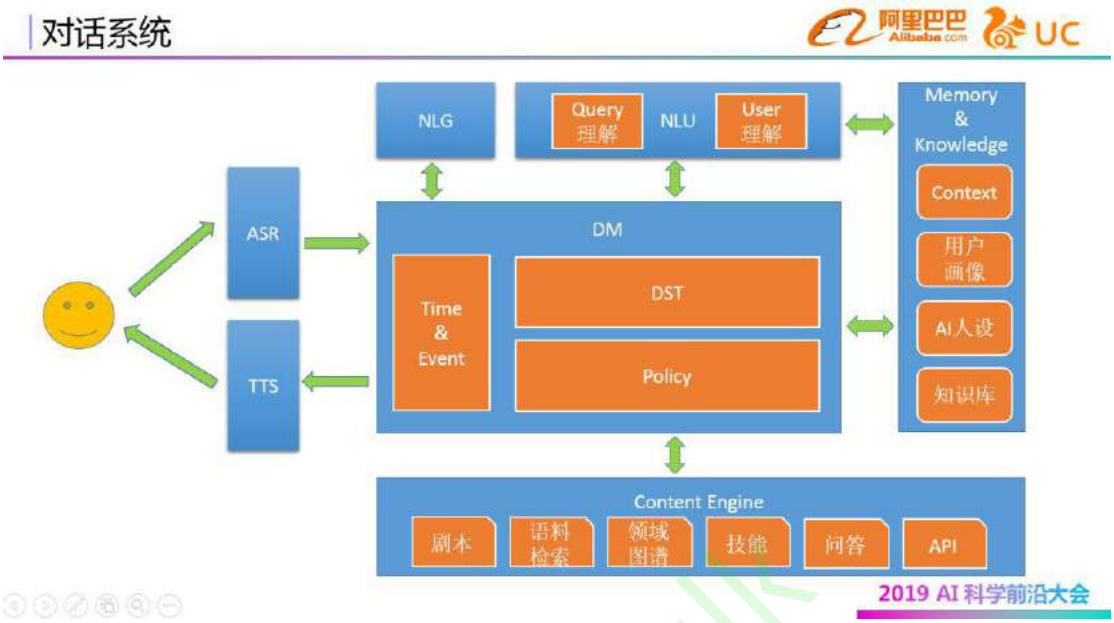
随着智能手机的发展，Web 也仅仅是互联网信息的一种，有了不同 APP 的私域信息，这对搜索提出了更大的挑战。目前又迎来物联网时代，不仅仅是手机、PC，还有更多的设备，这种情况下到底需要一种什么样的搜索。我们所面临的信息、知识、交互也不仅仅是 Web，变得更加多样性，人们更加需要的是一个跨平台的智能助手。这就是虽然还有很长的路要走，但是大家都很重视的原

我们部门属于阿里巴巴 UC 事业部下的神马搜索，其前身最早是中国的雅虎搜索，阿里收购中国雅虎后做了全网 PC 端的阿里云搜索，后来又做了移动端的神马搜索，接着推出了 UC 头条，目前随着人工智能的发展，推出了自有的夸克智能搜索和浏览器。

搜索的目标也是要做一个跨平台的智能助手，用到的技术有语音、搜索、推荐等，今天主要讲解 NLP 模块。如果我们要在技术上做好对话引擎其内容是非常重要的，就像大数据必须要有数据，没有数据很难做到智能。数据除了有传统数据、知识图谱、非结构化的数据、网页其他信息，我们还有用户画像，阿里的自媒体生态等共同支持我们的搜索引擎。

目前市场上对话系统架构大的层面都是差不多的，我们的对话平台分为三种对话类型，第一种是 TaskBot 就是任务型对话，如询问天气、订票服务，某些客服也可以看成是一种任务型对话；第二种就是 QABot，问答是人们见的最多的，主要是一问一答；第三种是 ChatBot，有的人称为闲聊，但是我们认为比闲聊范围广，常见的例子就是微软小冰，聊天是平常最多的信息交互，也是技术上难度最高的。下面支持它们的是各种各样的数据，包括知识图谱、用户画像、问答索引库、各种任务对话库、还有各种各样的剧本。在此之上还有一个统一的调度系统——对话系统，因为作为一个通用的对话，需

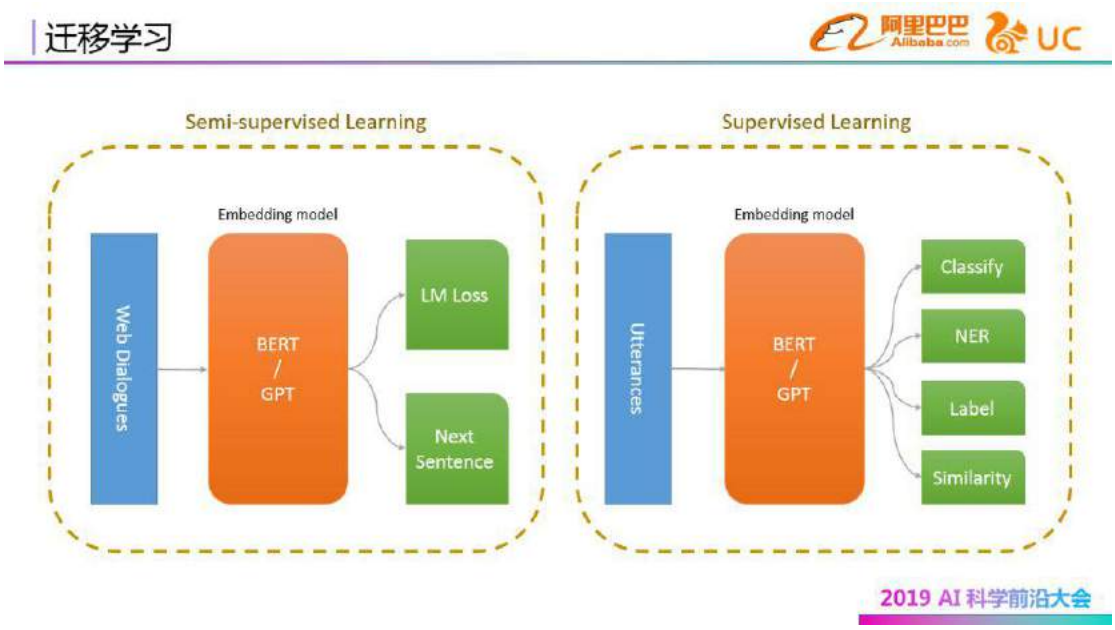
要能够在不同的任务、场景进行切换，而且需要像人一样记住不同领域的上下文。再往上面是一个对外提供服务的对话平台，在集团内也支持很多业务，尤其在通用领域的一些对话需求，如天猫精灵、自动驾驶助手等；还有就是我们自身的一些产品，如 UC 助手、头条评论，以及夸克宝宝。

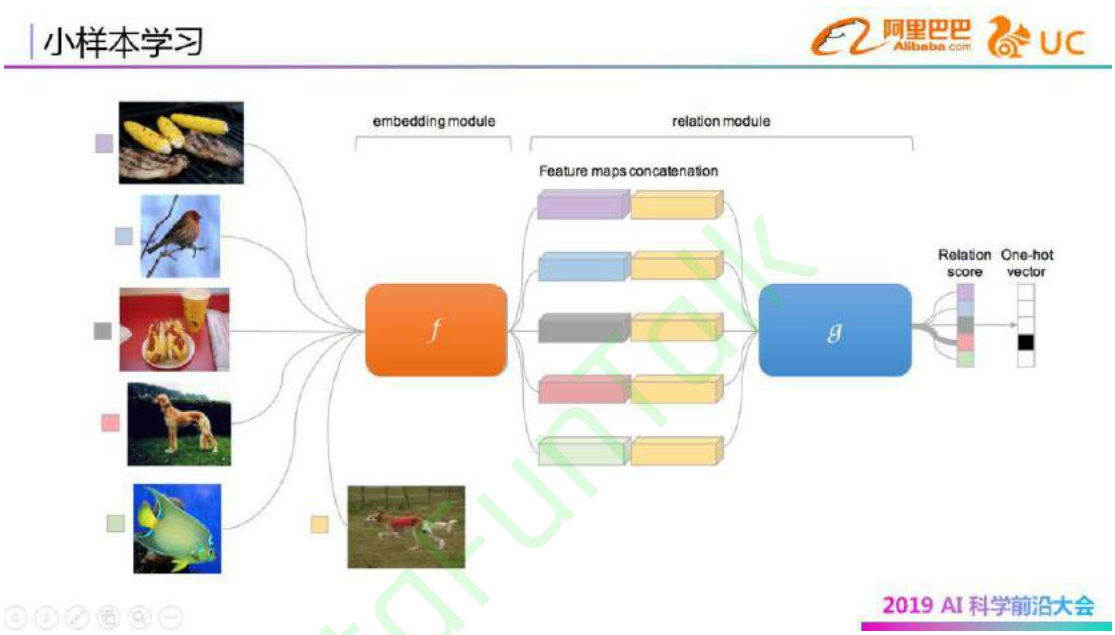


常见的对话系统除了语音的 ASR 和 TTS 以外，就是一个 NLU 和 NLG，中间一个 DM，但是这里多出来一些东西。从一个系统的角度或者更加拟人的角度来说明，对一个对话来说理解并不是内容的理解，还需要根据对话的人的对象的理解以及基于我们的记忆和知识去理解。因此就需要有一个 Memory 和 Knowledge 的部分，主要解决人的短时记忆 Context 和人的常识记忆，如去年告诉我名字不能今年就不知道他是谁。还有就是机器人自身的信息，也需要设置机器人的三观、喜好，在不同的领域和场景也是不一样的；另一个机器人要有自己的知识。另一部分就是各种各样的服务支撑，人是一个情感

机器，如计算机的众多外设一样，有很多小型的自处理器，如果要解决不同类型的任务也需要不同服务的支持。

任务型对话理解就是理解他的 dialog action、slot、intention、domain 字段就差不多了，但是对于一个通用的是远远不够的。理解大致分为三个部分，第一个就是基础 NLP，就是词法、句法、多轮指代等一些基本 NLP 知识；还有一块就是画像，需要人的一些信息；中间一个就是对对话上下文相关的一些理解，如领域、意图识别；还有 query 结合上下文的结构化，多轮指代消解是将其补全，补全后我们还需要理解和分析。除了语义内容的分解，还有一些状态的分类，识别是表达命令、观点或者愿望等等。图中每一类又包含上百类，也有一些是打标签的任务，那么如何解决这么多任务和类别是一个难题。

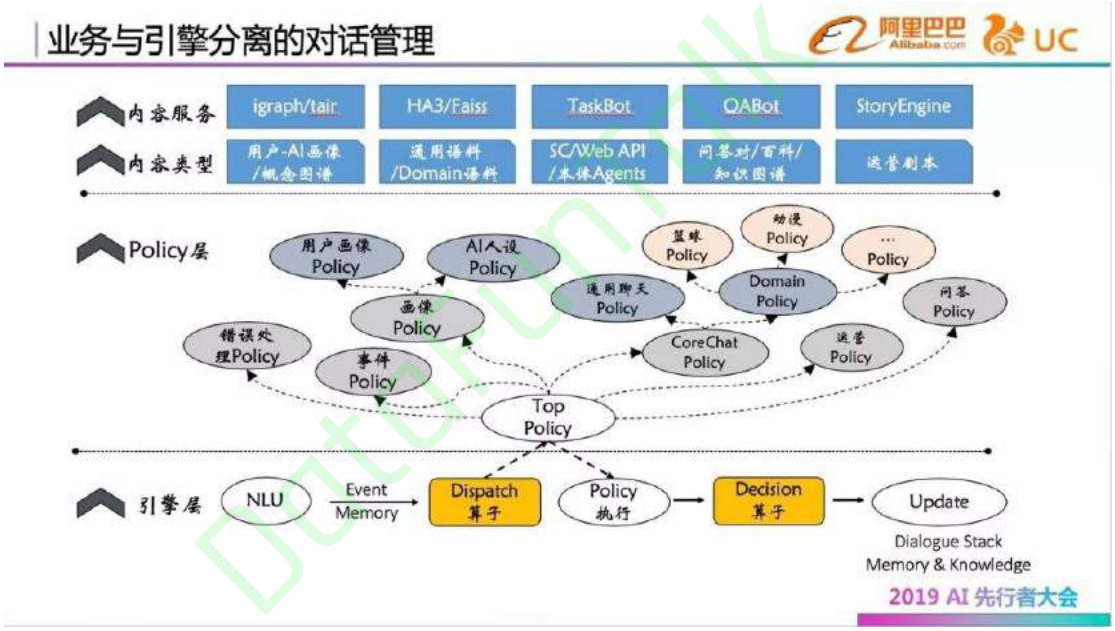




针对这一难题，会用到很多技术，因为需要达到 90%以上的准确率，才能应用于工业界，如果每一个都仔细做很难达到时间成本效率性价比。因此主要有两类技术，一类是迁移学习，一种是小样本学习。迁移学习能够在复杂的任务中快速做出一个好的效果，我们会从网上采集各种对话的语料，通过预训练做出一个 embedding 的 model，预

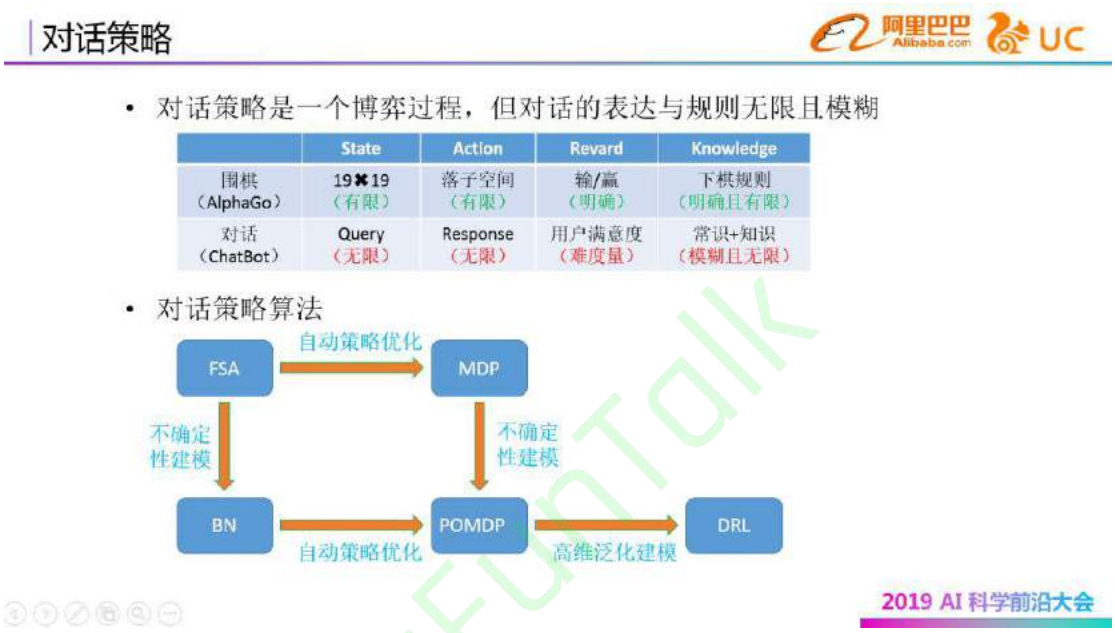
训练的 model 会依据任务或数据做一些改进，模型的选择取决于任务，并不一定 BERT 就是最优。

实际工作中有成千上万的类，很难获取充足的样本，而像深度学习或者传统的机器学习模型都需要大样本。小样本学习主要有三类：Model Based、Metric Based 和包含梯度和训练方法优化的 Optimization Based，我们主要是用 Metric Based 的方法，一个类别可能只有有限几个样本，将不同类别的样本与要学习的类别进行配对，构建度量配对后的相似度或者距离的 model，然后通过这个 model 跟已知的类的样本做对比，观察其更接近哪一个类别。实际处理中还有很多问题，如自然语言噪音的问题，有的比图像还要严重，还需要做一些事先检索和聚类，在做 embedding module 和 relation module 时还需要一些特定处理。



多维对话管理就如同人做决策，就是将非结构化的文字信息转化为结构化的数据和特征，对话管理需要依据这些信息表达什么东西，NLG 将这些要表达的概念转化为一个

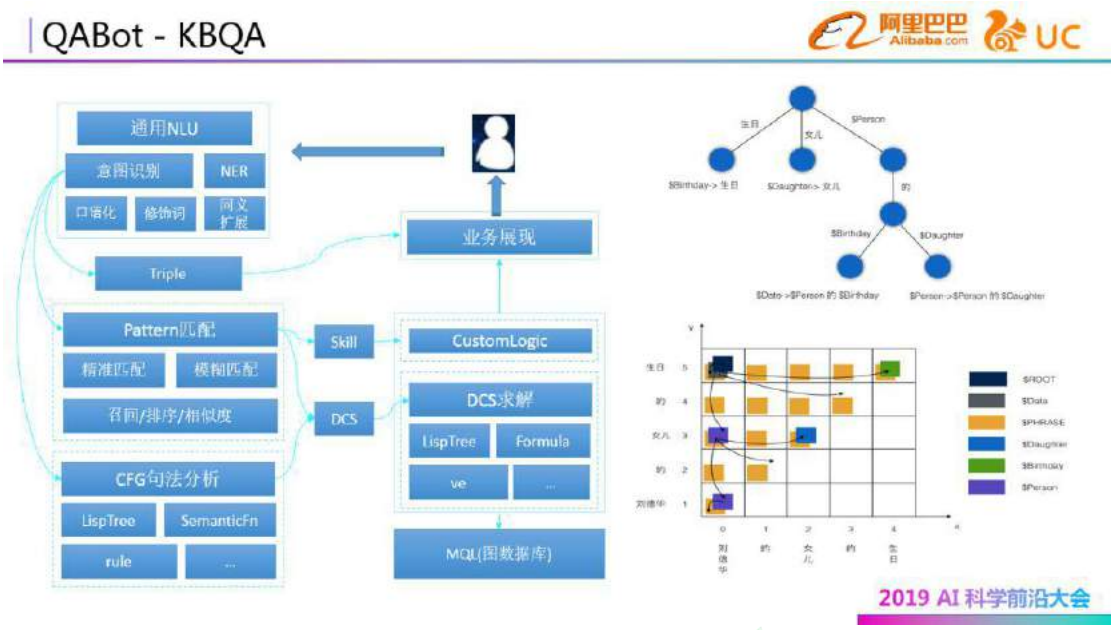
流畅的自然语言表达出来。但是作为通用领域难度比较大，因为不同领域业务范围不同，需要将不同领域的知识、架构统一起来，既不单调也能有个灵活处理，这是对话管理比较大的难点。还有就是如何做到良好的可扩展性，就需要做到业务和引擎的分离。对话管理的引擎是统一的，是引擎层，获取各类信息进行决策和更新状态。policy 层就如同机构的分支，易于各类业务的管理、扩展和复用，同时还需要不同类型的数据、不同架构进行支撑。



在有了不同领域的知识如何做明确的决策仍然是一 个难题，这种策略就对话管理而言是一个博弈的过程。对话是一个非常模糊的过程，有时让说话人自己标注都很难标注。这样就会产生很多问题，最早的解决方式就是有限状态自动机、slot filling 等，比较适合一些小的任务。这对于通用型对话远远不够，如何扩展有很多方法，一个方向是将

确定性转化为不确定性，类似于像贝叶斯网络发展；另一方面其策略变成一个不固定，可以依据前面的过程进行优化，如 MDP；POMDP 是综合了两种方法，上图是决策过程的优化。结合对话复杂度，需要有更高维度的建模，图中 DRL 是深度学习的增强学习过程。由于增强学习方法需要大量成本高昂的标注数据，目前应用领域比较局限。

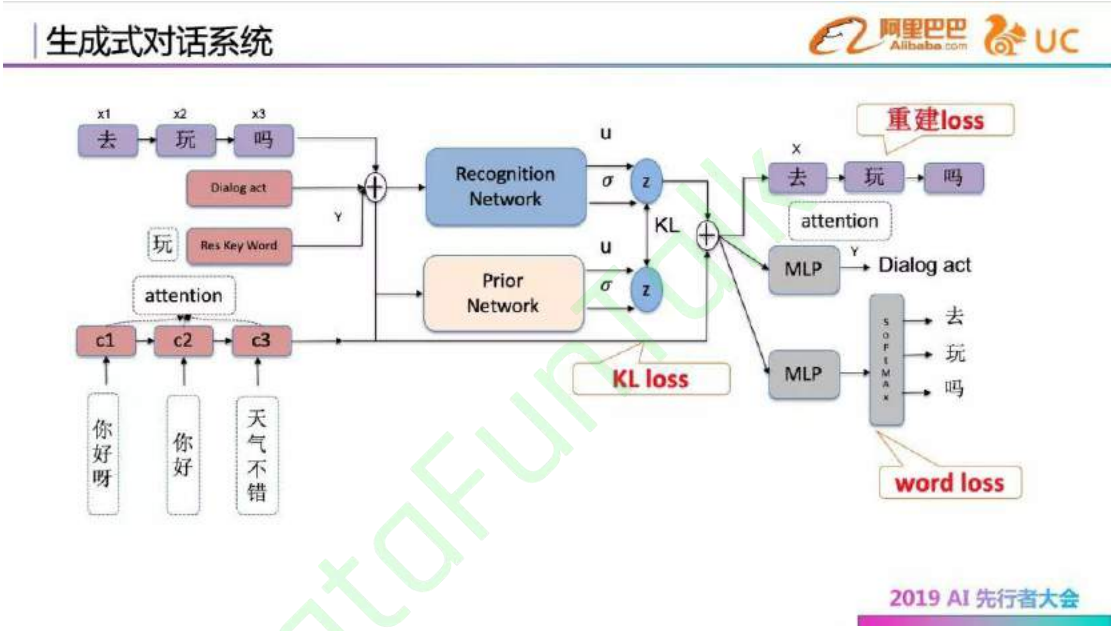
TaskBot 基于任务型的对话系统是一个小的通用对话系统，需要针对不同领域做一些优化，我们称为 Sub-NLU。类似地，在 DM 和 NLG 中也需要补充相应的 policy 和表达式。得益于阿里丰富的资源，夸克提供了涵盖各场景的智能信息服务。



问答机器人有很多种，最简单的是基于配规则的卡片式问答。这里主要讲两种问答，一种是基于知识图谱的 knowledge based question answering（KBQA），还有就是基于非结构化的网页知识来抽取答案的问答机器人称为 DeepQA。基于知识图谱的问答首先是构建全网的知识图谱，会通过 NLU 识别实体，然后再依据实体做消歧、同义扩展、口语化处理，随后将其做一个聚合分析，将其转化为逻辑表达式，再将其转化为 SparQL、MQL 等图数据库查询语言，从图数据库将数据取出来，再依据不同类型的知识做一个 NLP 生成，再将结果展现出去。图中比较关键的就是自然语言转化为逻辑表达式，这方面有两种方式。一种比较传统，就是模板匹配，这是为了在业界达到一定精度。除了精准匹配还会做很模糊的匹配，将一些 pattern 转化为 embedding 方式，做

一个更强泛化能力的匹配。还有一种方式就是通过句法解析树的方式来解析一些非结构化的 query，再将其表达成相应的逻辑表达式。

还有一种就是非结构化的 DeepQA，图中有个例子是唐代哪个诗人不畏权贵，网上的答案很长，但是我们希望回复只是李白或者精简一句话或者李白的介绍。实现方法就是主要抽取比较关键的句子或者摘要。这要求在传统搜索引擎的网页基础上要做很多工作，也包括句子排序，最终抽到的句子的组织。其中最核心的是交叉验证部分，整个互联网信息比较杂，有些信息就是错误的，有些信息还涉及违法信息。避免这类情形发生就要做各种粒度的交叉验证，众多答案交叉验证、与权威源交叉验证、句子与句子间交叉验证、问题类型与 answer type 的验证、关键词与关键词的相互验证等。技术实现核心就是数据，词级别的验证就需要知识图谱作为权威源，需要有问答对，还需要同义词、问题类型与关键词等，所以我们需要先挖掘高质量的数据，还需要挖掘一些训练用的样本，包括设置一些相关但是又离得远的负例，这都是真正复杂的工作。

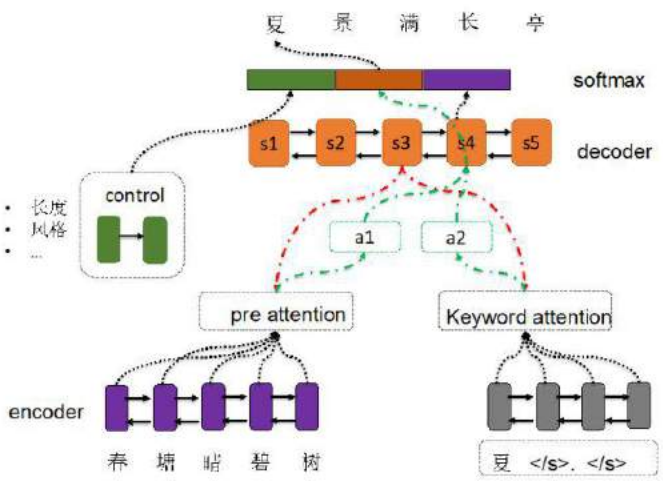


对话闲聊主要有两种解决方式，一类是检索式，一类是基于深度学习的生成式。传统的对话解决思路和问答是类似的，因此数据变得很重要。我们抓取了上百亿的不同对话组，也获得了数以亿计的高质量对话。排序和问答类似，但是又不一样，还需要用到聊天的

功能，这一模块用了大量 topic module，保证多轮聊天的一致性，还实现了一些基于话术模式的工作。

生成式对话系统通过深度学习直接生成最后的回复，目前应用还不是很成熟，其准确率很难达到 90%。生成式方法容易生成单调乏味的短回复，或者一些不通顺的长回复，相关性上也常会出现一些过于通用泛化的弱相关性结果。实现了基于 CVAE 的算法提升鲁棒性和多样性，以及引入相关关键词特征解决相关性问题。基于这些技术还实现了一些有趣功能，如智能评论、古诗生成、自动写对联等。

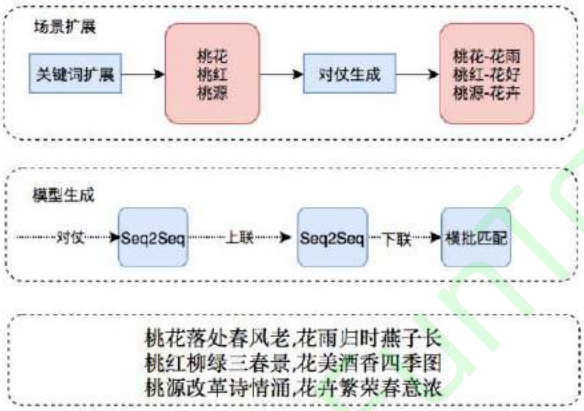
古诗生成



21

2019 AI 先行者大会

自动写春联



22

2019 AI 先行者大会

美团对话理解技术及实践

嘉宾：江会星　编辑：李岩哲

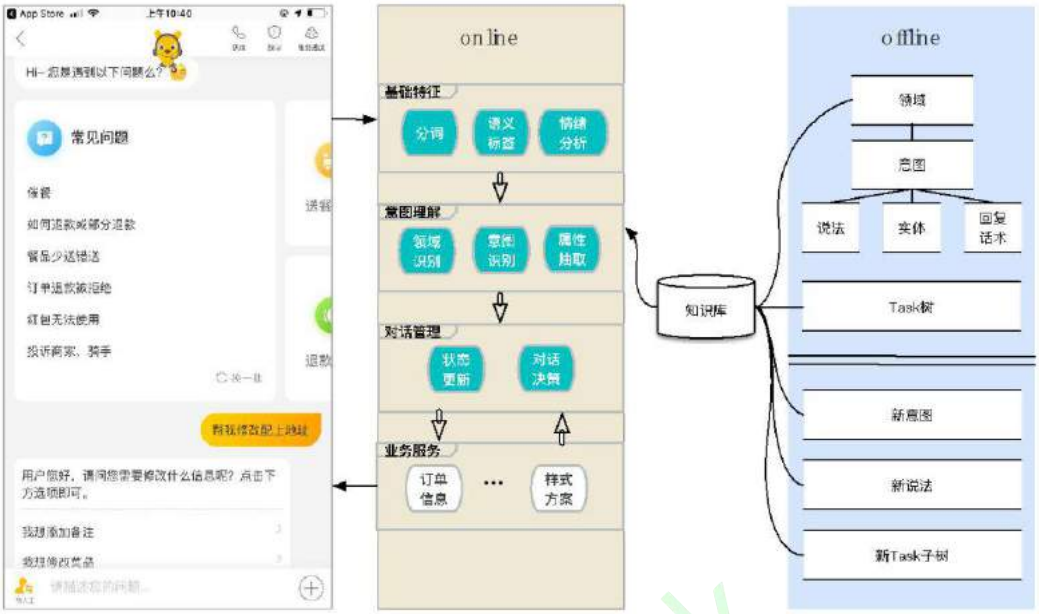
智能客服是一种使用自然语言与用户交互的人工智能系统，通过分析用户意图，以人性化的方式与用户沟通，向用户提供客户服务。

本议题首先介绍美团智能客服的对话交互框架，然后就我们在其中意图挖掘、意图理解、情绪识别、对话管理等核心模块中用到的机器学习算法进行详细的介绍。

美团点评的使命是 " 帮大家吃得更好，生活更好 " 。美团线上服务的 APP 涵盖了生活服务方方面面，包括餐饮、外卖、打车、酒店业务等。2018 年全年美团的总交易金额达 5156.4 亿元人民币，同比增加 44.3%；2018 年美团

单日外卖交易笔数超过 2100 万笔。美团的各个 APP 上具有大量的客服信息类的数据，这也为工程师提供了很好的练兵场。

一、智能客服对话框架

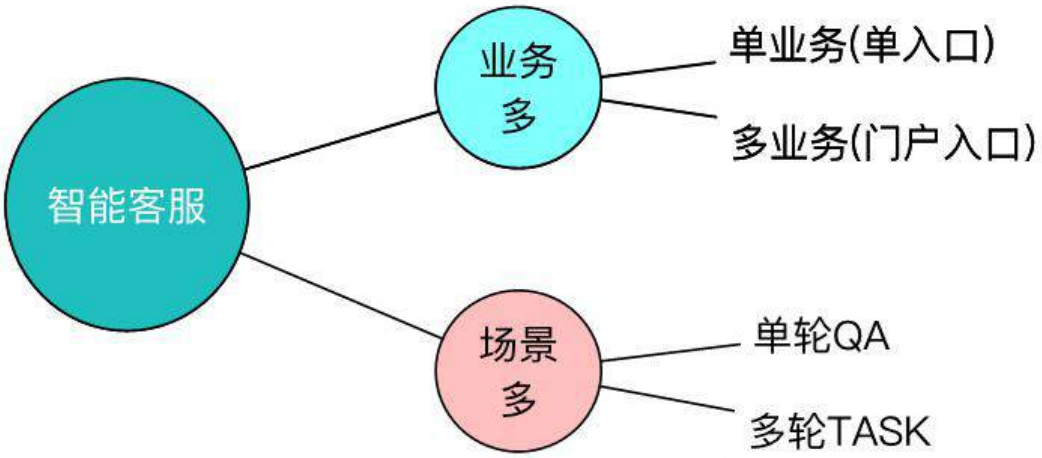


当用户进入到客服界面时往往是带着一个问题来的，那么我们需要做的就是对这个问题进行理解，然后根据理解结果去请求相应服务，去解决用户的问题。这里面主要分为两大块，一方面是离线训练和知识库整理部分，另一部分是在线处理部分。

在在线部分，首先需要对问题进行基础特征的提取，比如：分词、语义标签抽取、情绪分析、NER 识别等；进而进入下面一层-意图理解层，主要有问题领域分类、意图识别和属性抽取；意图理解之后就进入到了对话管理这个阶段，对话管理模块主要包括两个部分，状态追踪（DST）和对话决策，DST

根据上下文状态明确当前用户的领域和意图，而对话决策模块则根据用户当前意图决定后续动作；再之下是业务服务层，包括各业务的数据服务接口以及业务数据呈现样式等。

二、意图理解



美团围绕着生活服务有着许许多多场景和业务，针对客服服务而言，用户可能是从单一的业务窗口进入到客服中，这时我们是知道该客服服务属于哪个领域；用户也有可能是从美团的综合门户入口进入到客服中，在这种情况下我们无法判断用户需要进行哪个业务领域方面的咨询。

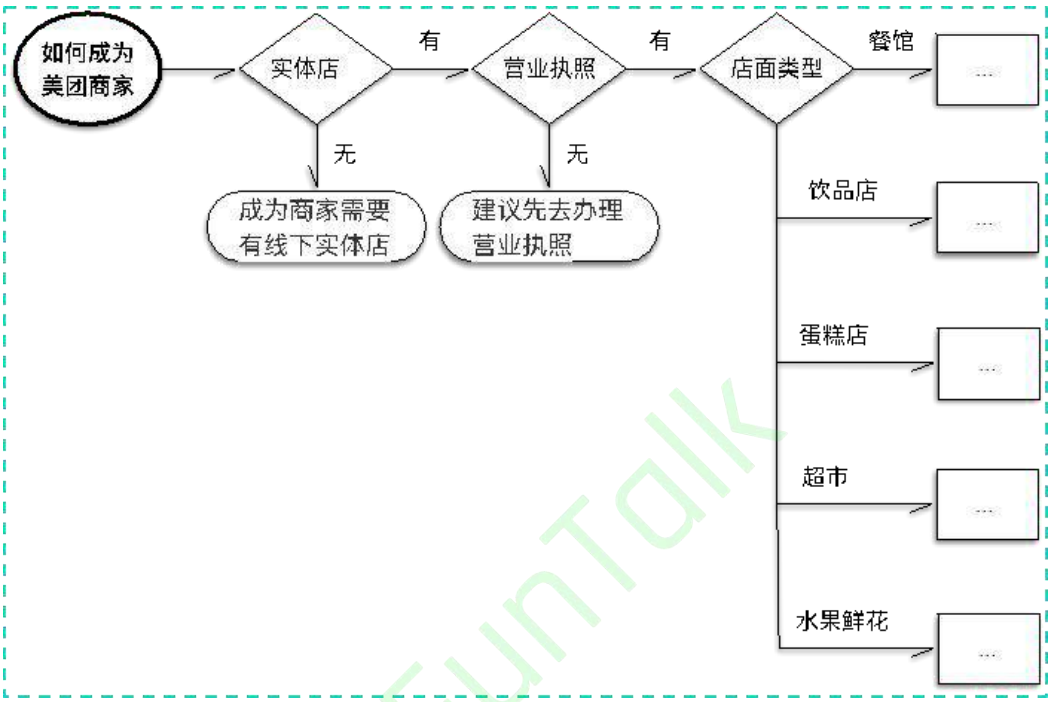
除此之外，在场景方面，主要涉及单轮 QA 和多轮 Task，针对一些简单的问题，单轮的 QA 就可以解决，举个例子：

U: 美团送餐时间

S: 用户您好，能否配送是以商家营业时间为准的。如您所选的商家正在营业，便代表可以提供点餐及配送服务。

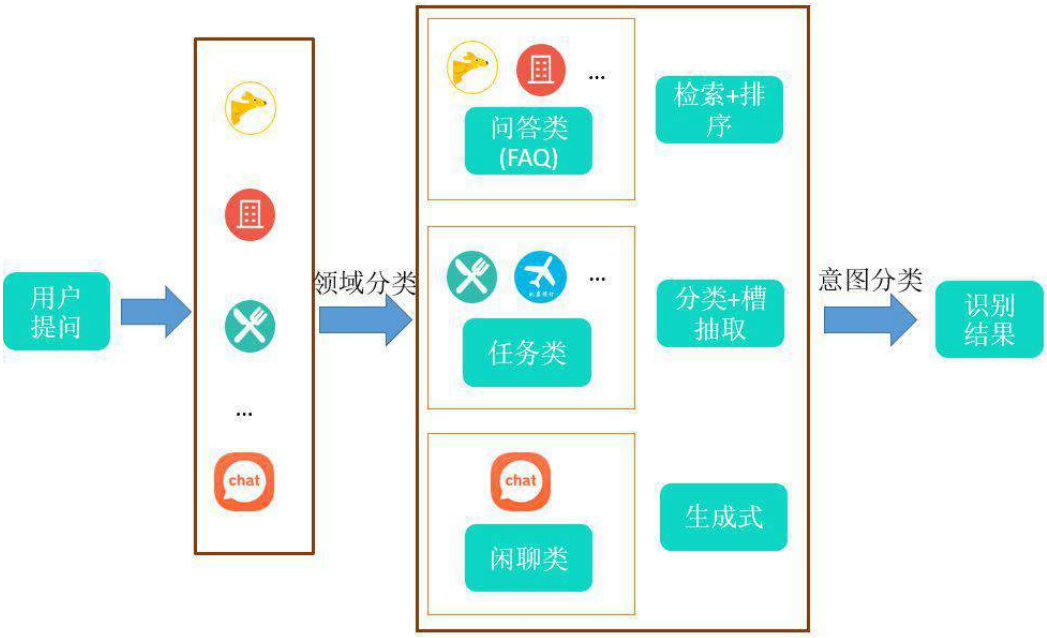
而大量复杂的业务通过单轮的 QA 是无法完成的，这时候就需要多轮对话，举个例子：

如何成为美团商家？



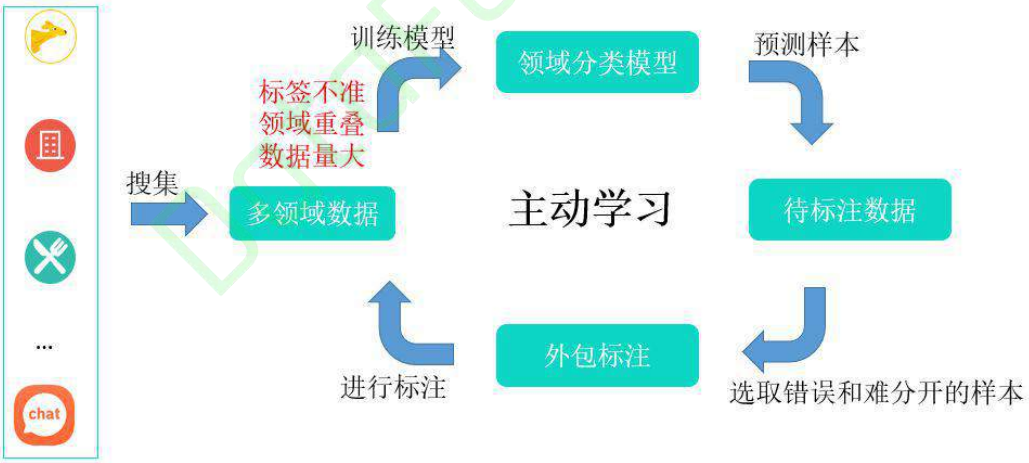
像这种就需要多轮的任务 Task 才能解决。

DataFunTalk



由于美团涵盖众多的业务领域，所以当用户提出一个问题时，我们首先要将这个问题进行领域分类，把它分到某个业务领域去，然后用业务知识去解决这个问题；领域明确后就是意图分类，根据问题的不同类别，比如问答类的、闲聊类的等，采用的方式也会有所差异。接下来将对这两个方面做详细介绍。

2.1 领域分类



针对领域分类任务，如上图所示，我们首先会从不同的业务中收集大量的业务数据，作为基础的训练数据，虽然这些数据来自不同的业务，但是依然存在一些问题，主要有以下两方面：

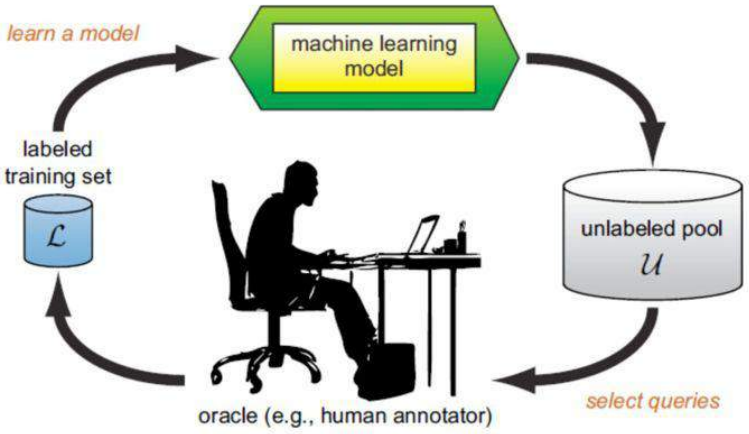
- 标签不准：用户有可能会在某个业务对话中提问其它领域的问题；
- 领域重叠：某些问题可能会在多个领域中出现。

所以，原始数据不能直接拿来作为训练数据，必须要经过人工筛选和标注方可使用。 为了节约人力成本和提高迭代速度，我们采用了主动学习框架，模型的迭代主要分为如下几步：

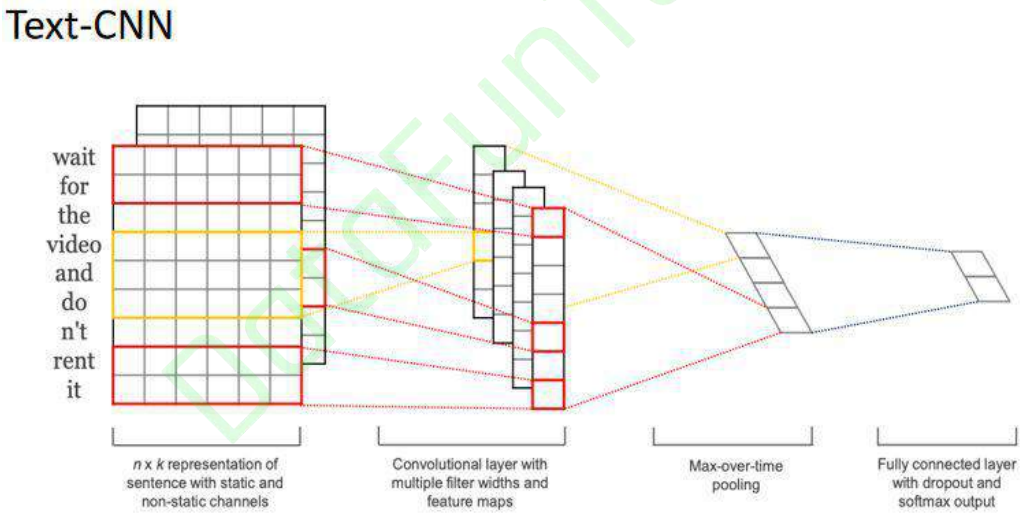
1. 搜集业务数据，为每条业务数据打上相应的业务标签
2. 对数据进行模型训练
3. 将上步训练好的模型对样本进行预测
4. 标注人员对预测样本进行标注，选出错误和难分开的样本
5. 返回第 2 步，对标注好的数据重新进行训练

DataFunTalk

模型	准确率
FastText	91%
TextRNN	86%
TextCNN(字)	90%
TextCNN(词)	92%
BERT	92.5%

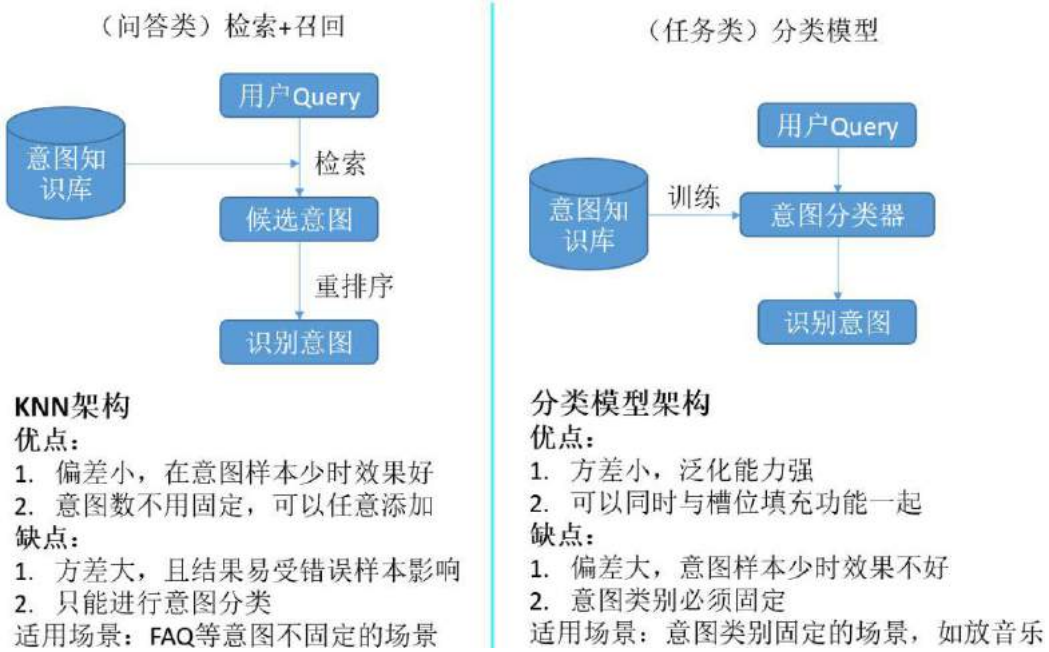


我们同时也在不同的模型上测试领域分类效果，实践中的各模型效果如上图所示。 从结果中我们可以看出，BERT 的效果是非常高的，但是呢我们也会考虑模型在实际运行中的效率问题。 对于一个 15 个字左右的 query 来说，用 TextCNN 模型在 10ms 以内就可以解决，如果用 BERT 模型的话可能需要 70ms 左右，这个时间还是比较长的，当前实际上线的时候我们采用的是 TextCNN 模型。 这是 2014 年 Yoon Kim 提出的一种方法。

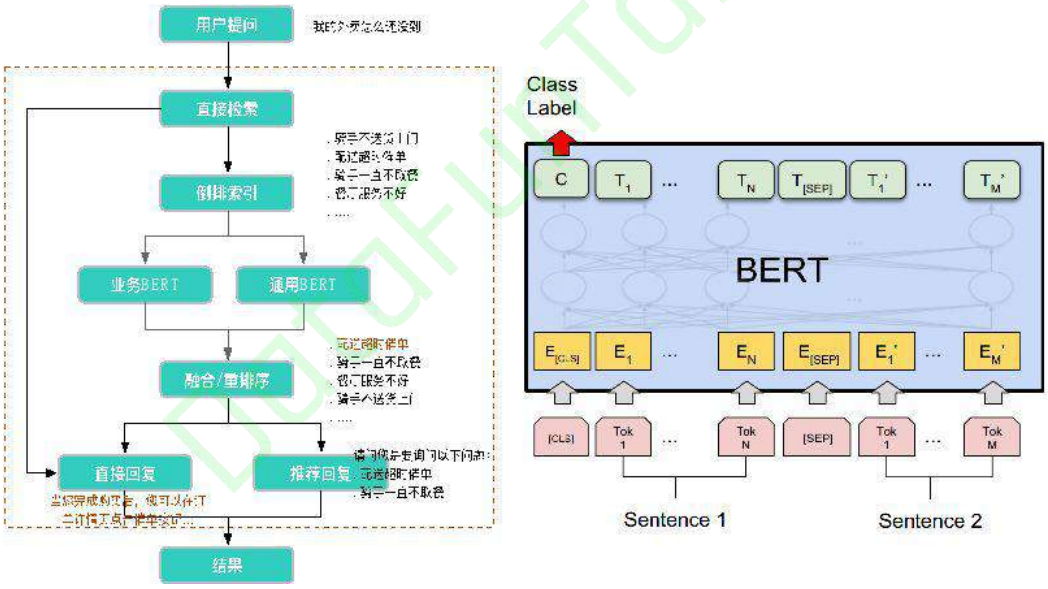


2.2 意图分类

针对意图分类，主要包括问答型意图理解和任务型意图理解两个方面。



这两类问题有着各自的特点，针对问答类，我们采用检索和相似度排序的策略，下图是问答类的设计架构。



针对任务型的意图理解，我们采用规则和模型结合的方式，一种是通过规则的方式，比如上下文无关文法，另一种是采用模型训练的方式。

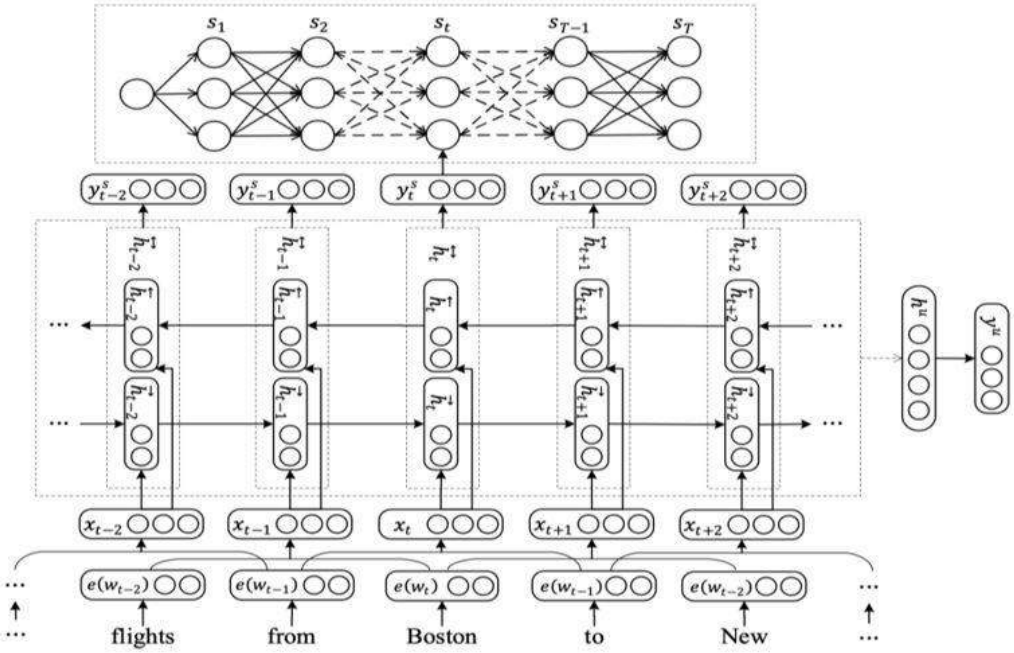
上下文无关文法

```
(订|预订)一家[<sys.DATE>(date)的]<sys.COOKINGSTYLE>(style)(馆|店)
```

上图就是其中一个例子，在工业界这种方式还是非常通用的，对于问题冷启动，高频出现的问题和常规的问题，采用规则的方式能够很好的解决。

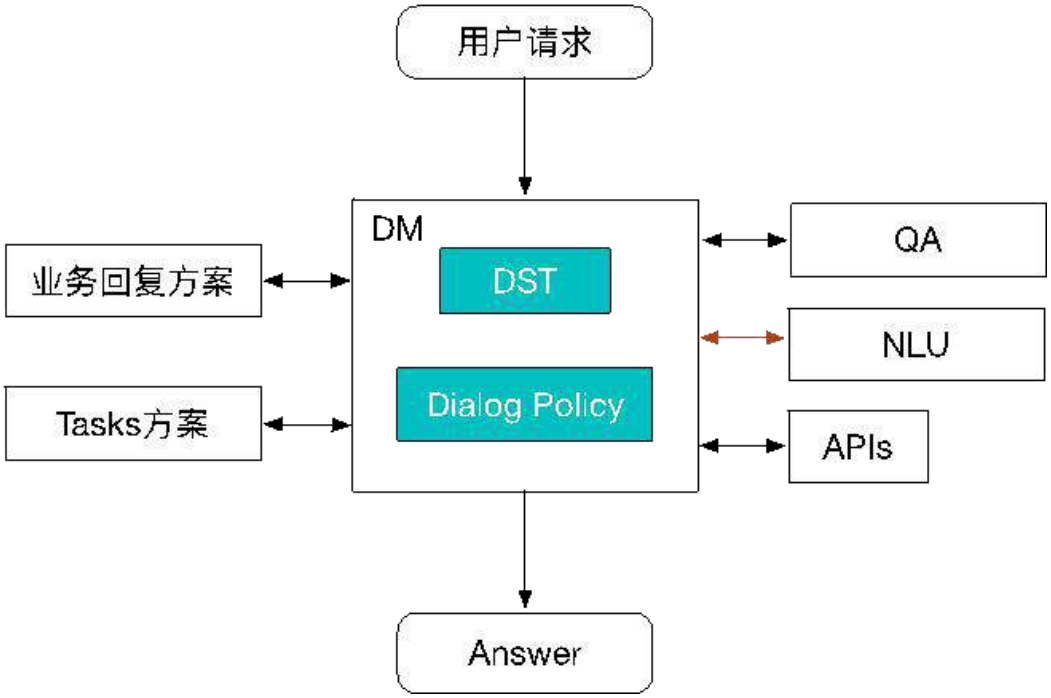
多任务学习模型

DataFunTalk



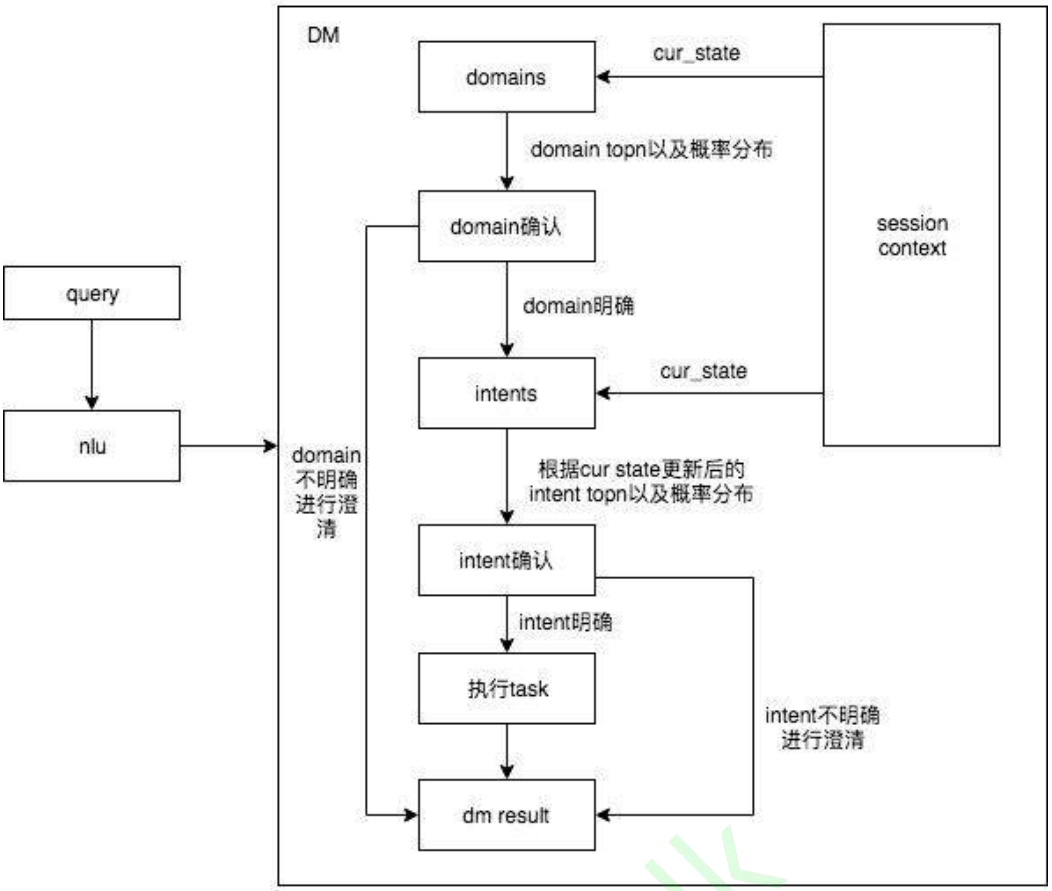
模型部分我们把意图分类和属性抽取联合建模，作为一个多任务学习（Multi-task Learning）任务，如上图所示（算法详见 Zhang, Xiaodong 2016IJCAI）。双向 LSTM 处理后，一方面会通过 softmax 分类输出意图多分类的结果（右边的 y^u）；另外通过 CRF 层，标记每一个词的槽位标签。具体来说，对于 " 帮我找一家明天中午适合 10 人聚餐的川菜馆 "，模型应该能识别出来它是属于订餐意图，同时又能够抽取出时间 " 明天中午 " 、人数 " 10 " 和口味菜系 " 川菜 " 等属性信息。

2.3 对话状态追踪（DST）



DST 解决的还是一个意图的问题，根据当前上下文的环境或者状态来明确当前用户的确切意图。

DataFunTalk



上图是我们当前的框架，这个 session context 为上下文信息，NLU 模块的输出信息可能是多个意图，我们需要根据其他的一些信息，比如，订单信息、门户信息，入口信息等，结合 session context 去明确它是属于哪个领域。

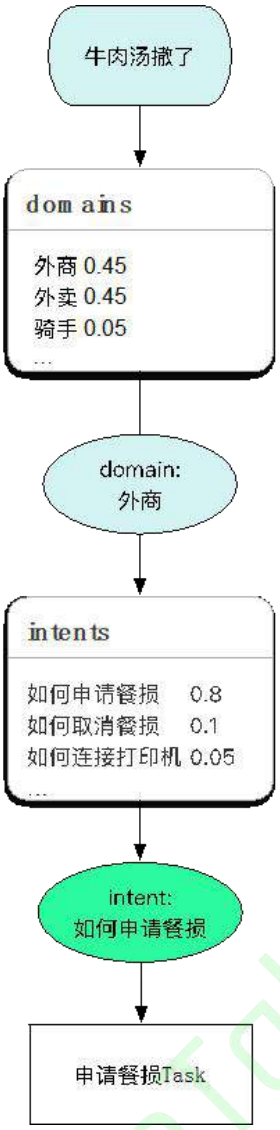
如果这个领域不能明确怎么办？我们的做法是会跟用户进行一轮的澄清，反问用户一次，来解决这个问题，也就是框架最左边的 " domain 不明确进行澄清 " 逻辑。

领域一旦明确后，下一步会进入到意图这一块，我们要明确它当前是什么意图，当然接收到的 query 也面临多意图判断的问题，同样我们也可以去做澄清，澄清包括利用上下文信息判定，或者增加一轮与用户的交互来澄清，如果明确则继续下面的流程，这是我们整体的架构。

举个栗子：

如果接收到一个信息是 " 牛肉汤撒了 "，首选我们要判断它是属于哪个领域的，它是属于外卖商家这个领域，接着判断其意图，对意图进行澄清后得知意图是 " 如何申请餐损 "，然后走餐损的流程。

DataFunTalk

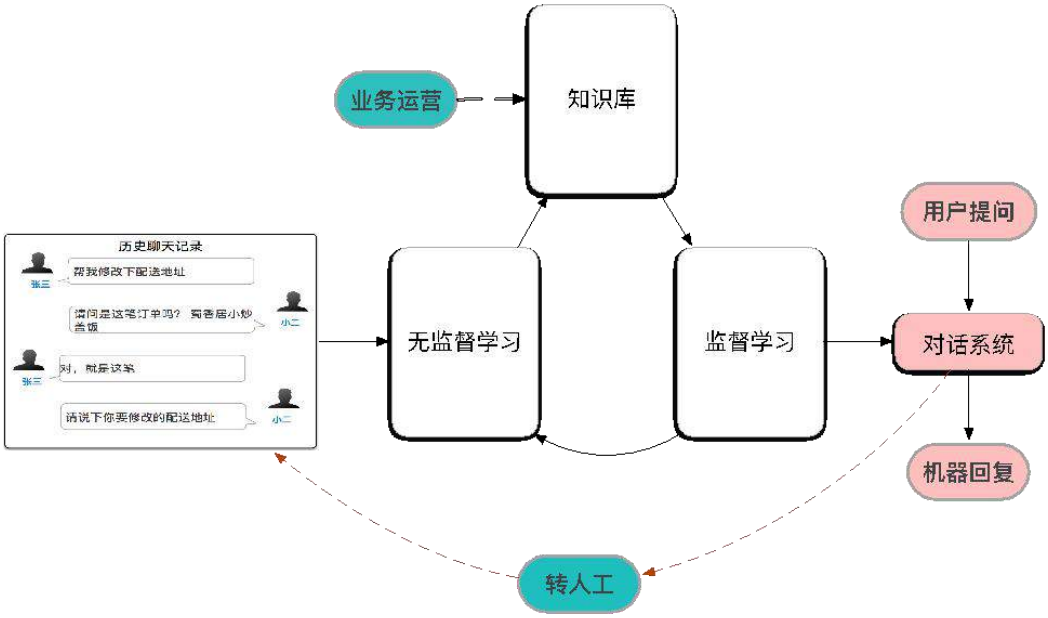


三、知识发现

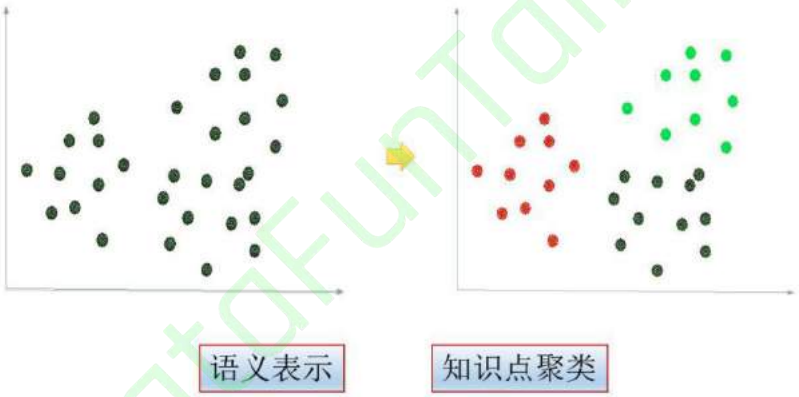
3.1 人在回路

客服的目的是为了解决用户的问题，AI 在现有的 work flow 中节省人力，但是机器解决不了的事情还是要交给人来解决。所以在下图中，我们一定要加一条转人工的服务。另外我们利用无监督学习从日志中挖掘出的知识点也

需要人工 " 业务运营 " 来 check 。在整个环路里监督学习从知识库中学习到的语义表示能力又可以提供给无监督学习使用，这个在下面会进一步提到。

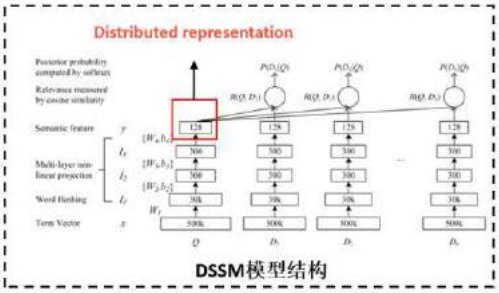


3.2 无监督学习在知识发现中应用

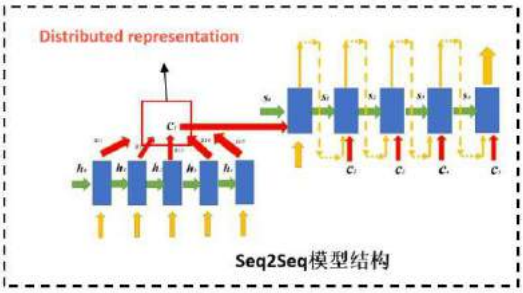


无监督机器学习主要涉及两个问题，一个是句子的语义表示，另一个就是如何做知识聚类。

在语义表示问题上，我们做了大量的试验，在迭代的过程中，我们用到了 DSSM 模型、seq2seq 模型和 BERT 模型来做意图的相似度计算，在这个过程中我们发现不同的模型有各自的特点，它可能抓住不同维度的特征，在离线模型中我们用多个模型的拼接的方式来表示其语义向量。



DSSM模型结构



Seq2Seq模型结构

在知识点聚类这个问题上，我们采用用了最通用的 K-means 模型来做的。

k-means算法

1. 随机地选择k个对象，每个对象初始地代表了一个簇的中心；
2. 遍历余下所有对象，根据其与各簇中心的距离，将它划分给最近的簇；
3. 计算每个簇的平均值，并作为该簇新的中心；
4. 重复2-3，直到准则函数收敛(k个中心点不再变化或执行了足够多的迭代)。

k及聚类结果评估

凝聚度 + 分离度 → 轮廓系数

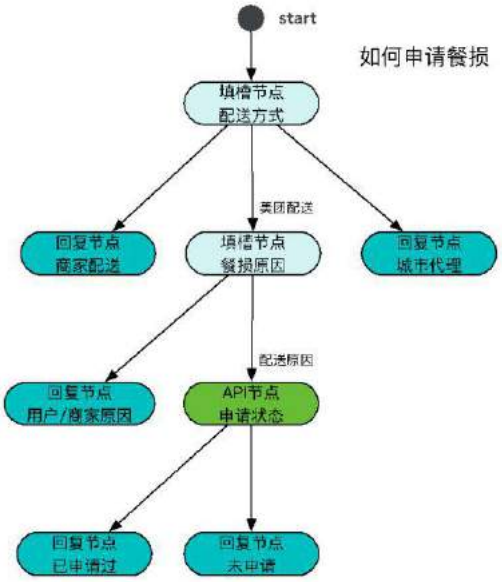
1. 对于元素x_i，计算x_i与其同簇内所有元素距离的平均值，记作a_i；
2. 选取x_i外的簇b，计算x_i与b中所有点的平均距离，遍历所有其他簇，找到最近的这个平均距离,记作b_i；
3. 对于元素x_i，轮廓系数$s_i = (b_i - a_i)/\max(a_i, b_i)$；
4. 计算所有x的轮廓系数，求出平均值即为当前聚类的整体轮廓系数。

上面讲到的是意图和说法的挖掘，在实际业务中我们有大量的 Task 的问题。下图呈现的是一个 " 如何申请餐损 " 的 Task 树。

知识发现-子树挖掘

美团点评

- Task树



当用户的问题触发 Task 后，Task 机器人根据和用户的交流来获取槽信息，调用不同 API 接口来获取槽信息，进而回复用户。上图 " 如何申请餐损 " Task 需要明确的槽包括 " 配送方式 " 、" 餐损原因 " 、 " 申请状态 "，其中 " 配送方式 " 和 " 餐损原因 " 是通过与用户的交互来明确，" 申请状态 " 则是通过请求后台服务来明确。

在这个环节中我们需要做的就是辅助运营人员构建 Task 类的知识。

我们根据用户的日志数据提取相应意图，然后根据意图共现，回复共现去挖掘，当一个用户问了一个问题之后还会提问哪些问题，当用户收到反馈之后还会反问哪些问题。根据这些去构建 Task 子树，离线构建好之后交给运营的同学，运营同学审核通过之后就可以上线了。

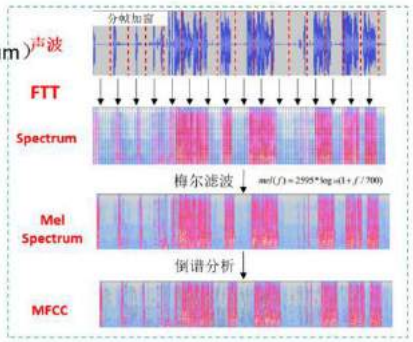
四、情绪识别

4.1 背景介绍

客服热线是我们公司对外服务的重要交流通道，在售前、售中和售后的各个环节中发挥着重要作用，为用户提供意见处理、资料管理、技术支持等多项服务。然而目前客服热线在运营过程中还存在一些痛点，如客服人工坐席的服务水平参差导致客户的体验存在差异，另外个别客户还存在动机复杂等问题。因此如何利用技术提升客服热线的服务水平、检测热线中可能的风险是目前需要解决的一个问题。本项目对客服热线中的语音数据进行挖掘与分析，通过量化用户的情绪能量值，实现对用户情绪状态（是否激动、情感倾向等）的追踪，并且在客户情绪超过设定阈值时提供预警信息，以便相关人员和应急措施的及时介入，从而为相关业务部门提供运营数据支撑和智力支持，最终提升客服热线的服务质量和服务效率。

4.2 特征提取

- 原始特征
  - 频谱图(Spectrum)
  - 梅尔频谱图(mel-Spectrum)
- Handcraft 特征
  - 梅尔倒谱系数(MFCC)



FTT：短时傅里叶变换

每帧语音都对应于一个频谱（ 通过短时 FFT 计算 ），频谱表示频率与能量的关系。

梅尔滤波：实验观测发现人耳就像一个滤波器组一样，它只关注某些特定的频率分量（ 人的听觉对频率是有选择性的 ）。也就说，它只让某些频率的信号通过，而压根就直接无视它不想感知的某些频率信号。

4.3 模型选择

特征处理完成之后就是采用哪种模型来进行训练。

在迭代中我们采用过传统的机器学习模型，比如 LR、SVM 模型，神经网络模型，和一些预训练好的模型，在这个里面我们遇到的一个挑战就是，一个情绪是不是激动的标签是针对整个通话记录标注的，但是用户在通话的过程中，不是一直的激动，而是在某个通话阶段情绪激动，而一个标签无法体现

出到底是那一部分激动，是全程激动，还是部分激动，还是全程平静的，其实这个里面就涉及到一个弱标签的学习，如下图所示。

方案1：Decision Level Attention　　方案2：Feature Level Attention

[Qiuqiang, 2019] Weakly labelled AudioSet Classification with Attention Neural Networks　2019 AI 科学前沿大会

这是 19 年提出的一个算法，在实际应用中效果不错，感兴趣的同学可以根据信息去查找。

在实际的效果上，各个模型的表现如下：

MFCC + LSTM < MFCC + CNN < VGGish + ferture level attention < VGGish + decision level attention

五、展望

- 多轮上下文建模，意图理解
- 让用户做选择题，做意图预判，意图推荐

- 语音与文本多模态，弱标记学习，情绪风险识别

- 对话历史的话题抽取及切割，话术推荐，坐席助理

以上是当前我们正在开展及探索的智能客服理解部分的内容，从 ToC 的用户侧，以及 ToB 的坐席助理侧两方面来优化整个客服闭环。

DataFunTalk

智能机器人在滴滴出行场景的技术探索

嘉宾：熊超　编辑：Hoh Xil

本次分享是在 2019 年 AI 科学前沿大会上的分享 ,主要介绍智能对话机器人在滴滴出行场景中的技术探索，主要内容为：

- 单轮问答
- 多轮对话
- 整体架构

单轮问答

单轮问答指识别用户问题，并给出相应答案。这种场景下的目标是做到识别准确，尽量理解用户问题，给出合适的答案。

DataFunTalk

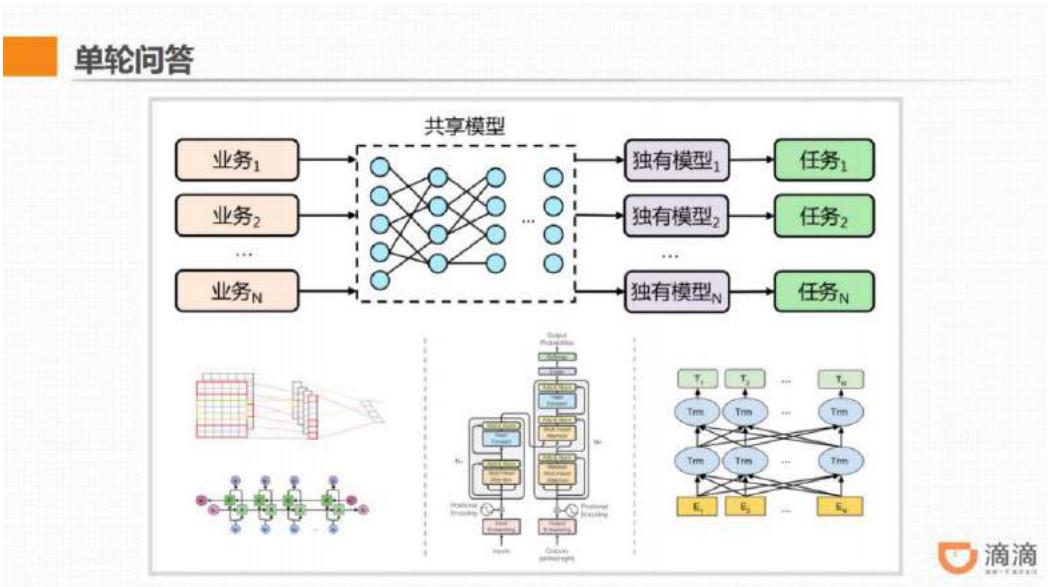
开发过程中的难点和挑战：

数据：标注数据少，这是 NLP 领域的痛点问题，因为标注成本相对较高；

业务：业务线比较多，我们目前支持滴滴场景下的业务线有 10 多个，会导致数据标注的问题更突出，数据量少的业务，可标注的数据更少。

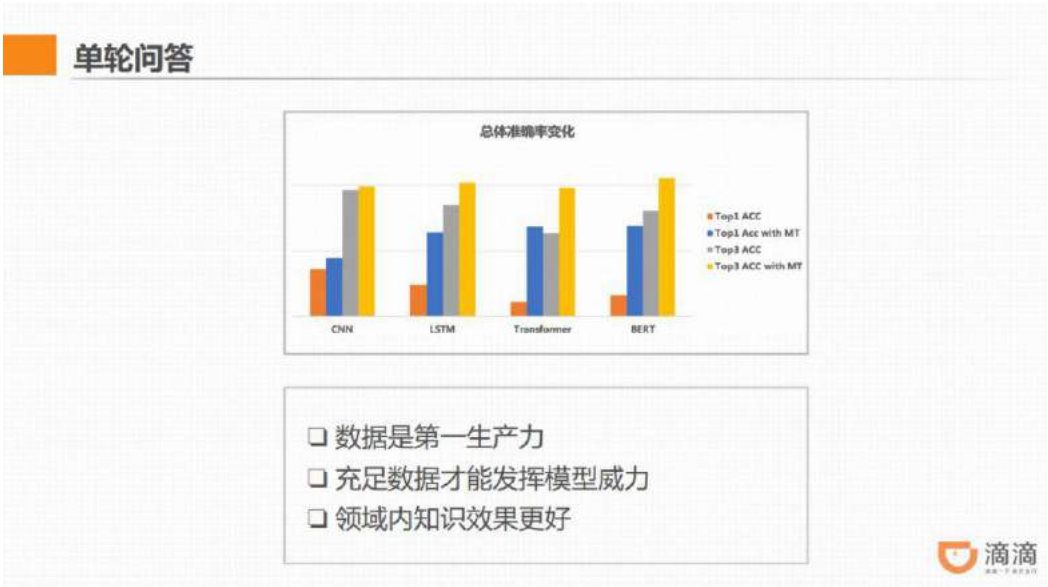
语言：用户的表达方式灵活多样，即同一个语义有多种表达方式。

针对上述问题，我们想了一些办法，分析了滴滴场景下和其他智能客服的区别，比如快车和专车业务线，都是由不同模型来支持的，但是快车和专车业务其实是非常相似的，经过统计分析，二者知识点重复率接近一半。我们考虑是否可以把大业务线的数据迁移到小的业务线，但是当我们细看数据的时候，发现还是不一样的，因为不同业务场景下的相似问，还是有区别的，比如业务独有的知识，不能直接用在其他业务线上。

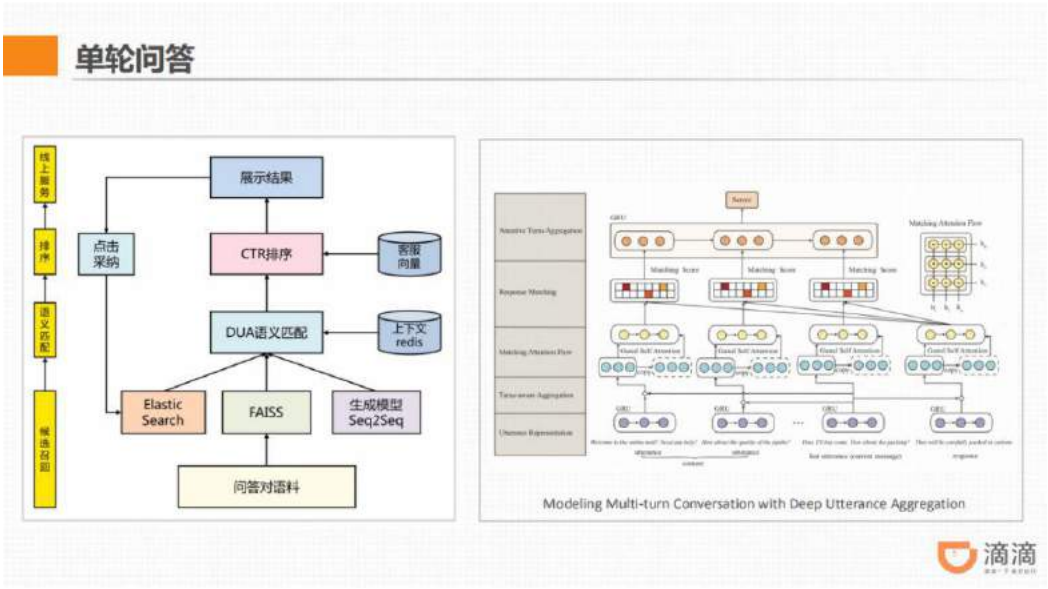


为了解决这些问题，我们一直在想数据如何更好的去迁移，减少数据的标注量。提出了类似 Multi-Task 多任务学习的架构，因为我们有不同的业务线，如果不考虑 Multi-Task 结构的话，每个业务线会有一个模型。有了 Multi-Task 之后，可以多个业务线共享一个语义模型，让模型的泛化能力更强，为了解决不能直接映射的问题，每个业务线还有独立的模型在后面，优化各自的目标。语义模型可以有任意模型，我们尝试过 CNN、LSTM、Transformer、Bert 等。

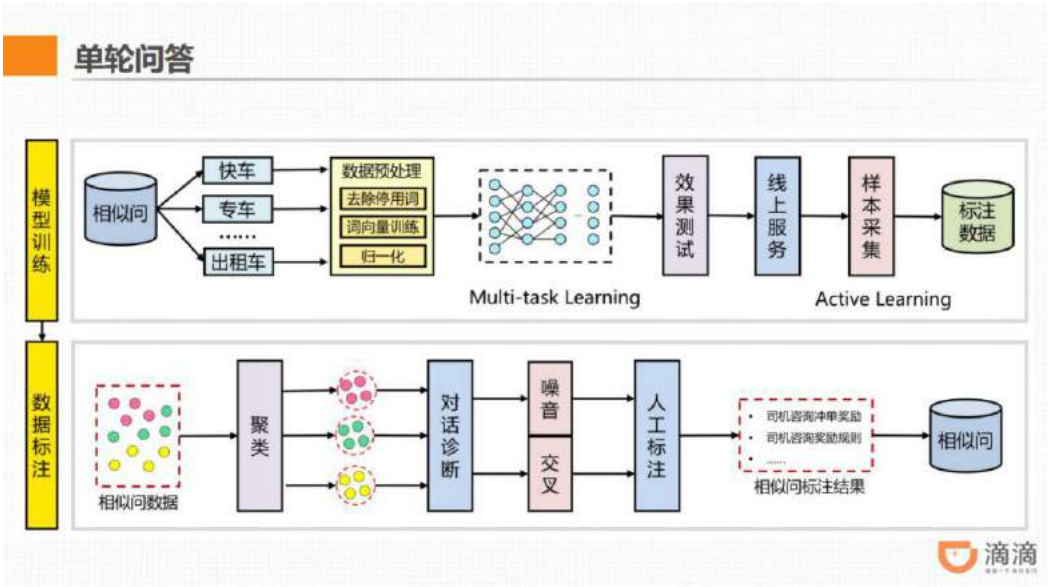
DataFunTalk



上图为我们加上 Multi-Task 之后的一些实验结果，包括 CNN、LSTM、Transformer、Bert ,其中 ,橙色和蓝色为 Top1 准确率 ,灰色和黄色为 Top3 准确率，橙色为模型本身的结果，蓝色为模型+Multi-Task 之后的结果，从结果上看，CNN+Multi-Task 后有一定的提升，从这一点上看 Multi-Task 还是有帮助的，进而我们做了更多的实验，比如 Bert+Multi-task 的 Top1 准确率相比于 CNN 有了显著的提升，在本身没有增加新的成本的情况下，提升显著，为什么加了 Multi-Task 后结果这么好呢？我们发现，新的模型特征抽取的能力比较强，但是也存在一些特点，需要足够的数据，才能让模型发挥出能力，我们看四个 Multi-task 模型对比（蓝色），给了充足的数据后，效果提升明显。效果好是不是因为模型好就可以了？也不是，其实如果单独业务线，同样的数据下，从图中不使用 Multi-task 模型结果（橙色）的对比可以看出 CNN 的效果反而更好。原因是在数据不充足的情况下，复杂的模型参数更多，容易引起过拟合。



除了分类的结构，我们也尝试了搜索+语义匹配+排序的架构，主要是用来做情绪安抚，思路是把候选的问答对语料，通过搜索、生成式模型得到候选，然后经过粗排，粗排是用文本相关性的分数来计算，最后交给多轮对话深度匹配模型 ,主要参考了去年这篇的论文 :Modeling Multi-turn Conversation with Deep Utterance Aggregation ，DUA 的特点是除了计算当前的对话，还会把上下文建模进来，重新考虑。比如情绪回复，如果是一个负向语句，如果单看这句话，它的回复可能是非常通用的，但是结合上下文，比如有的司机听不到单了，然后他会回复一些负面语句，这时我们的回复是针对听单场景的安抚。



除此之外，我们还有些离线的工作：

模型训练：如上图，为 Multi-Task 整体的一个效果，我们建立了一个每天模型自动更新的 pipeline，包括自动测试、自动上线。刚刚也提过了，数据很重要，我们会标注新的数据，来解决新的问题的出现，所以我们采用的是主动学习 Active Learning 的思想，去对边界样本进行采样，这样标注效率会更高，构建模型训练及在线服务的闭环，来达到每天模型更新的效果，让新的知识、新的问题，更快的更新到我们的服务上来。让机器人有了自我学习进化的能力。

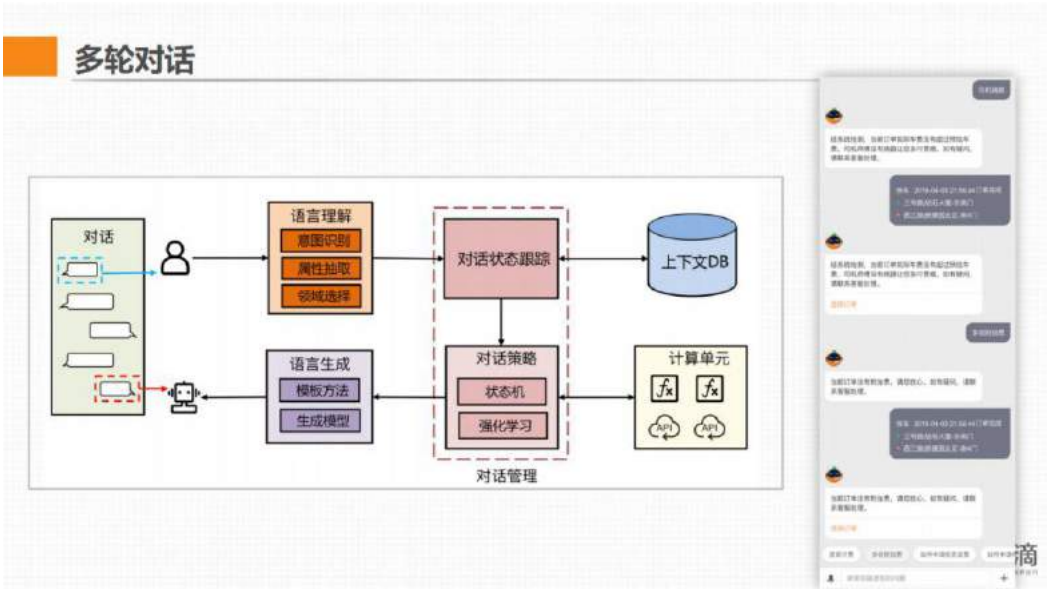
数据标注：其实在现有的标注语料中，还存在噪音，准确率没有那么高。我们通过聚类的方式，把已经标注的语料聚类，这时有些样本是偏离聚类中心的，然后把偏离的样本通过人工检查，如果真的错了，就可以把噪音删除，如果是对的则保留。

多轮对话

在出行场景下，存在俩大类的问题，一类是咨询了问题，比如用户需要咨询一些政策、规则等信息；还有一类是寻求解决的，这两类问题，单轮问答都很难解决用户问题，为此我们提出了多轮对话。

1. 整体架构

DataFunTalk



我们可以看下这个例子，比如有乘客反馈，司机绕路，如果是单轮的话，只能给一个答案，而我们现在可以通过交互的方式来引导用户去选择订单，选择订单之后，我们可以直接调用后台的接口服务能力，去判断是否绕路了，如果真实存在，我们就会直接在机器人里把多收的费用返还给乘客，提升了用户体验。

具体的方法：将传统的多轮对话，多轮交互，引入滴滴客服机器人。主要包括几大模块：

① 语言理解

意图识别，知识点的识别，明确问的问题是什么

属性抽取，可以理解为选择订单，日期等等

② 对话管理

对话状态跟踪：结合当前语义理解的结果，并结合历史对话，上下文综合来看，得到对话的状态（Act 和 slot）

对话策略：给定对话状态，选择对应的动作，目前主要采用状态机的方式，并尝试强化学习对话策略

③ 语言生成

有了动作之后，我们就需要生成用户可以理解的语言。

以上是多轮对话的整体架构。

2. 语言理解

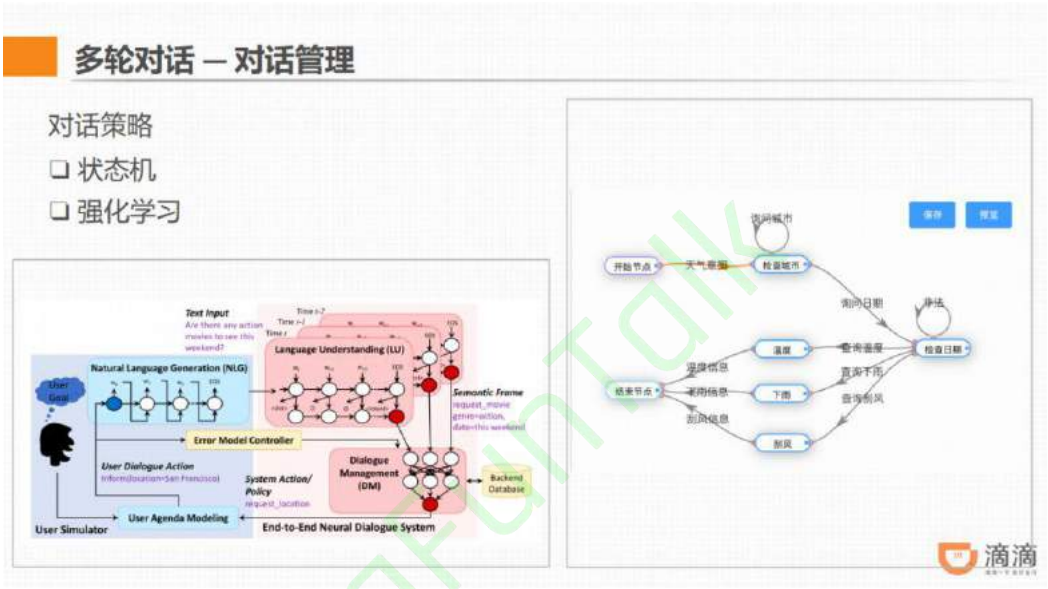
意图识别：

我们采用的模型为 BERT + Multi-Task Learning

槽位抽取：

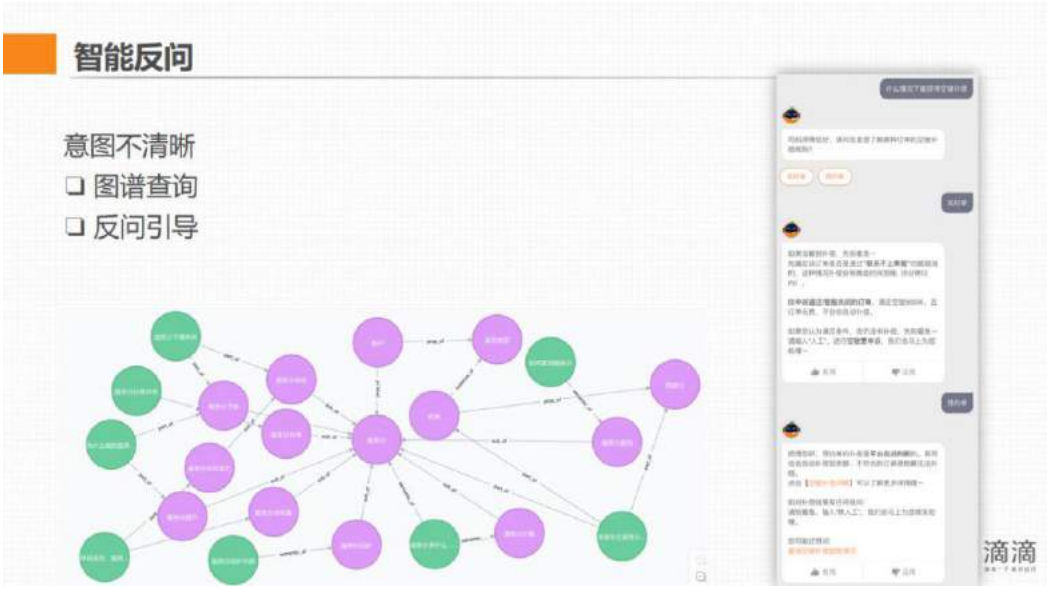
我们主要是基于规则和模型结合的方法，如选订单的组件，模型如 BILSTM + CRF 模型， 来对槽位信息进行抽取。

3. 对话管理



这个刚刚有介绍过，右图为状态机，基于规则配置，左图为我们在研发的强化学习模型，它需要一个用户的模拟器来模拟用户，抽样用户目标，根据目标和机器人去交互，从交互中生成经验，再根据经验进行学习，达到自动学习的效果，而不是像右边状态机，是由领域内的专家来配置的。

4. 智能反问



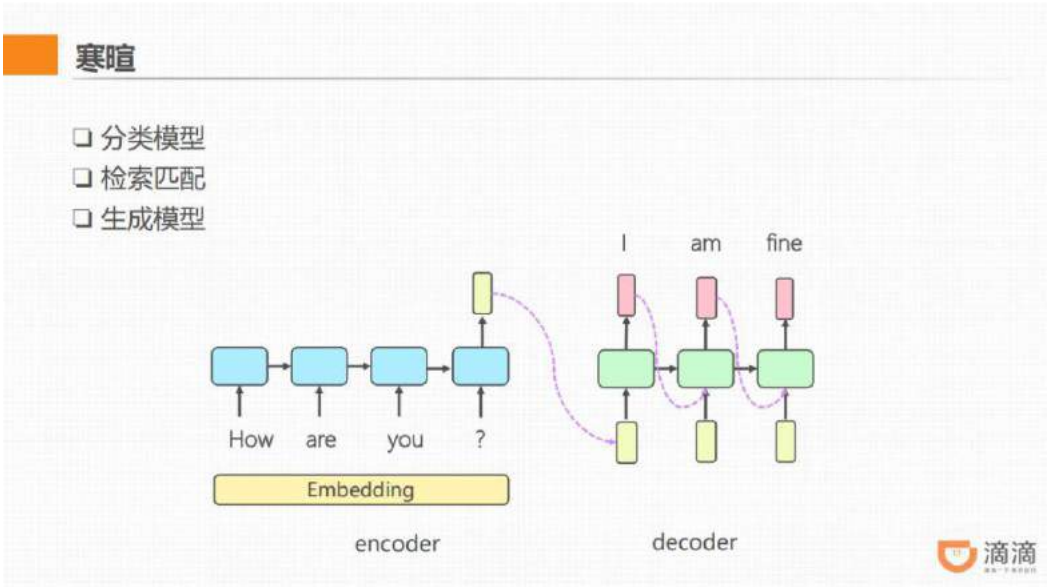
如果用户表达的意图不清晰，无法精确定位问题的时候，我们采用了智能反问技术：

图谱查询：通过图谱去查询，得到相关联的知识点。

反问引导：产品形式上，在这个例子中，我们会引导用户，会问用户是实时单还是预约单，用户只要选择之后，会给用户推送一个更具体的、有针对性的答案。

5. 闲聊-寒暄

DataFunTalk

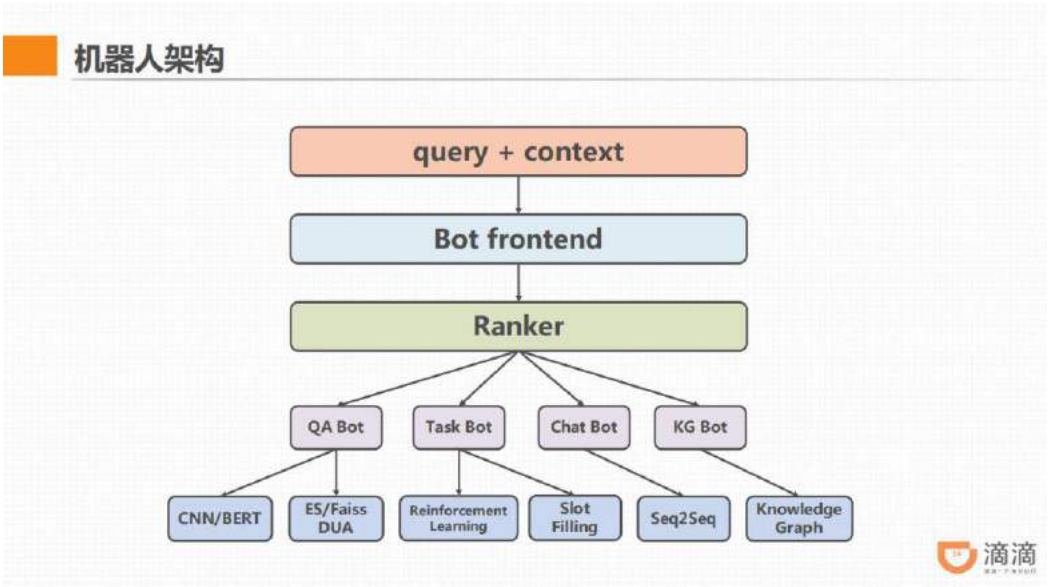


机器人里都会涉及到闲聊，比如“你好”，“谢谢”之类的。针对这些问题做的工作有：

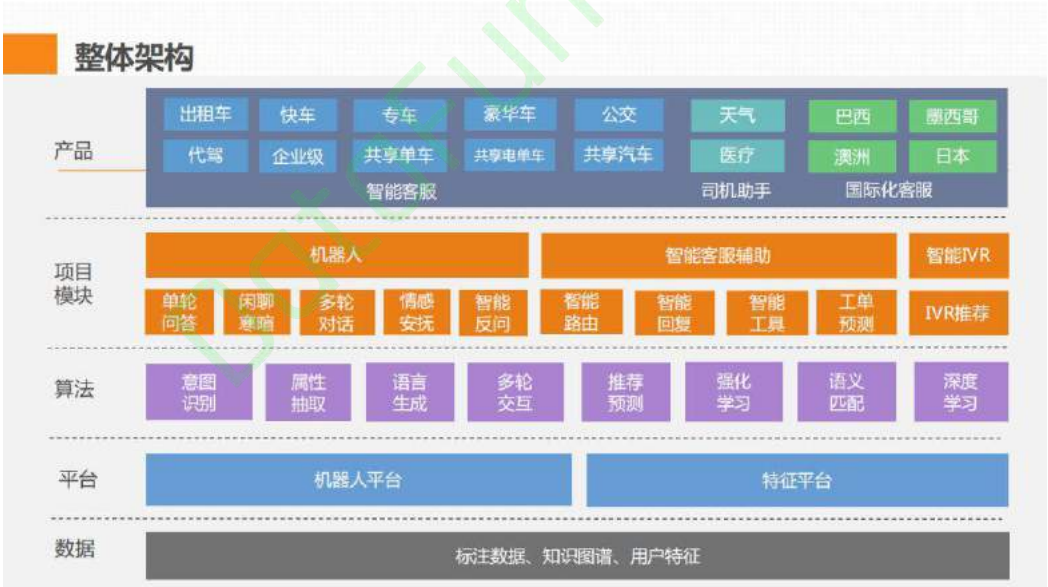
分类模型、检索匹配等，专家编写的答案，现在我们在探索的是生成模型，让答案更灵活。

机器人架构

DataFunTalk



我们整体看下机器人的架构：用户的请求来了之后，将“查询”和“上下文”作为输入去查询 frontend，frontend 作为机器人的中控，也会包括一些业务逻辑，然后通过 ranker 模块做分发和选择，下面有问答型、任务型、多轮对话型、闲聊型、图谱型等，综合的做一个仲裁去选择，给到用户一个最终的答案。



最后讲一下智能客服的整体架构：

产品：我们支持的业务，包括智能客服（出租车、快车、专车等一系列业务）、司机助手、国际化客服等。

这就是我们整体的架构，这就是我今天要分享的内容，谢谢大家。

DataFunTalk

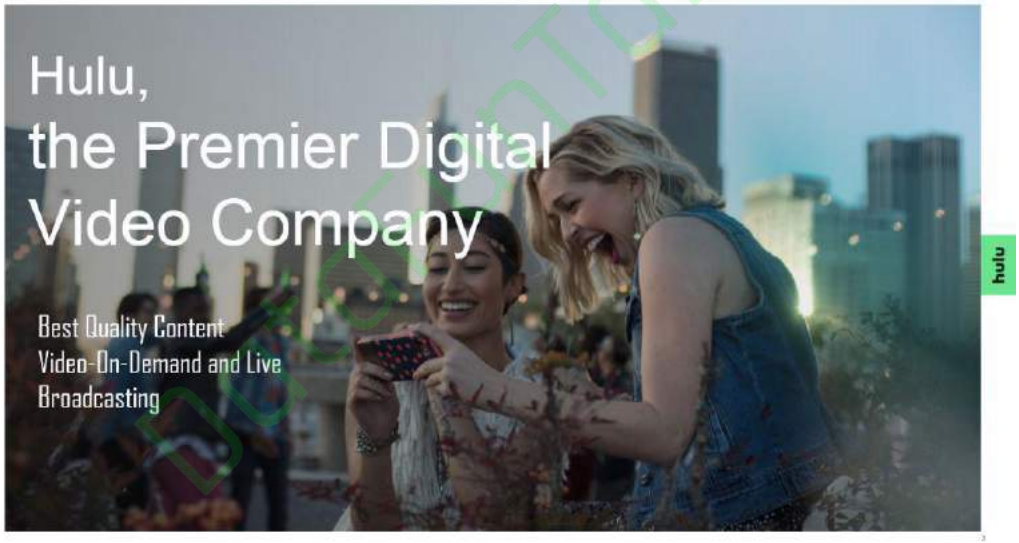
Hulu 在视频内容理解上的应用和最佳实践

嘉宾：谢晓辉　编辑：Hoh Xil

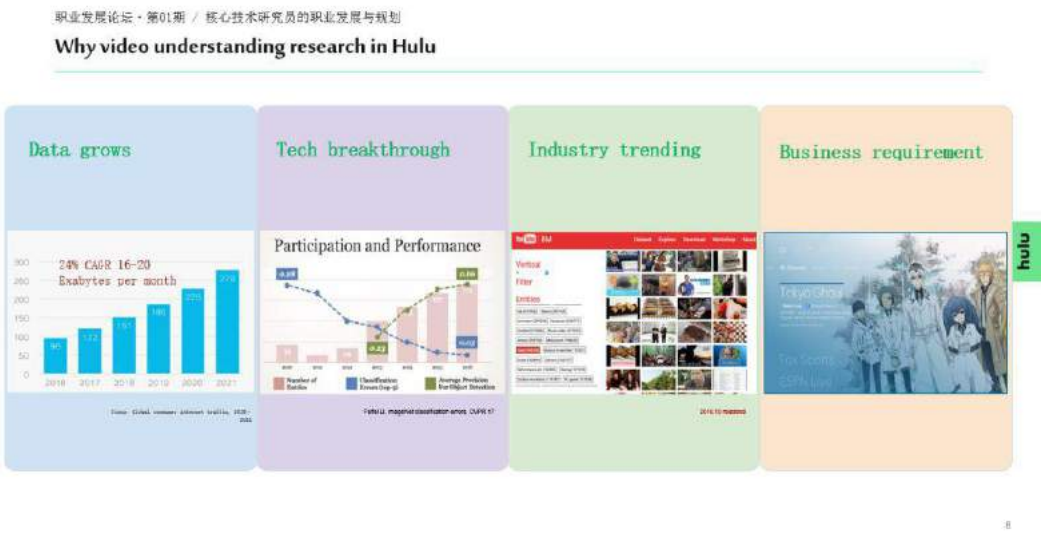
本次分享的内容有：

- Hulu 简述
- 为什么研究视频内容理解
- 面临的挑战
- Hulu 的视频理解实践应用

Hulu 简述



Hulu 是美国的一家视频网站，类似国内的爱奇艺、腾讯视频和优酷，目前 Hulu 相当于迪士尼旗下的子公司，Hulu 在北京有研发中心，其中大部分研发骨干来自北大、清华等名校。



为什么做内容理解

Hulu 是 16 年之后开始系统性的做内容理解，为什么是 16 年？这里有些分析：

随着互联网上的视频流量越来越成为主体，有 90%都来自视频，文本的比例越来越低；

在 12 年深度学习起来之后，为视频的内容理解提供了技术支持，

越来越多的大企业和研究机构投入到了视频理解的研究上。

在 Hulu 内部的原因是，推荐模型在往更深层次走，如果不理解视频的内容，是没办法完成的。

面临的挑战

- 数据的多与少。我们虽然有很多的视频影视剧和用户 ,但是只有很少的标注。
- 数据的真与假。有很多电视剧，动画片，里面有很多场景是假的，这种假的数据很难被利用。
- 技术的难与易。借助一些算法和算力的发展，研究视频理解成为可能。实际上，无论把深度模型神经网络做多深，在视频上的整体性能提升非常有限，这说明针对视频的处理，需要一些特殊的技巧，不能把图像上的一些模型，直接拿到视频领域来用，会有时序上的约束以及语义上巨大的鸿沟。

- 企业的买与研。为什么要自己做，而不是买第三方的工具或者第三方公司？因为不可能把第三方的解决方案，直接拿过来就能用，因为 hulu 的产品形态和对外提供的服务直接和算法挂钩，如果等其他公司提供的结果，会滞后于市场。

当我们开始做内容理解，我们这几年主要做了如下工作：

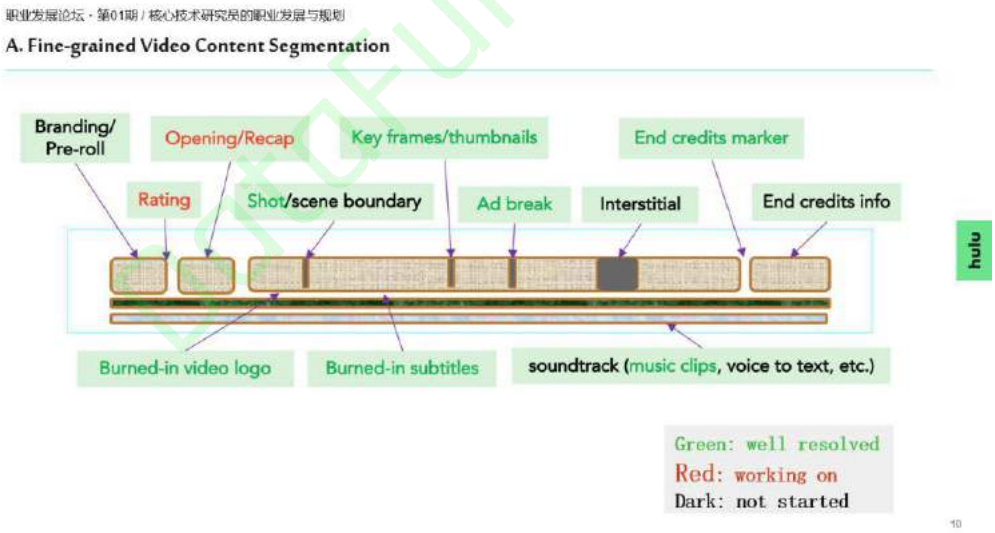
A. 细粒度的视频内容分割

B. 视频标签

C. 内容生成

D. 内容向量

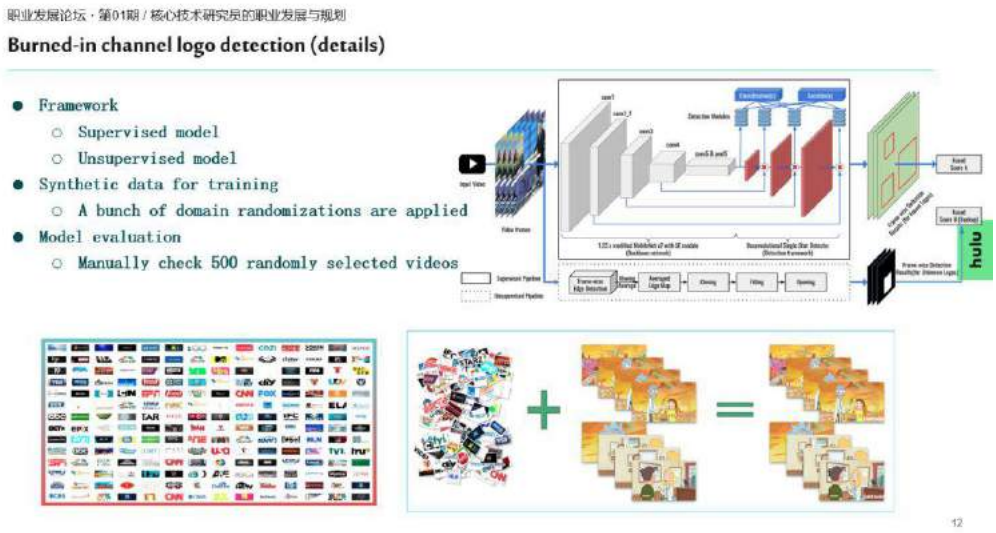
A. 细粒度的视频内容分割



内容分割：目前在 Hulu 对视频的精细化切分已经做得比较好了，可以细粒度的把镜头、场景、前情、片尾、字幕、音乐、分集标志等分开。这里主要讲下：

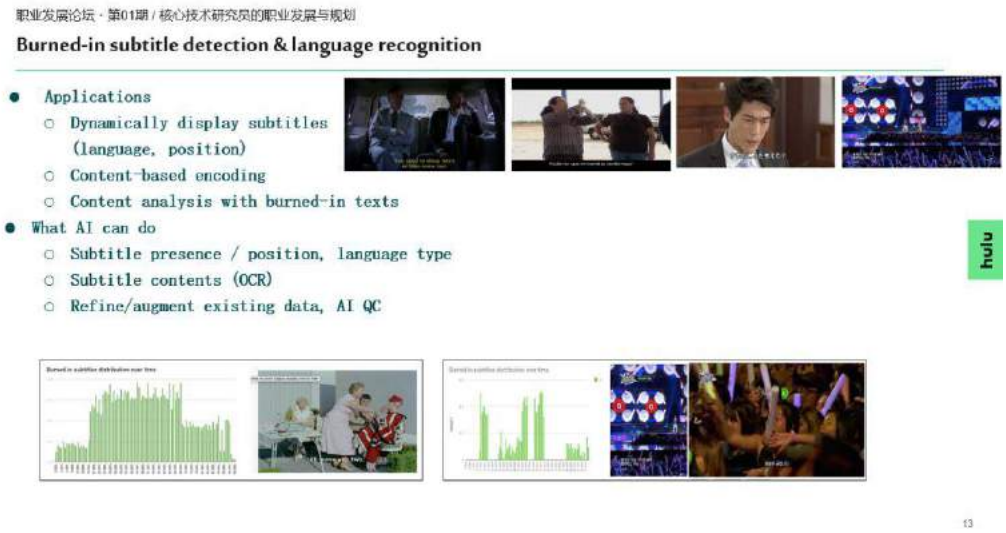
1. 内置频道 logo 检测

我们会检测节目上是否打上了一些 logo，大家可以看到图中有 tbs 或者 cnn 的 logo。为什么要做这件事儿？因为 Hulu 是一个内容形态，有 500+ 内容商为 Hulu 提供内容，在提供内容的时候，提供商会插入很多 logo，由于我们在将这些内容投放到其他平台或频道时，根据投放协议，会加入投放平台的 logo，这样就会有两个 logo 叠在一起，用户体验就会变差。还有就是广告商的需求，比如广告中什么地方出现宝马的 logo，什么地方出现可口可乐的 logo。



在 Hulu 目前检测有 300 个频道的 logo，使用 mobilenet+SSD 来检测 logo 位置。当做完这些之后，还发现个问题，因为某些频道的 logo 会变，可能是没见过的新增频道 logo 或者是新设计的，这时我们会采用非监督的算法来做一个预估，来检测某个位置可能是一个 logo，然后通过人工来确认。当然，做 logo 检测的时候没有标注数据，是在 Hulu 的数据上通过人工嵌入进去，再加入噪声等处理，然后自己生成的一套 data set。

2. 内置字幕检测 & 语言识别

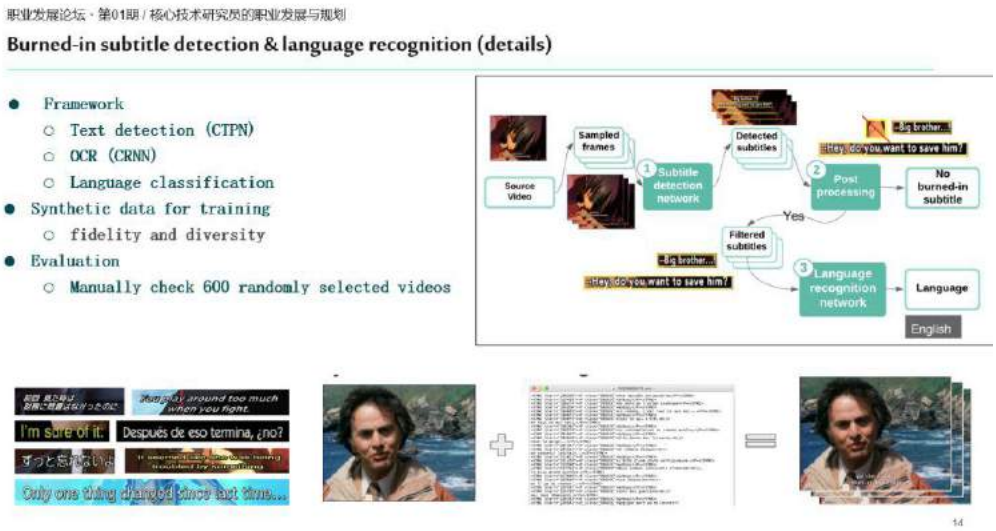


然后是检测有没有字幕，字幕检测有俩个用途：

第一个是我们不希望已经有了英文字幕，用户还再添加字幕；

第二个是视频编解码需要，因为视频编解码之后文字可能会变得模糊，所以编解码的时候可以对字幕做特殊处理，让文字依然保持清晰，可以提升用户体验。

DataFunTalk

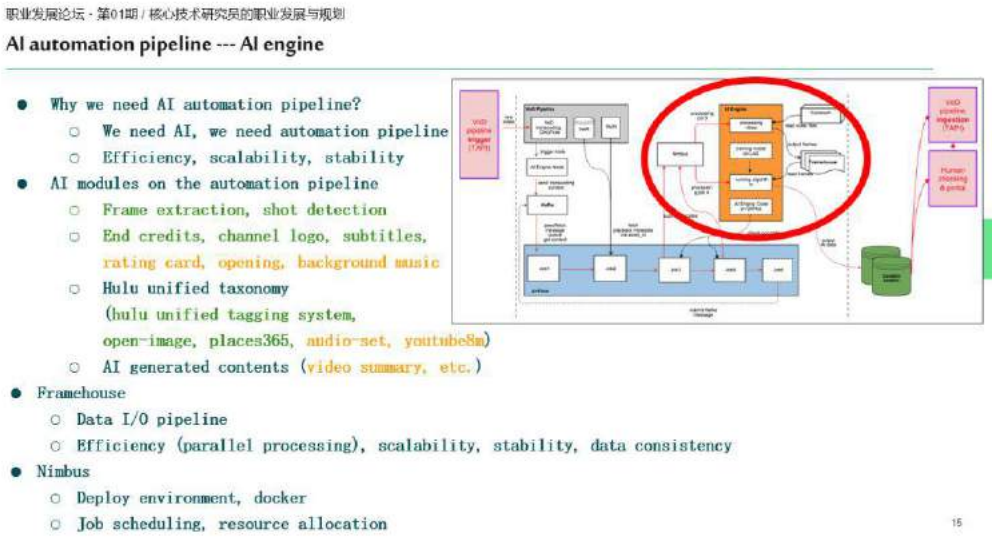


所以，这里我们会有一套算法逻辑：先拆分，然后检测有没有字幕，最后合起来之后得到一个 yes or no 的结果。

知道 yes or no 之后，我们还需要判断到底是什么语言，是拉丁语系，还是亚洲的中文，韩文，日文等等，会做语言种类的分类。

3. 自动化流水线

DataFunTalk



在做了上述工作之后，我们发现必须有一套流水线流程，因为 Hulu 每天更新的视频数量非常多，比如美剧每天就有 500 部更新，这不可能都重新跑一遍算法，最好是有一套自动化的流程来处理，并方便后期人工的复查，所以我们建立了一套 AI 的 pipeline。

我们还做了 Framehouse，因为我们要把每个视频拆分，做到 1s 或者 0.5s 拆出 1 帧，数量级非常大，做算法处理后，重新读帧，速度要尽量快，所以对 Framehouse 进行了很好的设计。

最后，我们还提供了一个 Deep Learning 分布式计算平台。

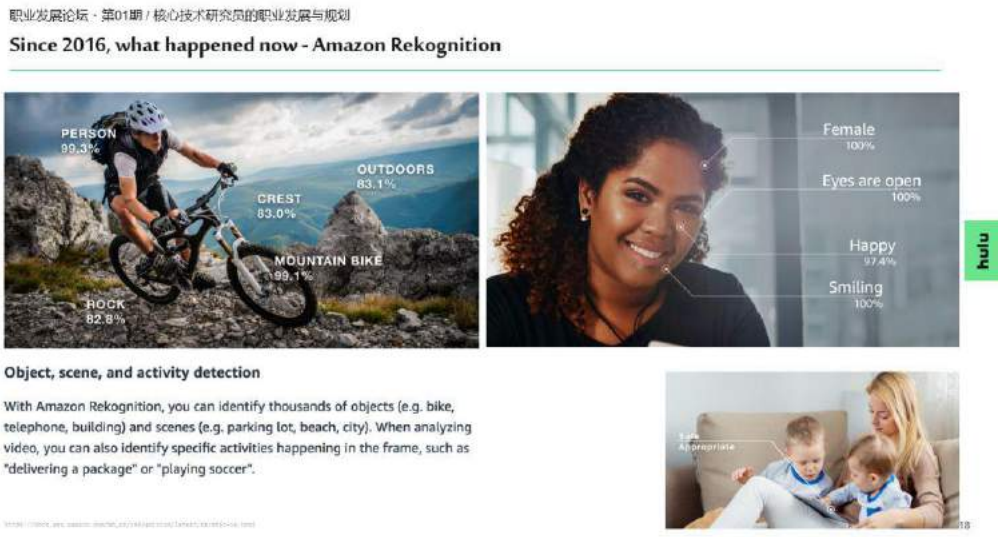
B. 视频标签

当对视频拆分之后，接下来的工作就是给视频加标签。比如在这个场景下，里面发生了什么事件，有什么元素，是在什么样的场景下发生的。

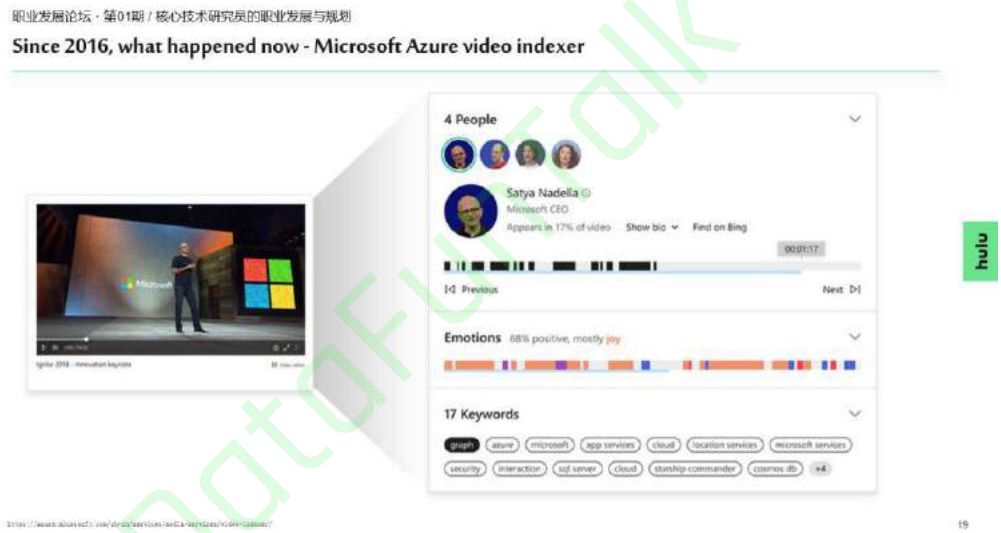
在 16 年的时候已经有非常多的标注数据，公开的 data set 包括 OpenImage ,Places365 ,Sports1M ,FCVID-LSVC ,YFCC100M ,MSR-VTT 等。

Hulu 是缺乏这些标注数据的，Hulu 的办法是一方面借助人工，启动自己的标注工程 ,另外一方面是借助一些学术圈相对开放的 data set 预先训练自己的算法模型。

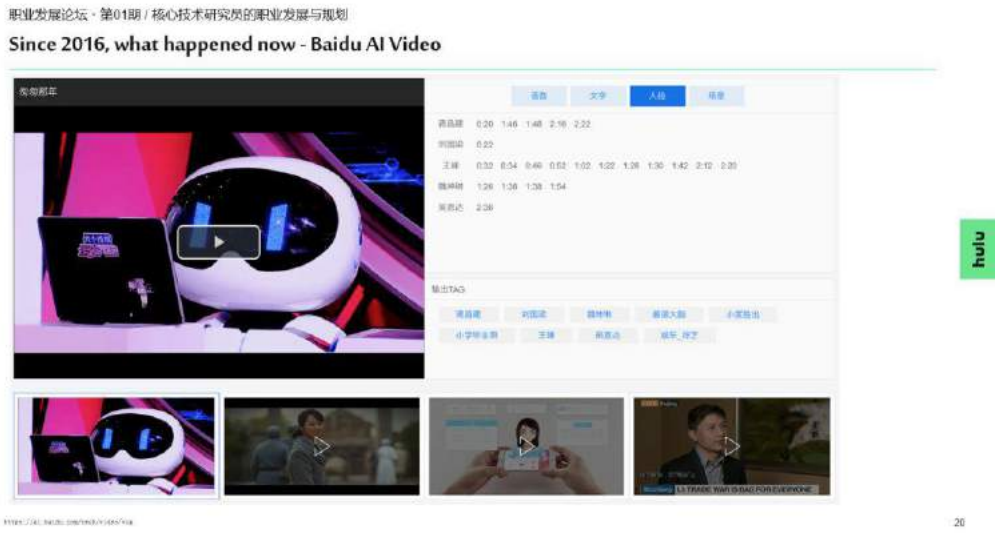
1. Since 2016，what happened now



① 亚马逊推出的图像识别 AI 系统：Amazon Rekognition，可以识别对象，人物，文本，场景和活动等。

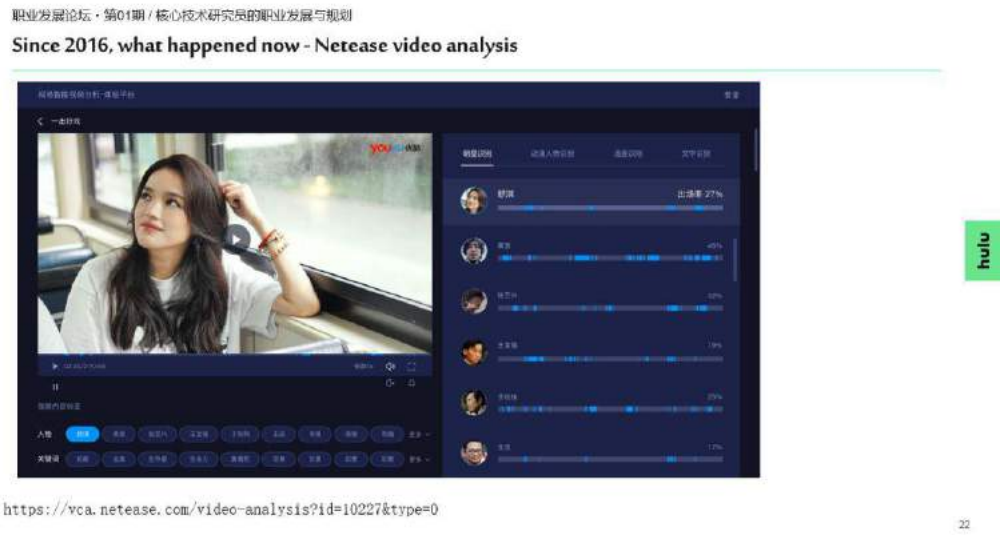


② Microsoft Azure 推出的 video indexer，不仅可以声音识别成文字，还可以对人物、关键词、情感等进行分析。



③ Baidu AI video，主要功能是人物识别和标签。

④ Alibaba AI in video，主要实现的功能除了人和标签外，还包括智能封面和视频摘要。



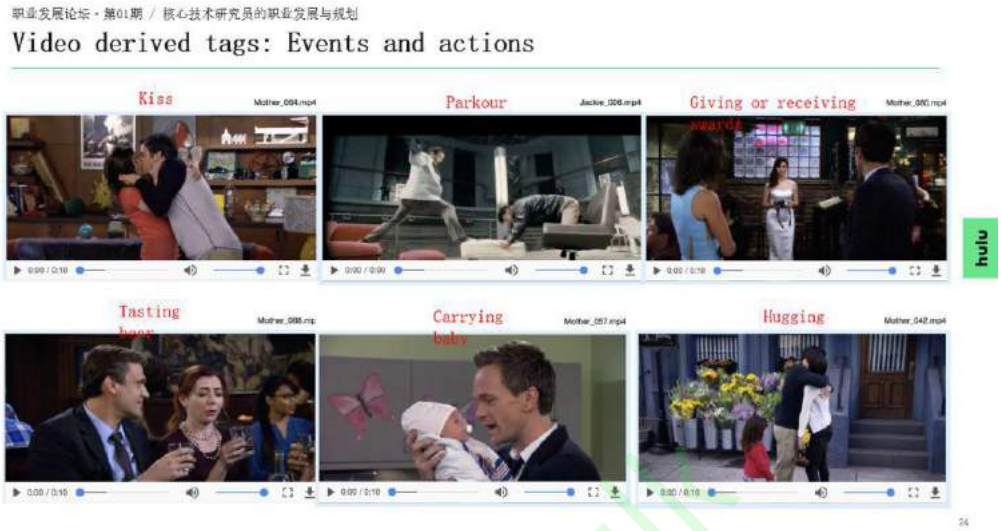
⑤ Netease video analysis，网易主要做的是明星动漫人物识别，语音文字识别等。

2. Hulu video derived tags

① Scenes and objects

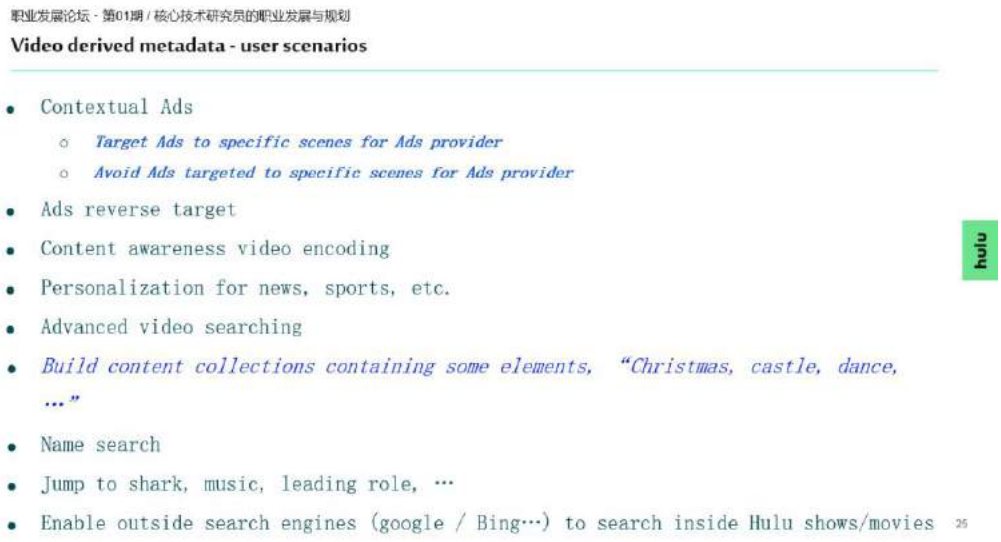
看了这么多，主流的公司都在做这件事情，大家的思路也都有很多相似的地方，这里重点介绍下 Hulu 是怎么做的，上图主要是 Hulu 的一些例子，是场景和物体的识别。

② Events and actions



这里是事件和行为标签。

3. 视频生成元数据：用户案例

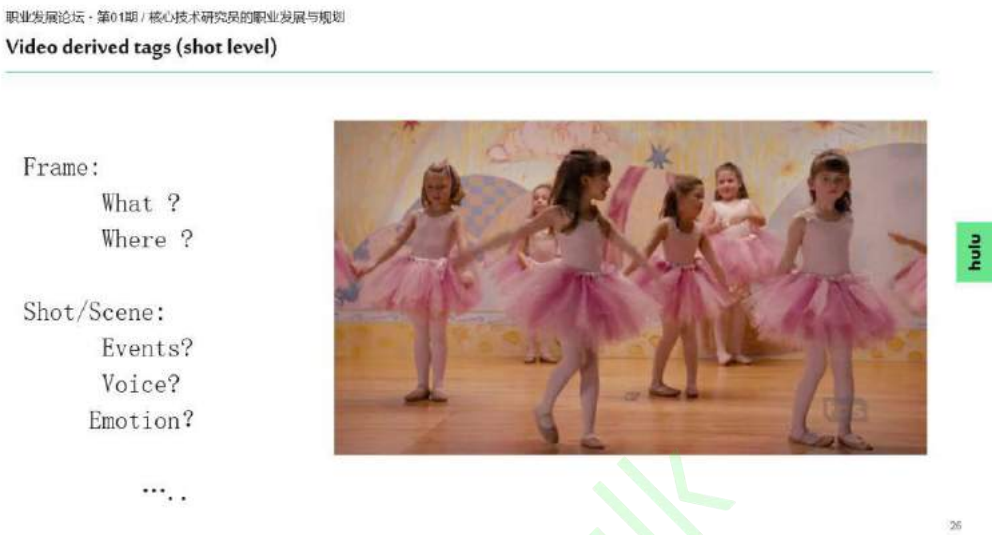


为什么这么多公司都在做内容标签，因为内容标签可以引入一些新的场景，比如：

- 上下文相关的广告，可以在暂停的时候插入广告，做文本相关性。
- 广告反向目标，因有些内容提供商是不允许在内容里插入广告的，比如 espn，所有和球员相关的广告都不能放，如果 Hulu 不知道过来的视频有没有球星代言的广告，很可能会造成侵权。
- 基于内容的视频编解码
- 基于内容的个性化的推荐

- 视频的搜索，比如检索美国大选新闻，出来的结果非常多，可能一段 30 分钟的视频，只有 2 分钟是相关的，用户不可能逐条去看，需要为用户标注出来需要的部分。

- 基于内容的分类，比如 90 后的最爱，城堡里的恐怖片，等等。



当开始做视频标签时，首先应从产品的角度出发，告诉产品视频里都有哪些内容，比如：

- 视频中有什么元素

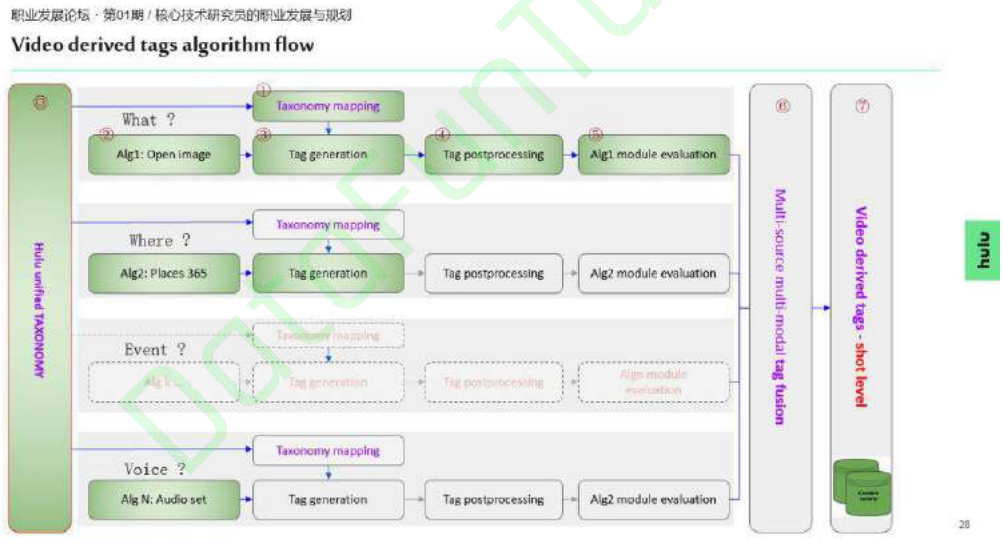
- 故事发生在什么场景下

- 发生了什么事儿

- 所表达的情感是什么

这是我们试图想回答的问题。

所以，第一阶段我们借助于很多 open data set 对算法模型进行训练，算法模型训练好之后，会在 Hulu 人工标的少量的数据上做 Fusion 融合。

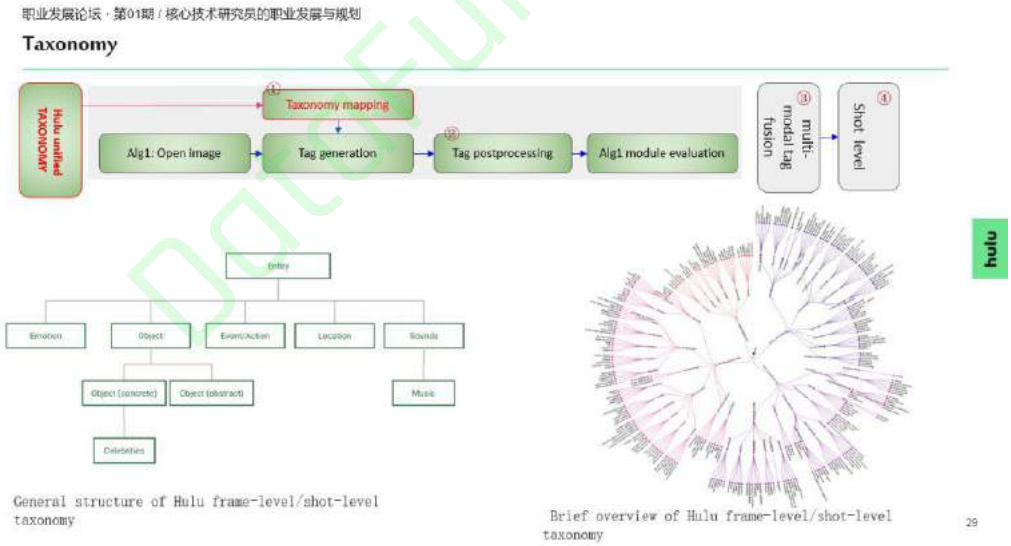


这里有一个更精细的算法流程来说明：

首先我们会定义一套自己的 Taxonomy 分类方法，这点非常重要，我们到底需要打哪些标签，这些标签以哪些层级来构建的，对于后面的算法会非常有影响。所以我们花了很大的力气人工的整理了一份 Taxonomy，构建了词和词之间的关系。

有了 Taxonomy 之后，当我们给定一个数据源，我们在上面有一套算法之后，会生成标签，但是这个标签不能在 Hulu 直接用，需要经过 tag postprocessing 标签后处理，因为这些标签映射到我们自己的 Taxonomy 时是有距离的，需要把多个标签 mapping 到一个 Hulu 想要的标签上，这就是标签后处理。

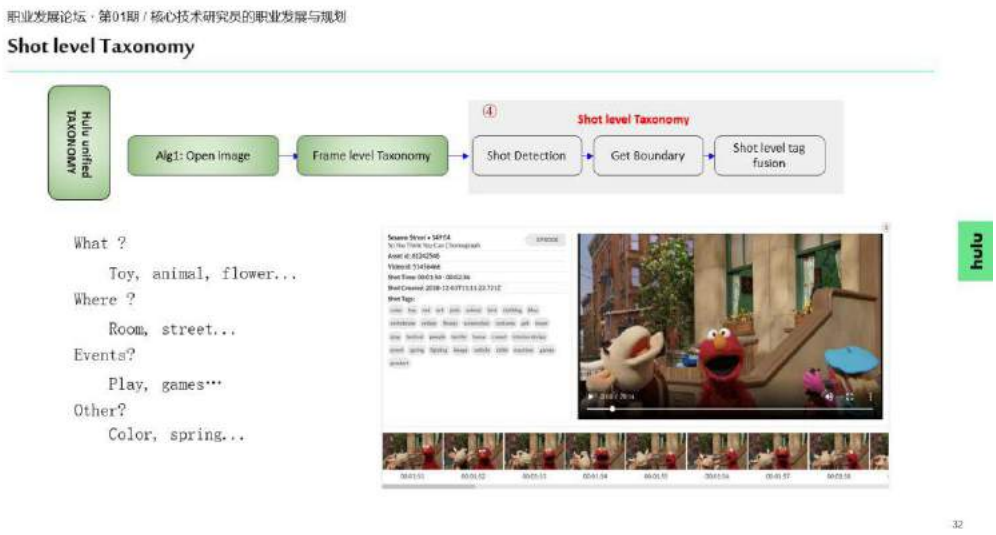
另外，需要做多个算法 module evaluation 模块质量评估，最终通过 Multi-source multi-modal tag fusion 融合，生成 Hulu 的某个镜头或者某个场景下想要的标签。



这里有更加详细的过程，这里主要是讲我们自己的 taxonomy，然后 taxonomy 如何划分的方式，右边是 taxonomy 示意的结构。

这里是讲标签的后处理，通过人工标注的标签来设定相应的阈值。

模型融合，设定阈值和分类准确度，对分数进行归一化融合。



这个是卡通片标签的例子，包括 what，where，event，Color 等。

C. 内容生成

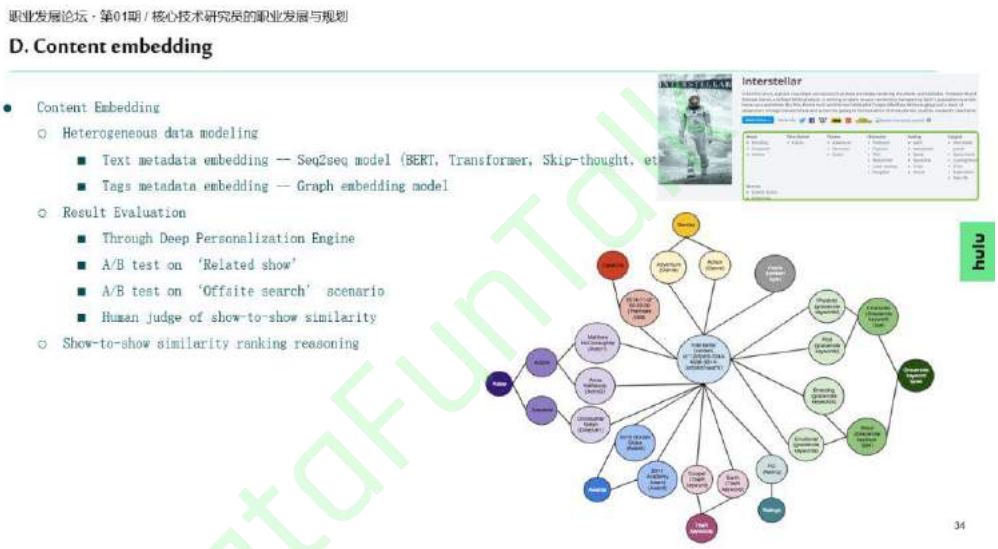
接下来的工作是内容生成，主要包括：

- Sports highlights and moments，精彩片段推荐。

- Video summary 视频摘要，合并出基于主角，剧情，精彩片段等的 summary，推荐给用户。

- Avatar，探索性的项目，比如有些酷炫的动作，进行骨骼动作模拟，之后再赋予一个虚拟的形象展示出来。

D. 内容向量



除此之外，结合 Hulu 已有的标签，我们叫事实标签，比如演员，导演，剧的介绍以及对整个故事情节的理解，可以把整个 video 合成一个 vector 向量。通过这个 vector 可以非常容易的去算两部剧的相似性，并且反推这两

部剧为什么相似，比如同一个导演，或者同样人物的出现，或者包含了某些非常类似的场景，所以 content vector 在视频推荐中占有非常重要的因素，并且可以带来不小的提升。

具体算法，对于 content embedding，文本使用 BERT，transformer 模型，标签使用 Graph Embedding 模型，结果评估使用 ABtest 在相关剧集推荐上做评估。

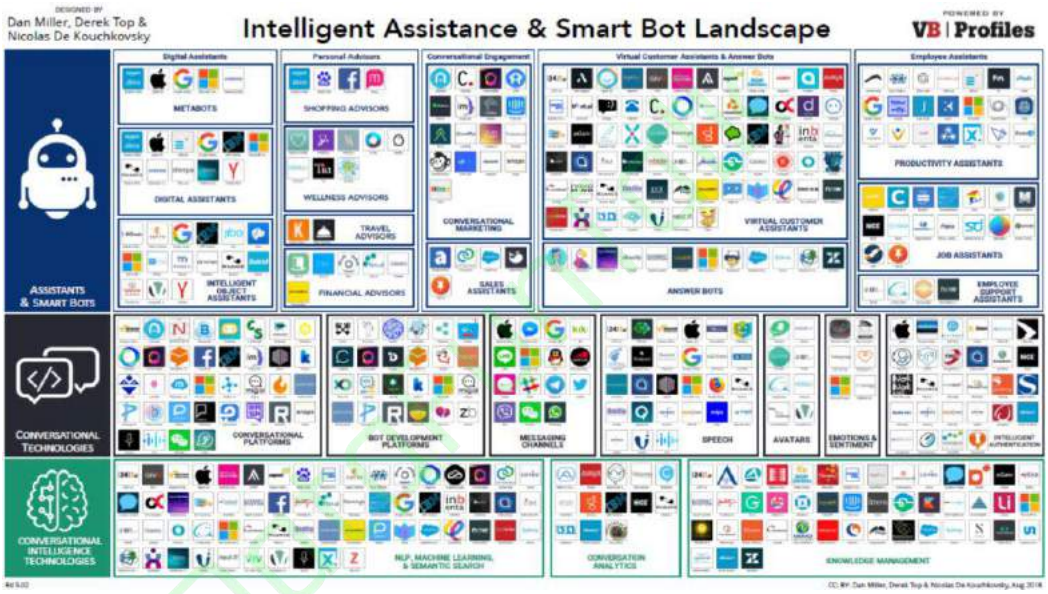
这是具体的生成标签的例子。Gracenote tags 是供应商的标签，Content derived concept 基于内容的标签是自己生成的，由于供应商或者第三方提供的标签有很多是不全的，通过我们基于内容生存的标签是可以弥补这些不足的。

对话智能：国际视角，国内形势及案例学习

嘉宾：黄惠燕　编辑：张满意

纵观全球，AI 战事风起云涌，懂语言者得天下- Conversational AI（人工智能对话技术）早已成为各大公司 AI 领域必争的技术高地，国际巨头 FAMGAs (Facebook, Apple, Microsoft, Google & Amazon) 一直大力投资发展此领域，中国的 BATXs（Baidu, Alibaba, Tencent & Xiaomi）的迅猛进步也惊为世人瞩目！这里我们将一起讨论人工智能对话技术的全球景观和中国市场的发展特色，共同学习以 Amazon Alexa，Nuance 和 Google 等为代表的创新型商业对话技术应用案例。

国际视角



上面我们援引了一张 VB profiles 的人工智能及智能机器人的整体布局图，从这个图中我们可以看出，对话智能整体上可以分成三个层次，最底层的是基础研发，这一层主要是研究自然语言处理，机器学习，语义分析等底层技术，中间层是对话技术，主要涉及对话平台，对话机器人开发平台，信息通道等，我们国内的微信和 QQ 就工作在这个层次，最顶层是各种助理和智能机器人的应用，在这一层，对话机器人和现实业务结合起来形成专属于业务的对话机器人，比如说导购机器人，销售机器人，员工助理，健康助理等等。

下面我们分别列举一下这三个层次及相关的知名公司：

基础研发

研究方向	代表公司
自然语言处理，机器学习，语义分析	Apple，Baidu，facebook
对话分析	IBM，NICE，Call Journey
知识管理	AnswerDash，Coveo，eGain

对话技术

DataFunTalk

研究方向	代表公司
对话平台	facebook，WeChat，Snaps
机器人开发平台	BotKit，Botstify，Zenbot
信息通道	WeChat，WhatsApp，Twitter，QQ，LINE
语音技术	Apple，Google, Amazon
在线客服	Botanic，Existor，FaceMe，Living Actor
情感语意分析	Boyond Verbai，Heartbeat ai，Microsoft Azure
智能认证	Apple，faceTec，VoiceVault

智能助理及智能机器人

研究方向	代表公司
引导机器人	亚马逊，谷歌，苹果，微软，三星
数字助理	亚马逊，苹果，微软，IBM，三星，谷歌
智能物体助理	谷歌，IBM，丰田
导购	百度，亚马逊，facebook，Mona
健康顾问	Babylon health,HealthJoy,HelloAva
旅行顾问	kayak slack，mezi
理财顾问	Abe AI，Cleo AI，Clinc

对话营销	Automat，Clustaar，Conversable
销售助理	Ample Market，Conversica，Salesforce
虚拟客户助理	xiaoi，Ada support，Agent IQ
问答机器人	Alterra.ai,AnswerIQ,DigitalGenius
生产率机器人	Boomerang，Cisco Webex，clara labs
工作机器人	Aizimov，Converse.AI , Equals 3
员工支持机器人	Ipsoft , Living Actor，ServiceAide

DataFunTalk

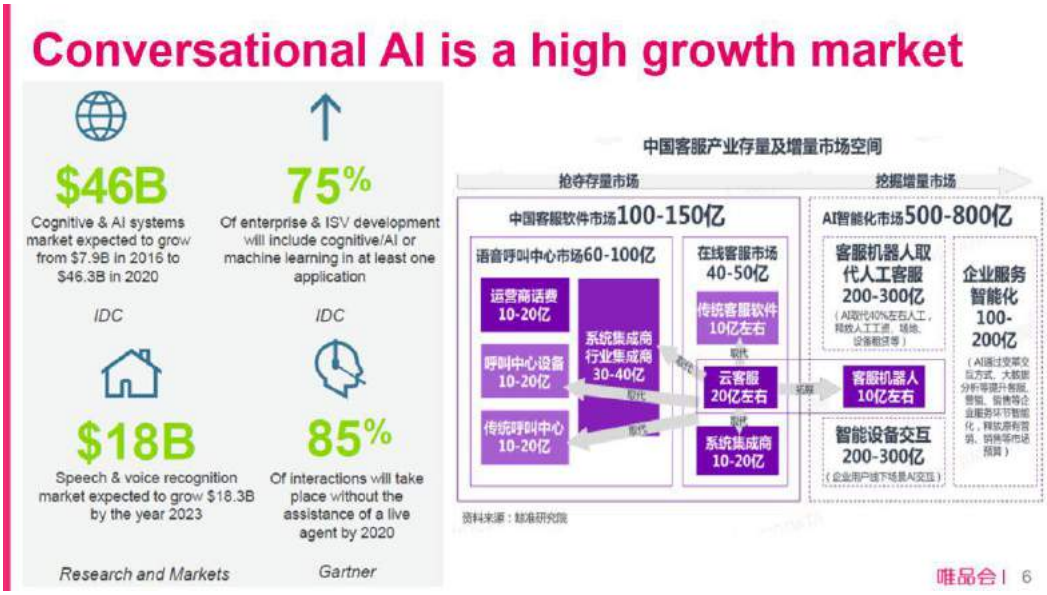
从上面的列表中我们可以看出，中国企业的上榜率还是很不错的，比如说在基础研发领域的百度，在对话技术领域的腾讯，以及在助理领域的 xiaoi 等，当然这些还不够，我们希望越来越多的中国企业能够纳入到这个图的各个版块之中。

智能对话市场是一个高速成长的市场

智能对话市场是一个具有广泛前景的市场，国际上权威的公司 IDC 预测到 2020 年认知及人工智能市场有望从 2016 年的 79 亿美元增长到 463 亿美元，75%的公司至少会在他们的一款应用程序中加入智能对话的技术，著名的咨询机构 Gartner 预测到 2020 年 85%的交互将不需要人工的干预，到 2023 年，语音识别市场有望增长 183 亿美元。

国内的鲸准研究院的研究也显示，随着云计算技术的普及，中国传统的客服软件市场将会受到极大的冲击，传统的硬件设备商，集成商，呼叫中心都会慢慢的被云化，取而代之的是云客服，云计算巨大的算力，大数据技术的应用与智能对话技术的融合衍生出来的客服机器人还会催生出新的市场，规模预计会达到 500-800 亿，远大于原来 100-150 亿的规模。

DataFunTalk



中国市场导读

智能对话领域有几大关键的卖点，分别是：

NLP：自然语言处理

STT： 声音转文本

TTS：文本转声音

国内的领军企业像腾讯，科大讯飞，百度在上面这些技术方面都有涉及，而阿里巴巴和唯品会在这个领域主要的卖点是 NLP 技术。

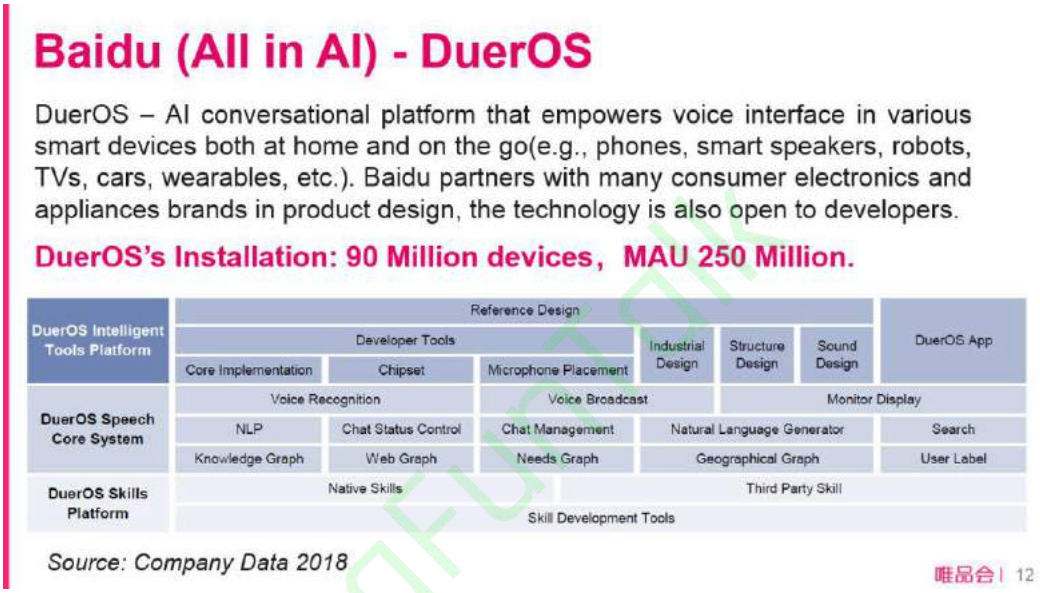
下面我们以 BAT 作为中国公司的代表分析一下国内对话智能的形势。

DataFunTalk

1. 百度 —— 中国最大的搜索服务商

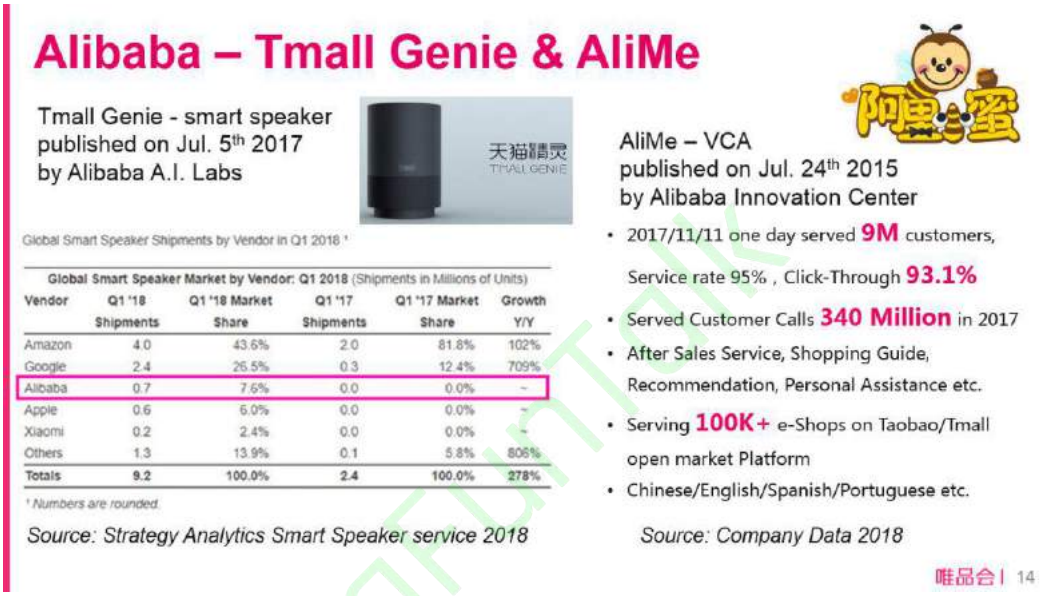
2018 年的数据显示，百度的研发人才占比达到了 43% ，市值 630 亿美元，百度重磅推出的 DuerOS 是一款对话式人工智能操作系统，装机量已经达到了 9000 万，日活达到了 2.5 亿，数量极其惊人。DuerOS 从架构上总共分 3 层，第一层是能力层，包含原生能力和第三方能力，原生能力指信息，服务，音乐，地图，聊天等。

第三方能力诸如购物，新闻，娱乐，IoT 等，第二层是核心层，主要功能有语音识别，语音合成，屏幕展示，自然语言处理及对话技术，知识图谱等，最上层是一个开放平台，提供了核心接入组件，开发套件和参考设计。



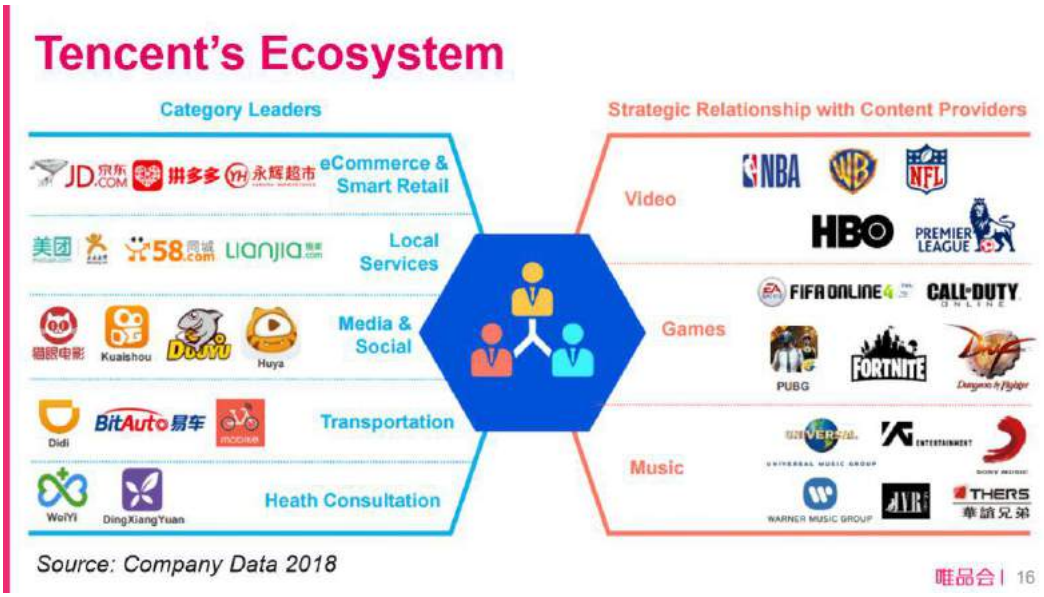
2. 阿里巴巴 —— 中国最大的电商平台

阿里巴巴是中国最大的电商平台，不仅如此，它还是亚太前三的云服务提供商，2018 年阿里巴巴的研发人员占比达到了 45%，略高于百度。AliMe 是阿里推出的一款智能机器人平台，定位是服务淘宝消费者的购物私人助理，提供极致的购物体验，阿里小蜜经受过 2017 年双十一的考验，一天服务了 900 万客户，服务率 95%，点击率 93.1%，2017 年全年服务了 3.4 亿次请求，服务范围涵盖售后服务，导购，推荐，个性化支持，阿里小蜜致力于服务淘宝上的 10 万+商户，会四国语言。2017 年阿里又推出了一款对话机器人，名为天猫精灵，可以和人进行语言交流，打通了阿里内部各大生态体系，不仅可以订外卖，购物，还可以听歌，轻松完成支付，和 IoT 结合之后可以语音控制家里的电器。



3. 腾讯 —— 中国最大社交平台

腾讯是中国最大的社交平台，2018 年度它的研发人员占比是 BAT 中最高的，达到了 51%，旗下的两款拳头产品 QQ 和微信分别是桌面和移动端的龙头 app，日活破 4 亿。腾讯的优势在于通过一系列的投资行为整合了各个类目的巨头，形成了庞大的生态，智能对话技术的升级可以很快惠及到整个生态系统。



经典案例学习

1. 亚马逊 Alexa: ocado 的创新

DataFunTalk

2017 年度，Ocado 是英国最大的只做线上的零售商店，全年完成了 13 亿英镑的销售额，每周产生 26 万个订单，平均每单 50 个商品，每周卖出 1300 万个商品，在库的商品有 4.9 万个，并且能达到 99%的准确性，支持这些庞大的数字背后涉及到 Ocado 很多的技术，我们简单列举一下：

– 对话技术

– 机器人技术

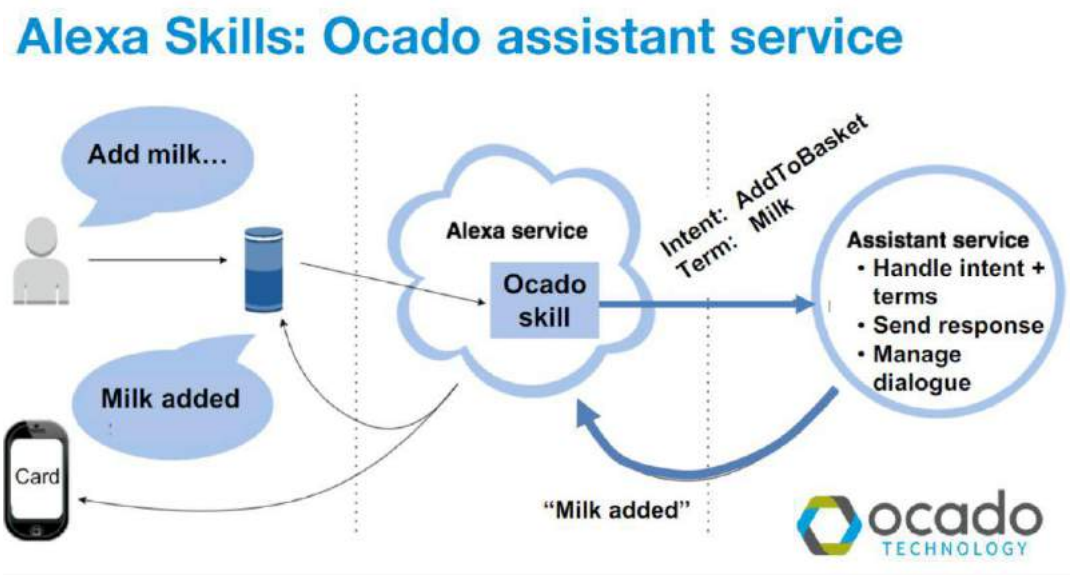
– 仓库技术及优化

– 数据分析，人工智能及机器学习

– 客服及最后一公里

DataFunTalk

下面是客户下单的案例：



Alexa 面临的挑战及解决方案：

识别错误，从键盘输入转向声音输入，这个过程中容易产生识别问题，比如说错把 cheese 听成了 chilies。

要解决这样的问题，我们需要用户个性化的数据来进行训练和学习，或者增加电话回访流程确认清楚，或者进行一些概率上的统计来帮助决策。

常识推理，比如说下面这句话：

add ten fish fingers , three bunches of bananas and some dishwasher tablets but not the lemon ones.

计算机会疑惑 :是 10 包鱼条还是有 10 个鱼条的一包呢？到底要多少根香蕉？哪些洗洁精不是柠檬味的呢？

要解决这样的问题，我们需要更加聪明的机器智能。

客户反馈

客户使用 Alexa 的之后，得到了积极的反馈，因为语音交互带来的便利，用户在做饭的时候就可以下单，体验非常棒，还有就是 Alexa 通过数据分析能够辅助用户下单，避免用户从 Ocado App 庞大的商品库中选商品，当你要下单的时候，你会发现你要买的东西已经在购物车啦。

2. Nuance ： Esurance

Nuance 公司是最大的专门从事语音识别软件、图像处理软件及输入法软件研发、销售的公司，它可以提供全场景的解决方案，定位方面和中国的科大讯飞类似。Nuance 的技术被广泛的使用到通信，金融，医疗，旅行，政务，零售等领域，并取得了骄人的成绩。

Esurance 是一家金融评级为 A+ 的拥有多个险种的保险公司，是美国最老牌的个人保险公司，它在全国范围内提供车险服务，截至目前，它已经拥有 17 个办公室，在保车辆达到了 520 万之多。

Esurance 的挑战：

– 创建一种沉浸式的用户体验

– 帮助客户更快的自助的找到答案

– 在合适的时间和合适的客户建立联系

– 利用对话智能和分析技术提升用户的整体体验

Esurance 的目标：

综合提升销售业绩，KPI，用户体验及网络智能。

解决方案

DataFunTalk

– 使用 Nuance 的对话技术，将人工客服和机器人客服结合起来，优先机器客服，如处理不了则转人工客服，切换快速，用户无感知。

– 增加协同浏览的工具来帮助客户解决问题，培养用户自助解决问题的习惯。

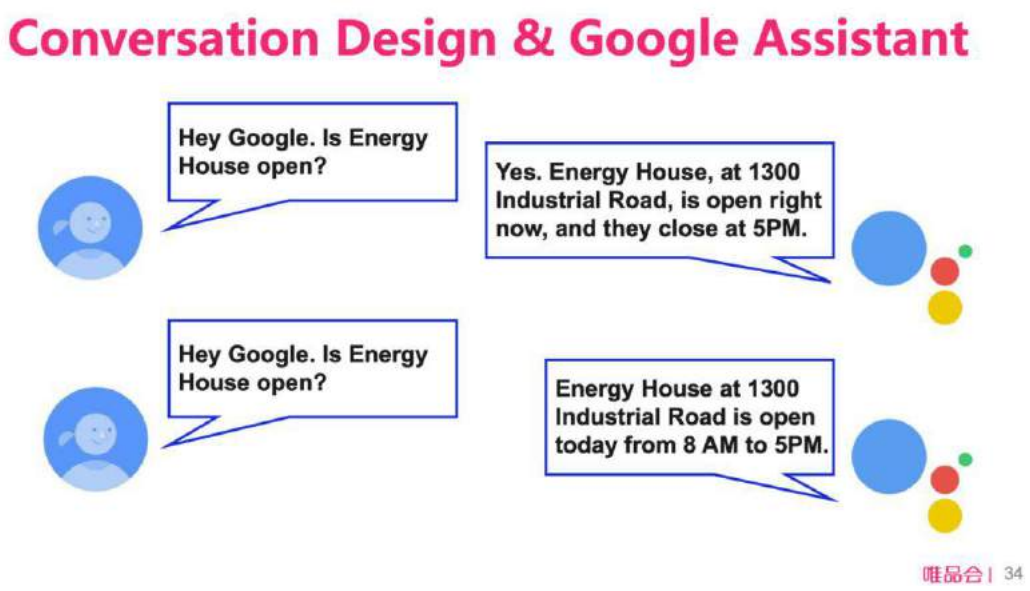
– 使目标和商业规则保持一致，最大化提供解决问题的效率。

– 洞察用户的交互行为，不断提升用户体验。

最终效果：

3. Google 助手：对话设计

对话一：



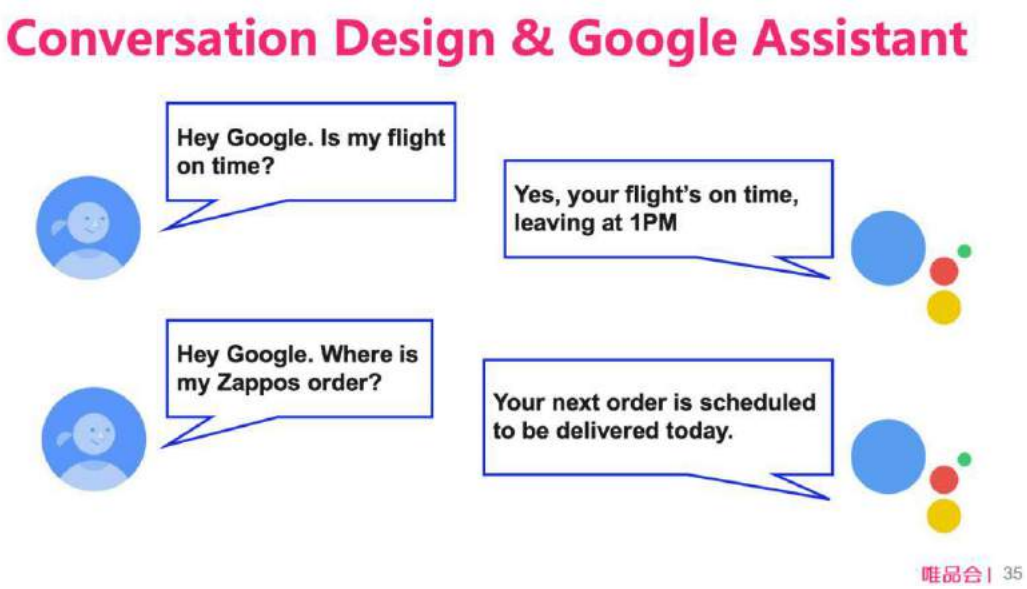
亮点：

1. 给出超出预期的答案，您只问这家店是否开着，我的答案包含了更多的信息，不仅回答是否开着这个问题，还告诉您它的开业时间，它的地址在哪里，这样减少了对话的轮数，因为这些问题可能您后面也会关心。

2. 针对同一个问题，以不同的方式回答出来，对话过程非常自然。

对话二：

DataFunTalk

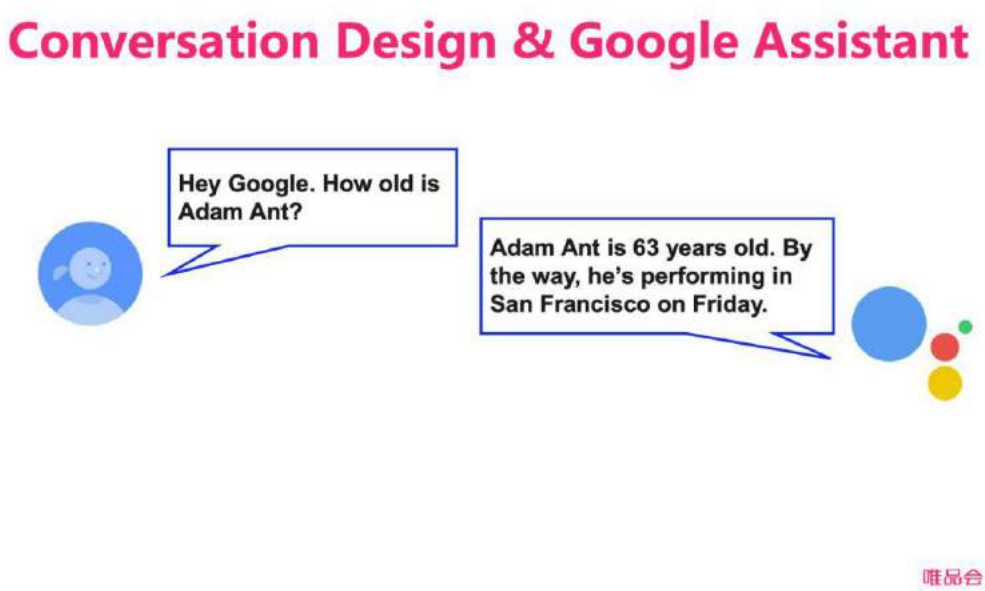


亮点：

意图识别准确，当问您的订单到哪儿了时，您可能并不关心它具体在哪儿，您可能关心的是它什么到您手上。

对话三：

DataFunTalk

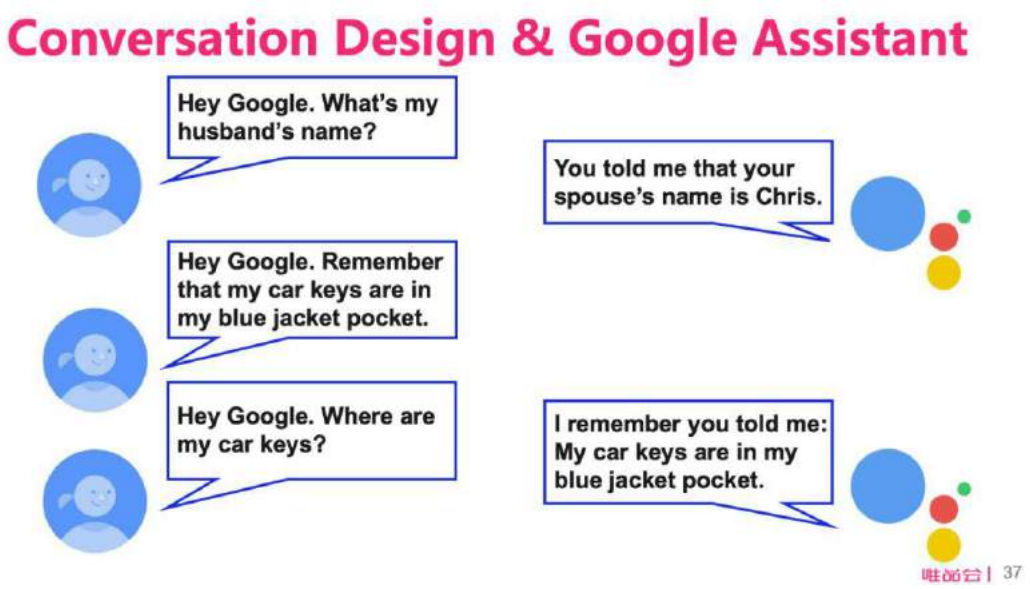


亮点：

因时因地提供 amazing 的答案，这个对话是发生在 San Francisco 的，google 助手觉得您可能是 Adam Ant 的粉丝，不仅告诉了您他的年龄还告诉您他周五在 San Francisco 有演唱会，感兴趣的可以去听一下。

对话四：

DataFunTalk

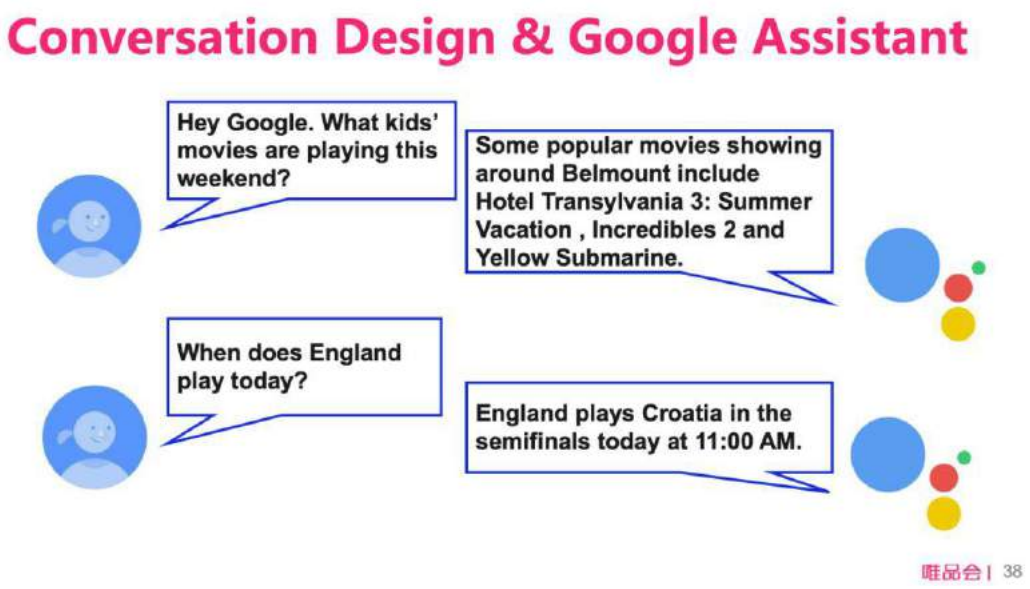


亮点：

隐私保护好，来源追溯清楚。当您问 google 助手一些比较隐私的问题时，它会准确的告诉您：您告诉我或者我记得您对我说过等类似的话，清楚的告诉您它答案的来源。

对话五：

DataFunTalk



亮点：

没有上下文语境也能很准确的给出答案，上面的问题没有语境其实挺难回答的，但是 google 助手也能回答得挺好。

对话设计的要点：

- 对话设计工作应该最先开展
- 问问题的方式非常重要
- 如果可以提供多一些的信息，但是也不能太过
- 如果条件允许，可以使用一些个性化的信息，比如说用户的地理位置等

DataFunTalk

- 让机器人有记忆

文章小结

本文专注于智能对话领域，分享了智能对话领域的国际形势以及国外权威机构对智能对话技术划分的三个层次，并以 BAT 为中国企业的代表粗略讲解了国内的发展情况及行业前景，最后我们分享了智能对话在零售，保险，个人助理领域的三个应用案例，剖析了他们的商业挑战及应用智能对话之后的效果，希望大家通过以上的讲解对智能对话能有一个初步的认识，并对它所面临的挑战和机遇能有初步的了解，谢谢大家。

DataFunTalk

云小蜜：在中国移动的落地实践

嘉宾：罗洪刚　编辑：刘汝洲

大家好，今天将从研发架构的角度对云小蜜自然语言理解技术落地实践进行分享，主要包括以下几个部分：

1. 云小蜜简介：智能客服领域内人机协同解决方案。

2. 移动的问题与解决方案：针对降本增效的打法。

3. 技术架构方案：项目背后的技术支撑。

4. 业务效果总结：典型业务场景的效果展示。

1. 云小蜜简介

阿里小蜜最早诞生的背景是淘宝平台存在着大量的业务需求，通过智能客服的技术手段可以有效提高业务处理效率，并可以将其赋能于阿里集团内部其他部门，阿里生态企业，以及集团外客户的不同业务场景中。辅助或者替代真人客服服务。

客服典型场景主要解决两类问题：问题咨询和交易纠纷。

经典客服模式（真人服务模式）：

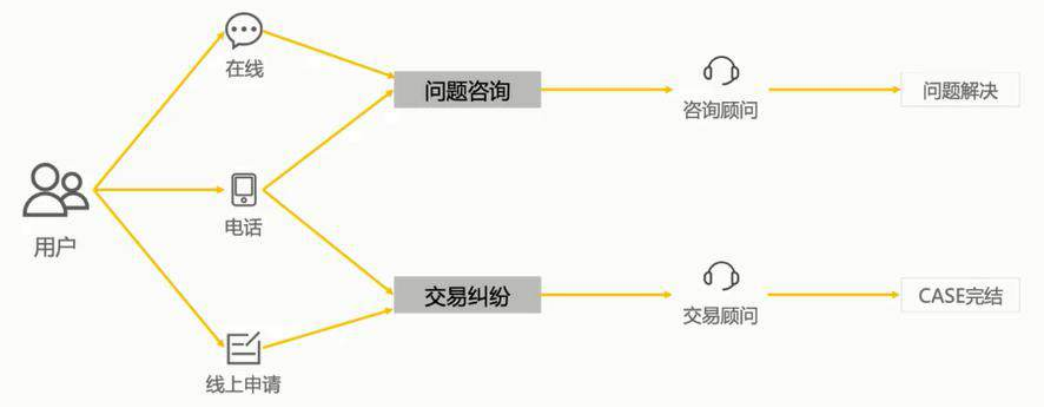


图 1. 传统客服真人服务模式

① 问题咨询：主要指用户咨询问题的业务，比如对产品，服务，资讯等内容请求。

② 交易纠纷：主要是在交易平台相关的业务中，比如在用户交易纠纷申请提交后，提供工单，由交易顾问确认责任等流程，处理结果判决等业务。

智能客服是通过对话机器人辅助或代替人工在传统客服中的功能，它的业务模式与传统业务模式类似。

智能客服模式：

DataFunTalk

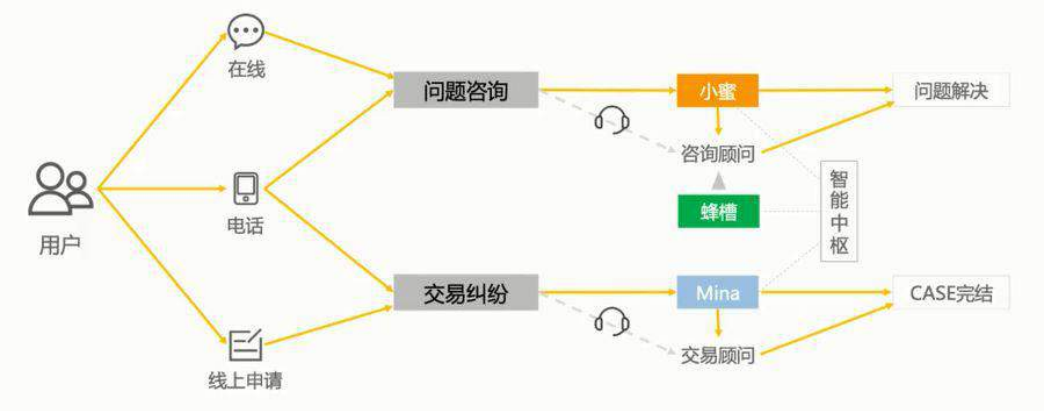


图 2. 智能客服服务模式

智能客服的业务目标跟人工客服类似，它的目标是在以上业务流程中，通过机器人小蜜自动识别并处理大部分请求（一般 90%以上），只把机器判断不准或无法解决的问题交给客服人员处理，从而达到降本增效的目的。

1.1 小蜜家族

阿里小蜜，最早是服务于淘宝电商平台的，提供智能对话辅助能力，发展到现在已经具备的能力包括：智能对话辅助，智能决策，智能服务路由，智能预警，智能质检等。

在阿里统一的人工智能技术中台支撑下，阿里小蜜家族成员包括：超级小蜜，行业小蜜，店小蜜，企业小蜜-云小蜜。

超级小蜜：主要是服务于在淘宝，天猫平台上做的咨询类，交易纠纷类问题智能业务处理。

行业小蜜：主要帮助赋能阿里生态圈的几十家企业的实现智能客服能力。

店小蜜：主要是针对在阿里电商平台上的商家，服务其电商用户，比如售前咨询，售后导购，物流处理等，这块业务量占比相对较大，

企业小蜜-云小蜜：主要针对的集团外的业务，通过与阿里云合作，提供整套的客服解决方案，通过阿里云输出给 ToB 企业，或者为在钉钉上几百万的企业的用户提供小蜜智能服务能力。另外，也为阿里收购的海外电商平台 Lazada 提供智能客服能力。

云小蜜不只是对话，而是一套针对客服场景下的完整人工智能解决方案，对话是解决方案的核心和最重要的一环，但不仅限于对话，还包括智能决策，智能调度，智能预测，智能分派等产品矩阵。

统一中台支撑：小蜜家族虽然对外提供不同的小蜜能力，但是在中台框架下，底层的能力是相通的，中台提供稳定可靠的积木式能力，业务线通过对积木的有效组合提供完整的行业解决方案。

两层的中台：

a. 算法层：分词，实体，MRC，分类等及知识库。

b. 能力层：或者叫产品层，包括：FAQ 问答，KBQA，对话工厂等。

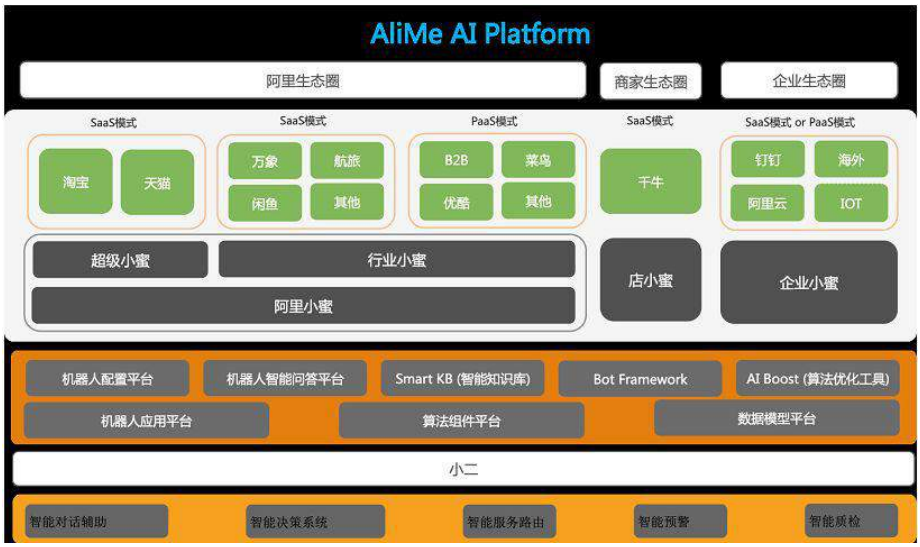


图 3. AliMe AI 平台

1.2 产品矩阵

云小蜜打造提供一套丰富的智能客服技术解决方案，典型的技术产品架构如下：

图 4. 云小蜜产品矩阵

其中，主要的产品功能模块：

① 对话智能路由（策略中心）：对用户请求，通过用户画像匹配操作，决定分配给人工或者机器人，并根据用户画像决定提供服务的类别和深度。

② 智能客服：主要提供智能对话能力。

③ 知识云：知识问答相关的知识库，为知识问答提供相关的辅助，对于知识云能够解决的问答任务，可以简单快速的返回答案。另外智能助手解决问题的答案也可以回流到知识云的当中去，丰富知识库。

④ 预测排班：预测服务量在未来一段时间（按周）的走势，合理排班坐席数量，解决传统人工排班的劳动量大，排班任务分配不准确等问题。

⑤ 智能质检模块：分为实时质检和离线质检两种场景，目的是监测服务人员的对话内容，情绪等，辅助客服任务的质量控制。

⑥ 分析报表：对业务处理结果以报表的形式提供展示。

⑦ 监控大屏：对业务处理过程，态势实时展现。

移动的问题与方案

2.1 合作与赋能

云小蜜和中移在线，客服业务共同点：一是，在某一时间段业务量会急剧增大；二是，公司规模急剧增长，迫切需要人工智能技术解决业务瓶颈。

因此，云小蜜和中移在线展开了一系列合作：

图 5. 云小蜜与移动合作发展路径

通过阿里的技术，云小蜜逐步赋能中移动的客服业务，在这个过程中，智能客服的流量切入比重是分步骤逐步迭代增加的。将传统的 FAQ 的服务逐步以智能机器人的方式扩展其性能，不断提升客服的智能化水平。

2.2 落地需要考虑的问题

① 降本：中国移动是全球客户服务领域体量最大的企业之一，2014 年成立的客服公司中移在线，中移在线用 4 万多客服人员服务 9 亿多用户，人力成本是其最主要的成本。由于移动月末出账单特点，移动话务量的周期波动非常明显，人力成本居高不下。

DataFunTalk

② 增收：中移在线作为有社会担当的企业，不能无限降低自有客服人员，需要释放出来客户人员带来新的收入，需要做到从成本中心到利润中心的转型。

③ 服务体验提升：移动旧有 IM 渠道基本无业务办理能力，只能通过简单的文本+链接跳转实现最基本的办理能力，交互复杂，用户体验不够连贯。

④ 热线渠道：话务高峰期，话务人员不够，排队时间长，用户体验待提升。

2.3 方案落地主要的挑战

① 各省技术设施不一致，性能表现不一，系统接入难度高。

② 数据能力，实时数据处理能力，记录数据不够细致等。

③ 合作的团队众多，协同的成本很高。

2.4 产品解决方案

DataFunTalk

图 6. 云小蜜在移动的落地方案

云小蜜为客户提供智能机器人的赋能能力，可以兼容到客户的客服业务系统中。其中在机器人模块，PaaS 层提供 QA 和多轮对话能力，以及知识图谱问答能力。云小蜜提供训练闭环、服务闭环、数据闭环三大闭环能力，从而赋能客户具备完全独立的自主运营能力。

2.5 任务型多轮对话（对话工厂）

对话工厂是帮助客户实现多轮对话落地的快捷方式，允许用户快速建立多轮对话能力。其基本形态如下：

DataFunTalk

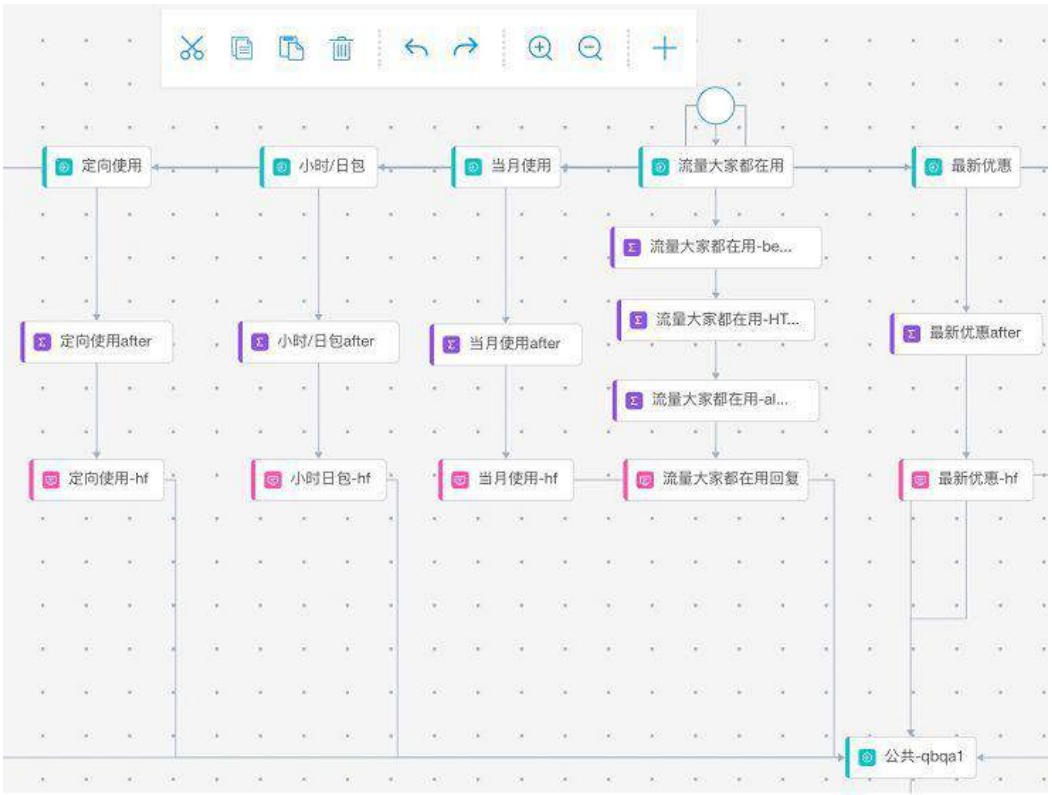


图 7. 对话工厂的产品形态

对话工厂能够实现业务自主运营赋能，图形化拖拽开发能力，无代码或少量代码的业务赋能，支持业务可以自主进行业务定制和开发，无需原厂商端到端持续支持。

实现的思路是，将对话过程分成三步：对话输入，对话处理，对话输出，再进一步抽象成图的触发节点，函数节点和回复节点。通过这三大类节点组合形成复杂的对话业务流程。其中每一种节点的作用是：

① 触发节点：意图识别结果，关联触发节点。

② 函数节点：具体执行任务的执行函数或者调用系统的接口。

③ 回复节点：把调用的结果做后续处理，反馈用户结果。

2.6 图谱+多轮对话

知识图谱可以提供以下能力：

① 提供复杂实体识别能力，将传统扁平化实体识别的任务转向层次化识别的手段。

② 基于图谱的问答能力，知识图谱的可以提供结构化信息的问答能力。

③ 基于图谱的推理能力，利用知识图谱，结合多轮对话的推理能力，可以完成在问答、导购、业务办理等场景任务。

④ 服务体验的提升，通过有机结合知识图谱，任务型多轮对话，卡片交互等能力，给在线渠道带来全新交互模式和服务体验。

在以上场景中，利用知识图谱的推理能力，可以将用户引导至业务办理的节点，结合对话工厂对话任务办理的模板，可以完成一个业务办理的任务闭环，有效提升业务办理成功率。

2.7 策略中心

智能策略中心，作为人工和机器人之间的泄洪闸，根据用户画像，工作台运行情况，用户对话等数据，综合运用推荐技术和自然语言理解等技术，实时识别高价值话务，在话务低谷期增加高价值人工流量进行营销，在话务高峰期控制人工服务流量进行业务削峰，从而实现用户体验和收益之间的动态平衡。

图 8. 策略中心界面

技术架构方案

3.1 移动架构

移动业务场景的技术架构如图：

DataFunTalk

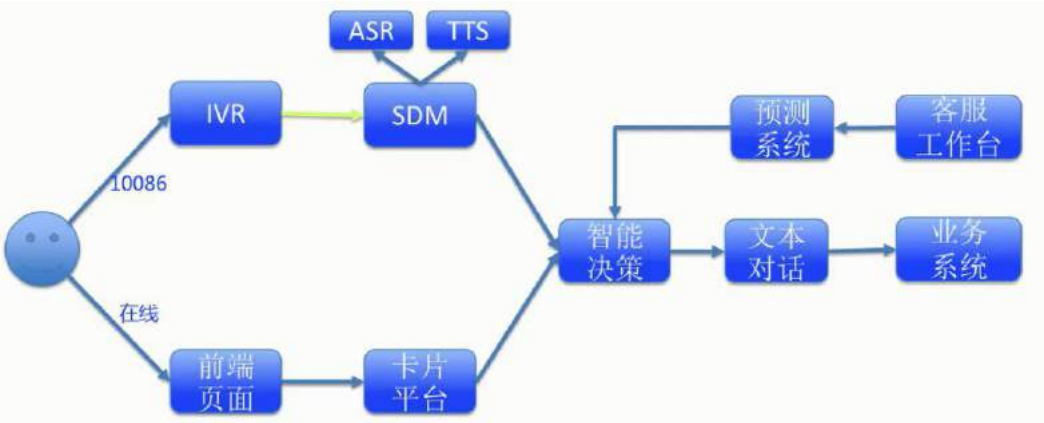


图 9. 移动业务架构

其中，主要的架构组件包括：

① IVR：典型的，经过 IVR 按键选择入口，经过 SDM 语音功能模块（阿里云语音对话管理系统），经过智能决策（人机智能路由）。

② 在线接入：用户通过 APP 或者 H5 页面接入，卡片平台可以消除部门交互的歧义可能，提升交互准确率，选择性触发卡片选项也可以降低工作难度，提高便捷性。

③ 策略中心：智能任务分流，决策系统。

④ 预测系统：预测话务量，根据话务负载决定人机分流的比重。

⑤ 文本对话：识别用户的意图和业务流程定制，完成移动的业务系统调用，并把业务结果返回给用户 。

⑥ 业务系统：移动各省公司的业务系统。

3.2 小蜜架构

为了达到技术能力的高效复用，灵活的部署，云小蜜的技术架构设计遵循：

① 分层架构

通过分层的架构把复杂问题逐步分解，让不同的工种专注于自己的领域和专长。自下而上包括部署环境，中台能力，产品与解决方案等层。部署环境主要指云小蜜根据用户需求的不同，运行在不同软硬件环境之中（公有云、私有云、专有云等）。为了屏蔽运行环境的差异，通过统一的中间件抽象层，很好地实现上层能力对环境的感知，从而实现平台能力复用和运行环境的灵活切换。中台部分，是由一系列智能化组件组成，可以很好地实现核心能力在小蜜各个产品之间的平缓复用，从而很好地把阿里的服务能力完整地赋能给到客户。在中台能力层以上的产品层，主要是针对不同业务线特点打造更加贴合于具体需求和场景的产品矩阵。

② 中台能力

通过中台能力降低开发成本，保证核心的算法和能力能够在各个产品中顺利复用。算法上，业务只需要关注自己业务上数据，模型沉淀即可。

DataFunTalk

图 10. 云小蜜的中台能力架构

在部署环境的适应性上，也综合考虑了支持集团，阿里云，企业私有云等不同用户群体业务部署的需求。运行于公有云或集团内时，主要的中间件有 Tair、HSF、MetaQ 等。当需要进行私有云部署的时候，直接使用 Redis，Dubbo， RocketMQ 等开源中间件打包替换即可。核心的思路是通一个抽象基础设施层，有效屏蔽上层能力对环境的依赖，从而真挚实现一套代码，多环境运行能力。

在基础设施之上，提供了统一能力框架层，主要提供账号、权限、接入、流控、插件等框架能力，不仅让小蜜产品体系具备一个完全可靠的架构基线，同时也能让算法、应用研发同学只关注于算法和业务本身，精力更加聚焦 。

3.3 中间件框架

① 核心目标

剥离业务代码对集团中间件的强绑定，保证算法，中台，业务代码等能够在阿里集团内部，阿里云，集团外的各种环境中无缝复用，真正做到‘一次编写，到处运行’。

② 核心思路

依赖倒置原则，通过增加接口层，让业务代码依赖于中间件接口，而不是依赖于具体的中间件实现。增加一个间接层，解决中间件不一致的情况。

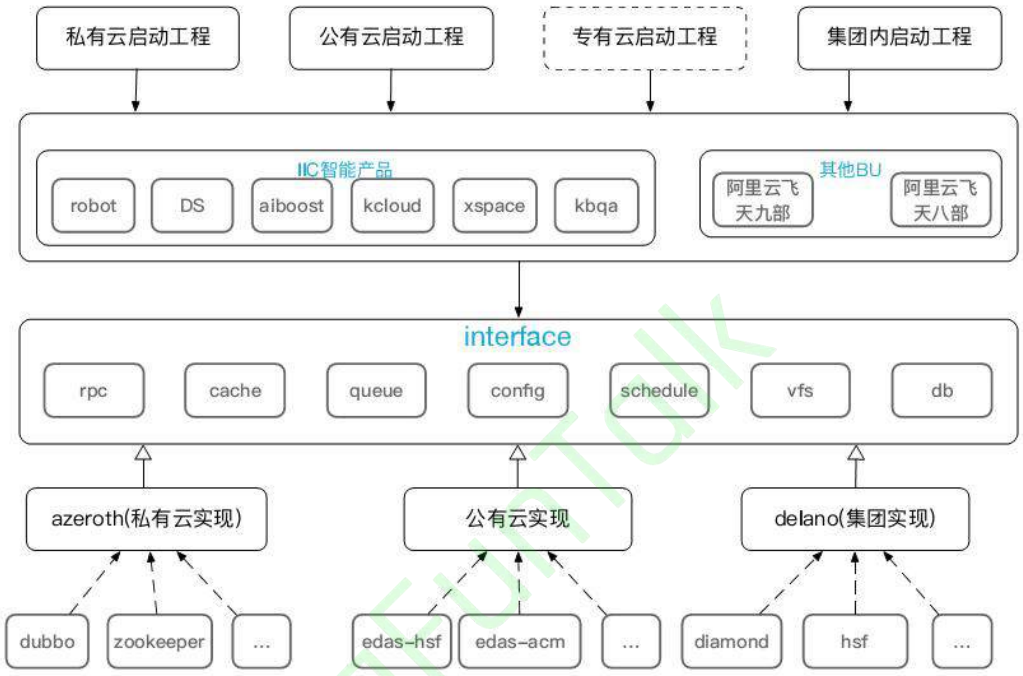


图 11. 云小蜜中间件框架设计

业务效果落地总结

4.1 智能导航全语音门户

通过智能导航服务占比曲线可以看到智能客服比重在不断增大，现在云小蜜在移动两省的日对话轮次达 3、4 百万，月对话轮次在 6、7 千万。

通过全语音门户导航系统代替传统 IVR，完成两省全量客户覆盖。

移动在线分公司可以基于我们的产品自主研发和配置业务流程，并实时上线。

业务波动系数，作为降成本的一个重要指标，在云小蜜的支持下，从近乎全网最低到全网最好，成本大幅降低。

DataFunTalk

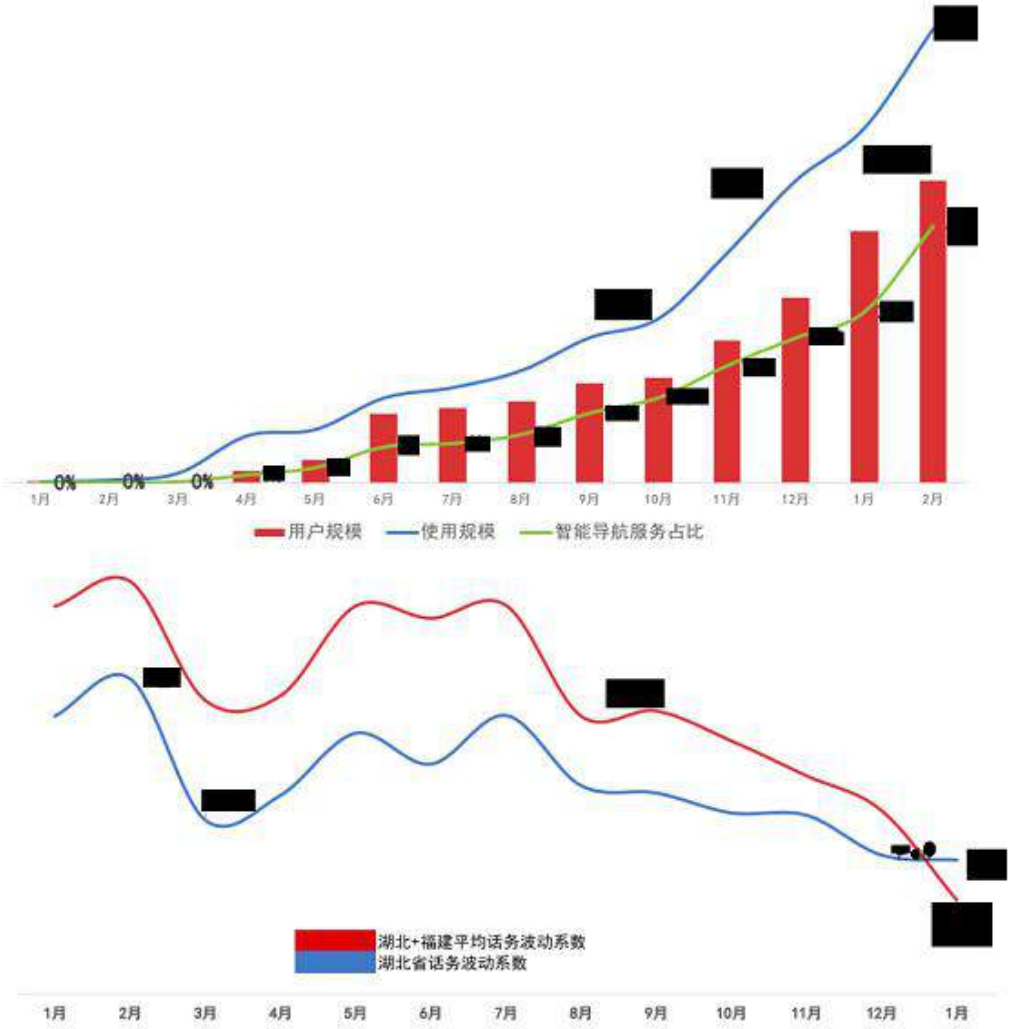


图 12. 云小蜜赋能移动业务智能导航业务

4.2 在线 IM 智能服务渠道

知识图谱+对话工厂实现 IM 渠道从简单问答到咨询，业务查询，业务办理的跨越式能力升级。

提供自动的商品导购能力，将导购转化率提升 1 个数量级。

用户评价解决率超过均值 20 个百分点，算法准确率 94%以上。

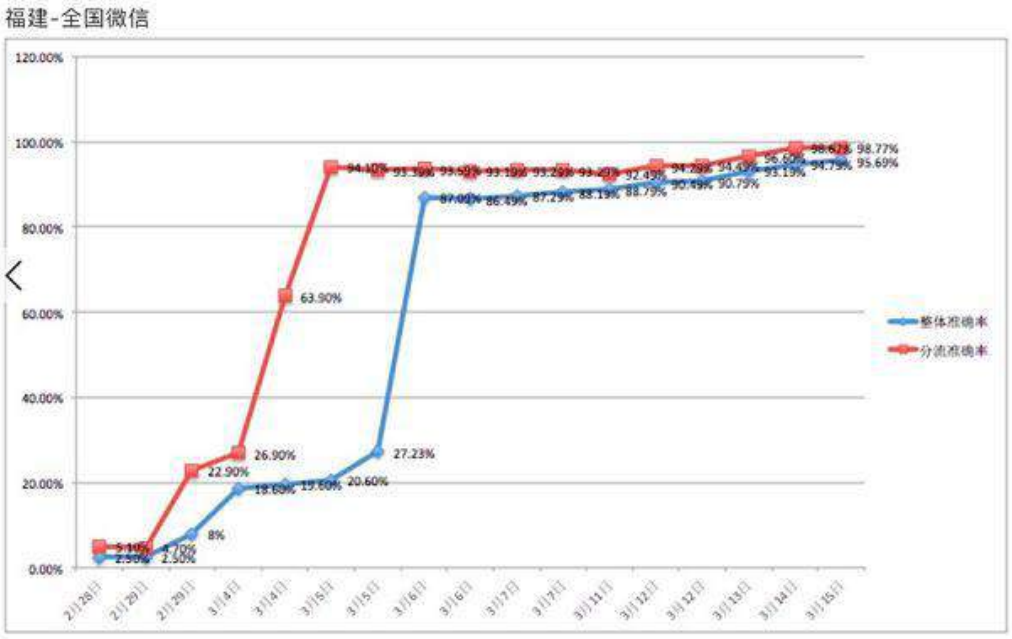


图 13. 云小蜜赋能 IM 智能解决能力

4.3 营销能力

- 在线营销转化率的数量级增长。
- 热线渠道营销收入接近成倍的增长。

这就是今天分享的全部内容，谢谢大家。

DataFunTalk

阿里小蜜：知识结构化推动智能客服升级

邱立坤　编辑：Hoh Xil

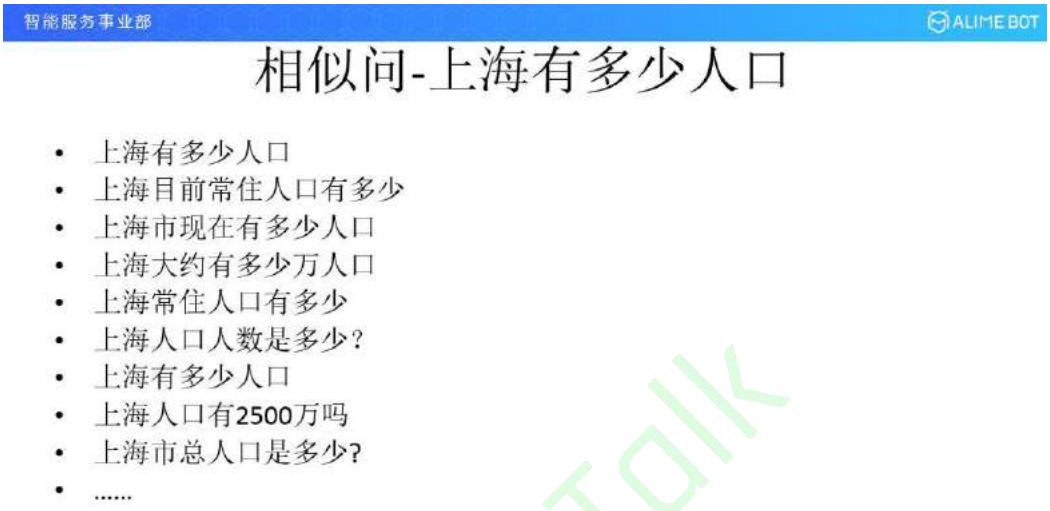
导读：大家好，我今天报告的题目是知识结构化推动智能客服升级，主要为大家介绍阿里小蜜在知识图谱问答（结构化智能问答）方面的工作，我们的基本思路是：通过知识结构化，解决现有智能客服系统在知识管理和语言理解上的痛点，为客户带来高效复用、精准理解、精细管理等方面的显著收益。

大家对智能客服应该都有一定的了解，智能客服在中国已经有很长的历史了，从 2007 年开始就有很多企业逐渐的应用了智能客服，这种智能客服，现在可以称为第一代智能客服。最近几年，很多大型企业，包括一些中小企业都希望升级到新一代智能客服。今天的报告分为以下四个部分：

- 新一代智能客服的主要需求是什么？
- 面对这些需求，我们主要的解决思路是什么？
- 基于结构化知识的解决方案。
- 使用这些解决方案的时候会得到什么样的收益。

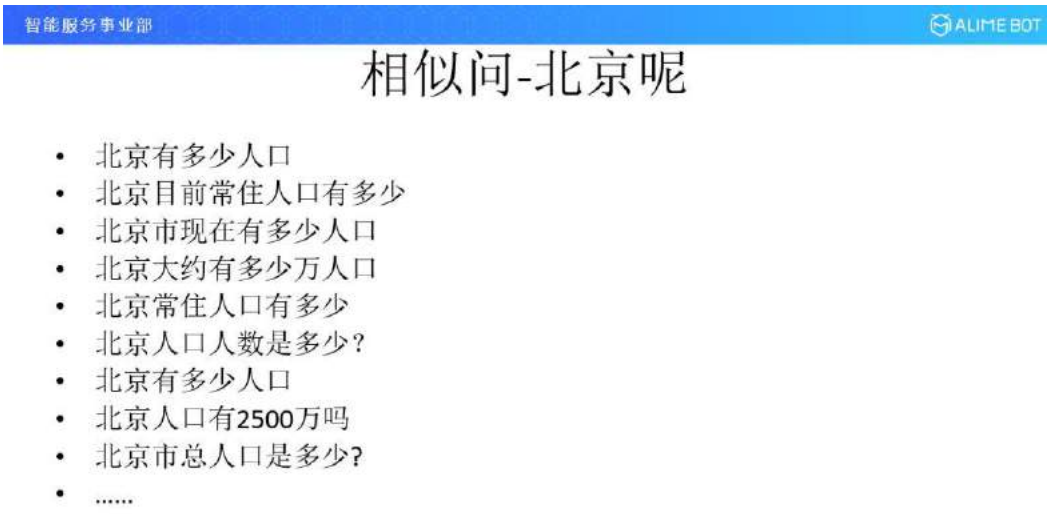
智能客服升级的需求

首先看下智能客服升级的需求。



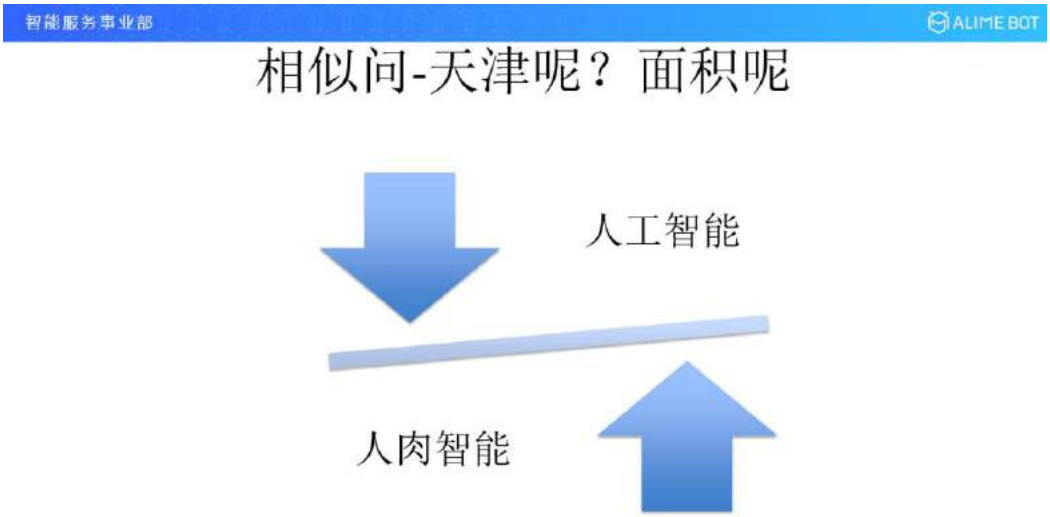
这个例子，大家一看就会明白，现在用很多智能客服的时候它都让你配很多标准问，然后再为每个标准问配很多的相似问。一些大型智能客服系统会有几千个标准问，每个标准问需要配很多相似问，才能得到差强人意的效果。这里举了一个简单的例子，比如问上海有多少人口，你可以给配置相似问，

这些相似问用于覆盖同一个问题的各种可能的说法，因此最终系统可以将各种不同的 query 定位到同样的标准问上，进而可以给出统一的答案。



如果把这个句子中的“上海”换成“北京”呢？你又需要给它很多类似的说法。依此类推，这样需要配备的相似问数量是非常巨大的，你肯定不愿意这样蛮干。但是现在大多数智能客服系统，就是这么做的。所以维护客服系统的运营人员都是很没有成就感的，每天都在做大量重复性的工作。

DataFunTalk



这也是人们将人工智能诟病为人肉智能的主要原因。

我们可以把目前智能客服系统存在的问题归纳成两个痛点：

一是知识管理痛点。成千上万的知识点，缺乏关联，给知识运营和管理带来困扰，却又需要维护大量的相似问。其表现出来的两种现状：

- 设置多个知识点，答案细致，但知识管理任务重。

- 设置 1 个知识点，管理任务轻，但答案过粗。

这两种方法是两种极端，都没有很好的解决这个痛点，或者导致知识管理任务重，或者导致用户使用体验差。

二是语言理解难点。语言具有多样性、歧义性、复杂性、复用性、模糊性等特点。现在的系统即使使用了深度学习算法，也不能很好地解决复杂问题、模糊问题，在迁移到新的领域时也不能很好地复用原有领域的训练语料。

DataFunTalk

2017 年用户输入“上海有多少人口？”这样的问题时，机器人能够自动问答“2017 年上海的常住人口是：2419.70 万”，我们就可以认为机器人具有一定智能。在传统的智能问答（FAQ）中，知识运营的主要工作是为每个标准问题配置一定数量的同义问法，比如“上海有多少人口”“上海目前常住人口有多少”“上海市现在有多少人口”等等。对于“天津有多少人口”这个标准问题，再配置对应的同义问法，“天津有多少人口”“天津目前常住人口有多少”“天津市现在有多少人口”等等。

从传统智能问答到知识图谱问答，第一个重要变化是引入知识图谱，具体包括：

① 将实体归纳为实体类型：将“上海”归纳为“城市”类实体，将“2017 年”归纳为“时间”类实体；

② 将标准问题归纳成属性：将“[城市]有多少人口”映射到“城市”类实体的属性“常住人口”上；

③ 抽取实体-属性-值：将具体的知识组织成“(上海,常住人口,2419.70 万)”这样的包含简单属性值的三元组，进而组织成“(上海,常住人口,(2017 年，2419.70 万))”这样的包含复合属性值（CVT，Compound Value Type）的 N 元组。

第二个重要变化是在语言理解算法从分类或匹配算法改为 Semantic Parsing 算法。传统智能问答在语言理解部分只需要将用户问题映射为一个标准问题，而 Semantic parsing 则需要对用户问题进行深入理解，并输出一个结构化的语义表达式，可以直接从知识图谱中查询答案，或者经过推理后产生相应的答案。在语言理解算法的核心部分，Semantic Parsing 算法有三种主要的形式：

- seq2seq 的端到端方案
- 传统的 Semantic parsing 方案
- 多阶段的融合深度学习的 Semantic parsing 方案

知识结构化的主要思路

结构化的主要思路可以分为三个方面：

① 从非结构化业务文档或者半结构化的数据过渡到结构化知识图谱

② 从非结构化的用户表述解析出结构化的语义表达式

③ 从非结构的文本型答案升级为结构化答案

DataFunTalk

知识图谱的概念大家都比较熟悉，知识图谱实际上是网络化的结构。

知识图谱的构建一个非常复杂的过程，我们在内部有一套非常复杂的平台，从知识建模、挖掘、融合、编辑与管理到推理与计算，它用于构建和管理超级复杂的知识图谱。在构建服务于智能客服的知识图谱时，一般不需要使用这个超级复杂的知识图谱构建与管理平台，而是使用一个相对轻量级的、人机互助的知识图谱编辑和管理平台。

DataFunTalk

这是阿里巴巴商品知识图谱的示例，它把淘宝天猫中上百亿种商品构建成一个非常庞大的商品知识图谱，在淘宝和天猫平台治理、智能客服、导购等许多方面发挥了很大作用。

但是这种知识图谱的量与我们所要服务的个体企业的商品和服务在量级上是完全不可同日而语的。多数大型企业和机构能有几万种商品就已经相当可观了。

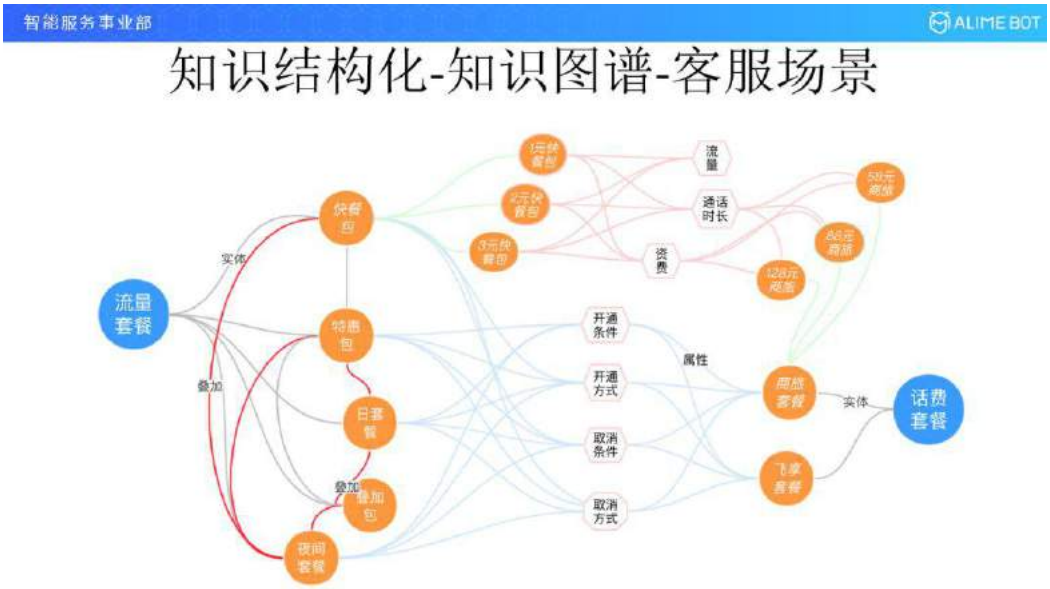
1. 构建结构化知识图谱

DataFunTalk

第一块，将非结构化的业务文档或半结构化数据转换成结构化的知识图谱。我们在 To B 的场景中，每一个企业都需要建立知识图谱，却不可能像建淘宝天猫商品知识图谱那样，投入那么多的时间和人力去建设，否则客户会觉得成本太贵了，承受不起。

但是知识结构化的目标仍然是可以以较低成本实现的。

这是一个很简单的文档的例子，为一个套餐提供了非结构化的表述。我们最终是希望从非结构化表述中整理出一个关于这个业务场景的 schema，它有实体类型、实体、简单属性/复合属性等。最后把各种实体的值都填上，就得到了这个场景的知识图谱。这个知识图谱对于客户来说通常不会很复杂，因为一般客户所提供的服务和产品均不会大于万这个量级，所以它的知识图谱不会特别复杂，构建成本也没有那么高。



这是为移动运营商做的知识图谱的一个简单示例。它的实体类型有流量套餐、话费套餐等，然后每个套餐下面有很多实体，实体下面有很多属性。

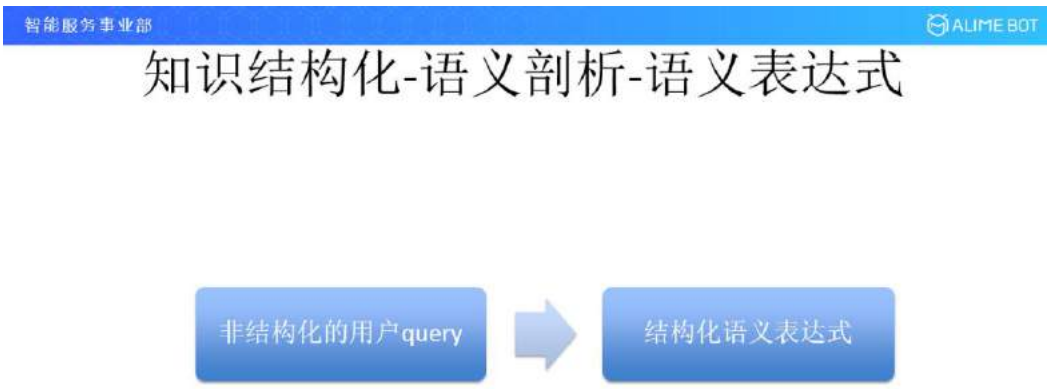
在建知识图谱的过程中，抽象出 Schema 是最关键的一步，包括归纳出实体类型以及各实体类型下的各种属性。智能客服厂商对客户的业务了解不够，

客户自身对知识图谱的概念缺乏了解，因此沟通的成本较高。那么，应该如何快速构建一个定制化的知识图谱呢？一个非常简单而又有效的做法是从客户现有的 FAQ 知识库出发，快速地归纳 Schema 和知识图谱，比如这是我们内部的一个场景，原来大概有 700 个标准问，我们把它归纳完之后就变成了 70 个属性。

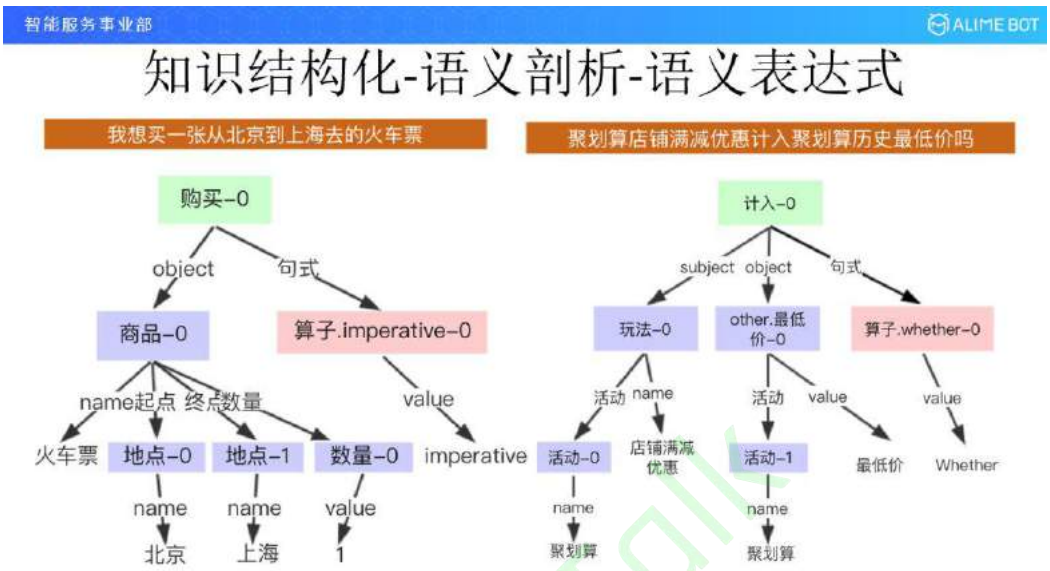
然后这个是移动运营商的场景，大概是有 4000 个标准问，归纳完之后是不超过 400 个的属性。

所以从知识管理角度来说，通过知识结构化可以大幅度降低知识量级，而且耗时比较短，成本比较低。

2. 结构化语义表达式

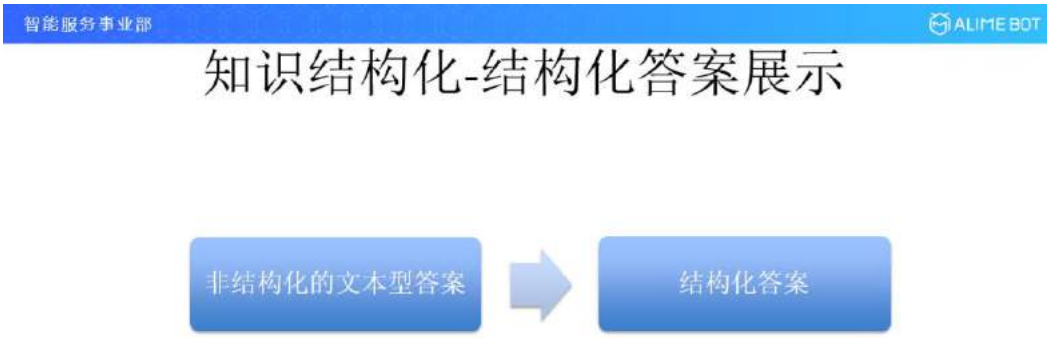


第二块是语义理解部分，把非结构化的用户 query 变成结构化语义表达式，就是我们通常所说的 Semantic Parser 的过程。



上面是一句话，我们希望得到这样一个结构化的表达，最上面是动词，下面有动词的客体，客体下面还有它的修饰成分。

3. 结构化答案



第三块，我们把原来非结构化的文本型答案变成结构化答案。

这是淘宝天猫双十一的一个问题，这个问题在不同的时间段，回答的答案是不一样的，所以我们就把它做成了一个时间性的结构化答案。用户在不同的时间段进来可以看到对应时间段的答案。

这是一个弹性的答案，用户可以问的很笼统，也可以问的很细。

知识图谱问答解决方案

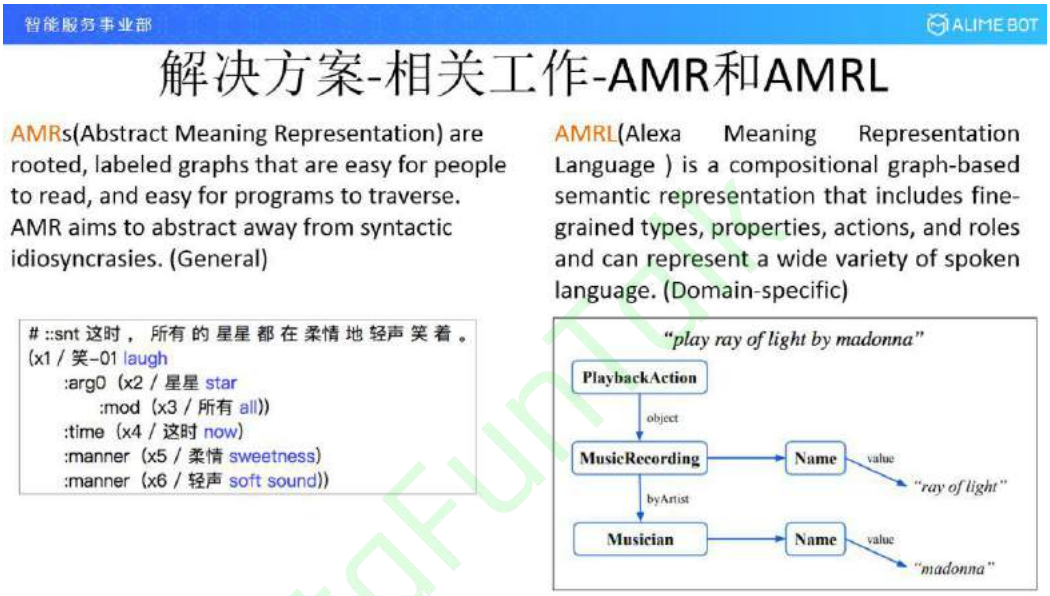
我们的知识图谱问答解决方案主要可以分为三个方面：

① KAMR（Knowledge-driven Abstract Meaning Representation），是由我们提出的语义表达体系。我们参照的对象是 AMR 和 AMRL。

② KAMR Parser，基于 KAMR 语义体系的 Semantic Parser。KAMR Parser 是我们第二代的 Semantic Parser，上一代的 Parser 是 MulCG Parser。

③ 结构化问答平台，是以 KAMR 为核心的包括人工智能闭环在内的结构化问答平台，将 FAQ 问答和知识图谱问答融为一体。在上一代产品中，FAQ 问答和知识图谱问答采用两个不同的语言理解引擎。

1. 语义表达体系



语义表达体系方面我们参考的对象是 AMR（Abstract Meaning Representation），是由美国学者提出的语义表达体系，是一种通用的语义表

达体系。AMR 的特点是将句子中的词抽象为概念，因而使最终的语义表达式与原始的句子没有直接的对应关系，这是它与语义角色标注、依存语法等的一个显著区别。AMR 是 2013 年提出来的，本身并不是全新的体系，但是到目前为止 AMR 仍然是自然语言处理学界对于深度语言理解的最好最系统的解决方案。

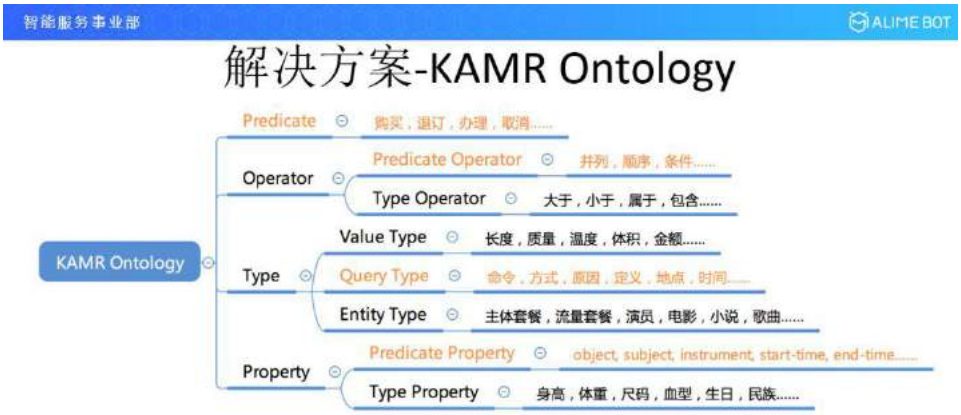
另外一个参考对象是亚马逊基于 AMR 提出的 AMRL，其语义表达形式与 AMR 一脉相承，但主要应用于任务型场景。

2. KAMR

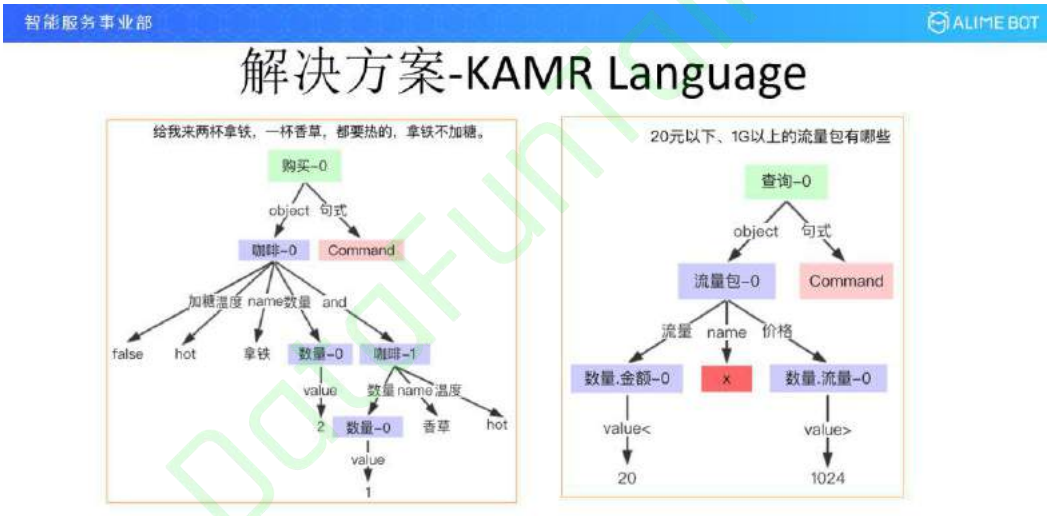
通过上述参考，结合我们的业务场景，我们提出了基于知识驱动的抽象语义表达 KAMR，主要由 KAMR Ontology 和 KAMR Language 组成，KAMR 框架可以支持跨 domain 的 query、弹性粒度的 type、复杂的语义以及各

种组合，进而可以支持精细理解、弹性理解和高效复用（从企业客户角度来说，非常注重高效复用，因为我们希望同一个场景中数据之间可以复用，跨场景也可以复用，以及跨客户复用，这样才能降低成本，提高效率）。

刚刚提到语言理解有很多痛点，我们的 KAMR 着力于解决四个方面的问题：歧义性、复杂性、复用性、模糊性。

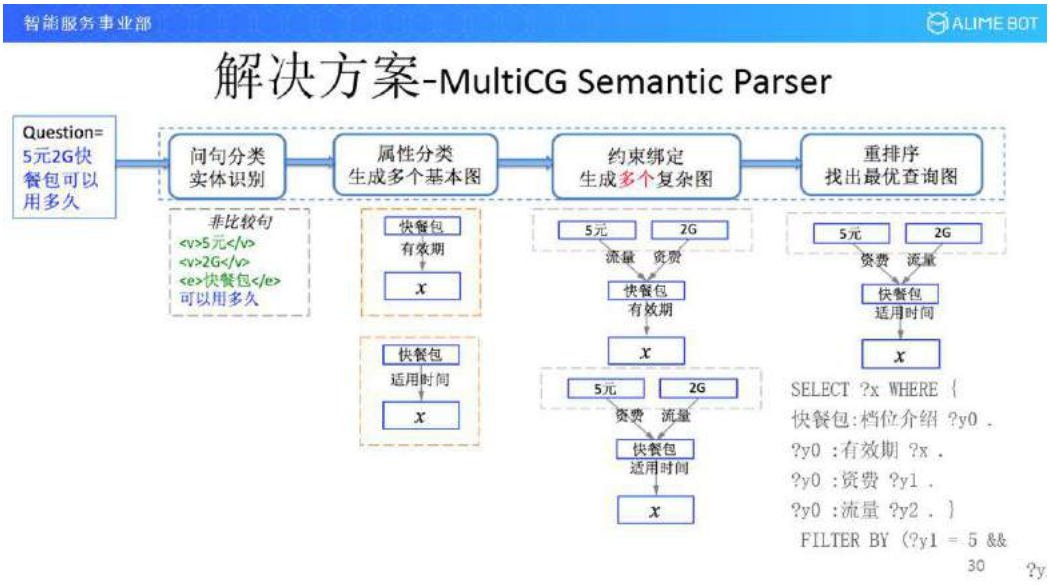


这是 KAMR Ontology，包括谓词、算子、Type、Property 等部分，每个部分都是一个层次化的 Ontology。



这是 KAMR 语义表达的两个例子。KAMR 本质上是有向无环图。从表达能力上讲，它可以表达多谓词、省略、歧义、复杂约束条件、并列结构等各种复杂现象。

3. Semantic Parser

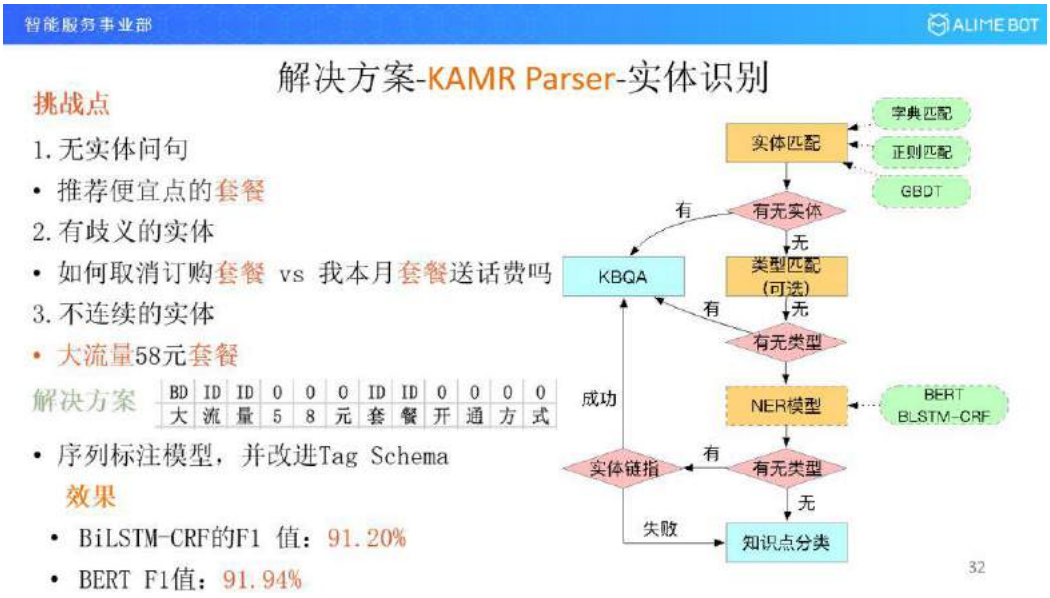


这是我们第一代的 Semantic Parser，其流程包含四个步骤：实体识别->属性分类->约束绑定->重排序。其中属性分类、重排序都使用了深度学习。目前很多做知识问答的系统，已经使用了端到端的系统，直接从 query 得到一个结构化的语义表达，但是这类系统在精度上离我们所采用的 Pipeline 方案有较大的差距。Pipeline 方案的优点是每一步都比较容易的进行人工的干预，精确度也远远的高于目前纯粹的端到端系统。端到端的系统对训练数据量要求非常高，我们在多数情况下还不能获取足够多的训练数据。因此，在这种情况下，Pipeline 的方式更为合适。

针对实际业务场景中遇到的问题，我们对第一代 MultCG Semantic Parser 进行针对性地系统改进，进而发展出第二代 Semantic Parser，即 KAMR Parser。它包含实体识别、属性识别、约束识别和 Semantic Dependency Parser 四个模块，每个模块的内涵与第一代 Parser 相比也有本质的变化。

① 实体识别

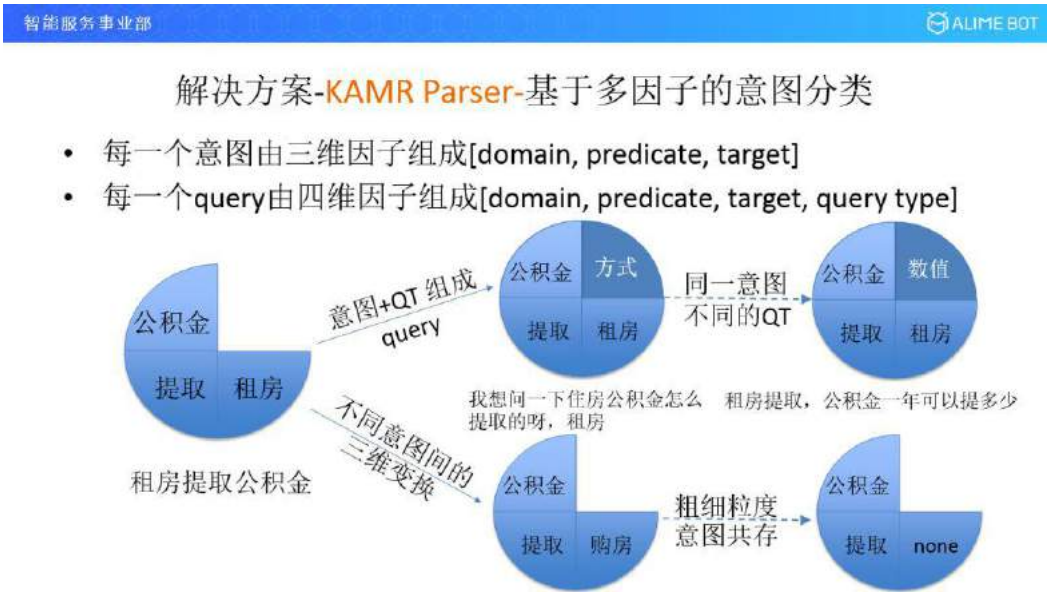
DataFunTalk



首先是实体识别，我们在一些场景中落地的时候，发现实体识别远远比我们想象的要复杂，比如中移动上有一些套餐，大流量 58 元套餐，本来这个实体叫大流量套餐就可以了，但是这个实体名被数字隔开了，这是我们之前没有遇到的。因此，我们引入了模型化的实体识别，这里模型采用的是比较常见的 BLSTM+CRF 模型。特别的地方在于，序列标签可以是不连续的，就可以把前面的 BD、ID 和后面的 ID 合成一个实体。

② 意图识别

DataFunTalk



更核心的是意图/属性识别部分，这里我们提出的是基于多因子的意图分类方案：

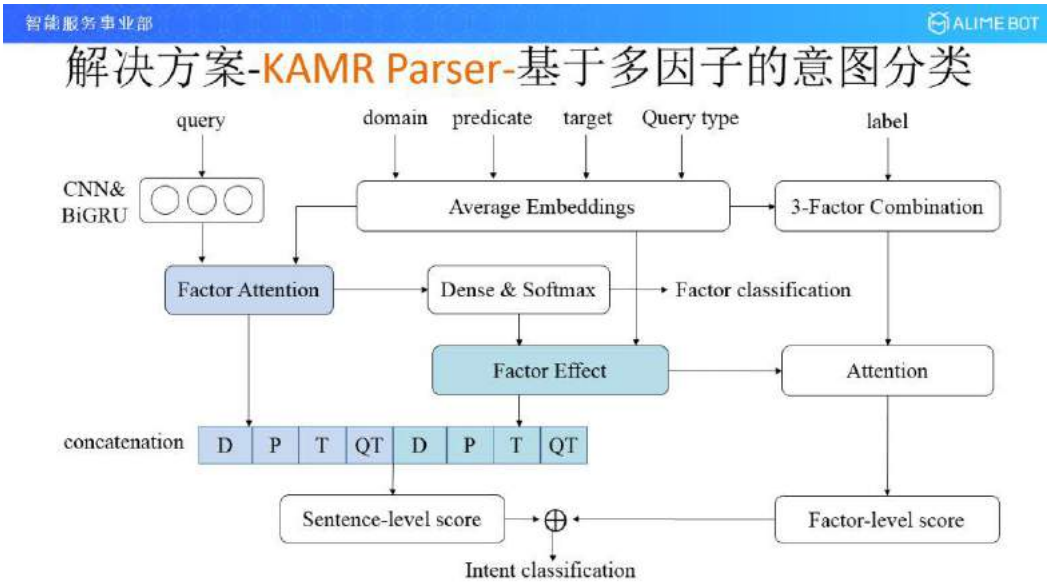
- 每一个意图由三维因子组成（domain，predicate，target）
- 每一个 query 由四维因子组成（domain ,predicate ,target ,query type）
- query 的四维因子中的前面三维可以直接从它所属的意图继承而来 ,query type 则是个性化的，同一个意图下的多个 query 可以属于不同的 query type。

DataFunTalk

intent	domain	predicate	target
社保参保	社保	参保	none
社保缴存基数	社保	缴存	基数
社保基数申报	社保	申报	基数
社保卡办理	社保卡	办理	none
社保卡新办	社保卡	新办	none
社保卡补办	社保卡	补办	none
社保卡信息变更	社保卡	变更	信息
社保卡初始密码	社保卡	none	初始密码
社保卡挂失	社保卡	挂失	none
社保卡异常	社保卡	none	异常原因
社保拉取缴存证明	社保	拉取	缴存证明
社保转移	社保	转移	none
社保补缴	社保	补缴	none
职工中断社保补缴	社保	补缴	职工
医保	医保	none	none
医保参保	医保	参保	none
城镇职工医保参保材料	医保	参保	城镇职工
城乡居民医保参保材料	医保	参保	城乡居民
成人医保参保材料	医保	参保	成人
在校学生医保参保材料	医保	参保	在校学生
大学生意外伤害险参保材料	医保	参保	大学生
医保报销比例	医保	报销	比例
城镇职工医疗保险报销比例	医保	报销	城镇职工
城乡医疗保险报销比例	医保	报销	城乡
公务员补助报销比例	医保	报销	公务员
大学生意外保险报销比例	医保	报销	大学生
市外就医报销	医保	报销	市外
医保报销账户	医保账户	报销	none
医保账户金额	医保账户	none	金额
医保账户使用规则	医保账户	使用	规则
医保报销账户建立标准及使用范围	医保账户	建立 使用	标准 范围
医保报销账户到账及生效时间	医保账户	到账 生效	时间
医保报销账户余额利息及提现	医保账户	提现	余额利息
申请特殊病种待遇	特殊慢性	申请	待遇
特殊病种申请对象	特殊慢性	申请	对象
特殊病种申请材料	特殊慢性	申请	材料
申请慢性病待遇	特殊慢性	申请	待遇
医保卡无法使用	医保	使用	异常原因
医保报销上限	医保	报销	上限
家庭共济	家庭共济	none	家庭共济
家庭共济使用对象	家庭共济	使用	对象

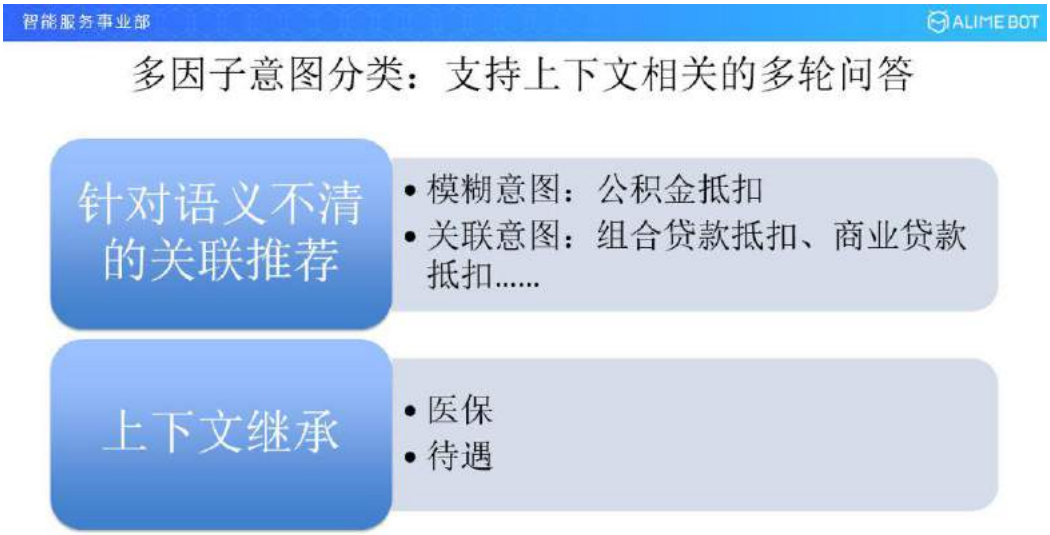
DataFunTalk

这是我们在某个业务场景下对意图进行多因子分析的例子，其中 none 表示有些因子的值可以为空。



这个是我们的多因子意图分类模型，主要包含两个模块，第一个就是编码核心，可选择诸如 CNN、LSTM、Gated CNN、Transformer 之类的方法进行 encoder。剩下的关键的就是把四维因子和原始 query 之间建立 Attention，通过因子与 query 之间的 attention 对 query 的表示进行引导，将与因子相关的部分进行增强。

当把一个意图拆解到因子这个粒度的时候，可以实现意图间的语料复用，因为一个场景中不同的意图是有些相同部分的，比如“办理”这个 predicate，可能出现在“办理方式”、“办理条件”、“办理时间”等意图中，这里的 predicate 都是一样的。之前把意图作为一个整体的时候，是没法利用 predicate 层的共享信息的。还可以解决不同意图语料之间不平衡的问题。



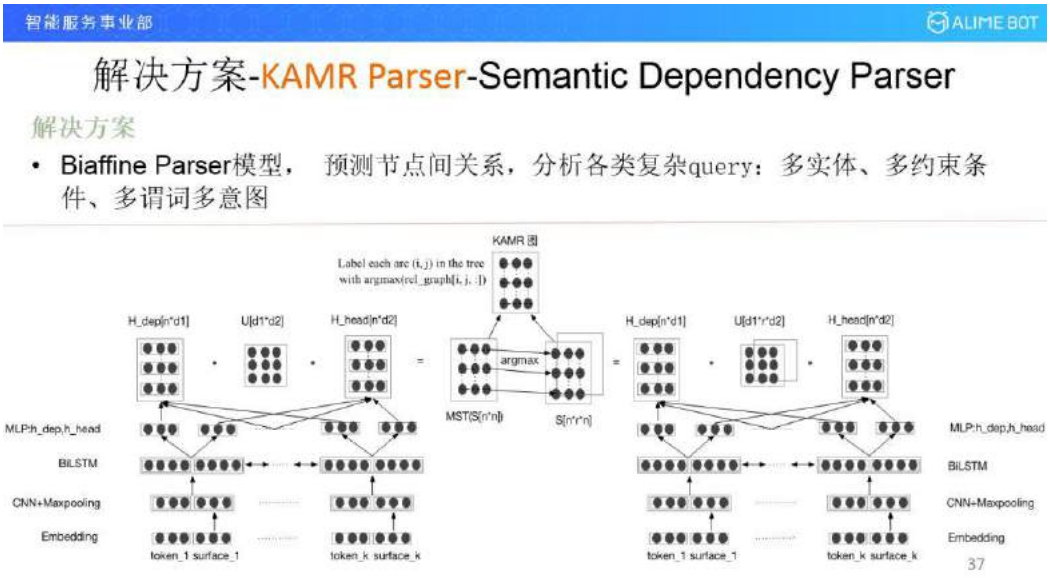
多因子意图分类，还可以发挥其他作用：

针对语义不清的关联推荐：比如用户提出了一个比较模糊的意图“公积金抵扣”，这时我们就可以给用户推荐一些关联的意图：组合贷款抵扣，商业贷款抵扣……。

上下文继承：用户在第一轮说了“医保”，可以解析出 domain 这一维因子。接下来第二轮说了“待遇”，可以解析出 target 这一维因子。把两者组合起来，就可以分析出“Domain:医保；Target:待遇”这样的意图。

③ Semantic Dependency Parser

DataFunTalk



在实体识别和意图识别阶段系统可以输出多个结果，可能存在多个实体、多个意图、多个约束条件以及算子，因此还需要有一个模块将多个结果组合起来形成一个完整的语义表达式。在此，我们使用的是基于 Biaffine 的 Semantic Dependency Parser。这是一个简洁高效的模型，但是速度很快，效果也很好，可以用来解决复杂问题，如多实体、多约束条件、多谓词多意图等。Biaffine Dependency Parser 本质上是一个基于最大生成树的依存分析器。

4. 结构化问答平台

DataFunTalk

这是我们的结构化问答平台的整体架构图，分为算法层和知识层两部分。这个平台中为客户可见的部分主要是 KG 编辑平台，包括知识图谱 Schema 编辑器和实体知识编辑器，以及目前服务于结构化问答的智能闭环已经投入使用，支持用户在线进行无答案或未解决问题筛选、人工标注、模型训练、评测、模型部署等。

DataFunTalk

知识图谱 Schema 编辑器的界面

知识图谱实体知识编辑器的界面

知识结构化的收益

基于我们的知识图谱问答（结构化智能问答）解决方法，可以为客户带来以下几个方面的收益：

高效复用：如果一个企业有几百种产品（实体），可以达到举一反千的效果；如果新增加一个或多个实体，无需新增相似问，直接把实体和属性的值填上，就可以生效；同时还实现了知识的瘦身，将知识点数量从 4000 减少到 400 左右；并且做到了同场景复用、跨场景复用、跨客户复用。

精准理解：通过 Semantic Parsing，KBQA 对问句的理解更加深入，它可以知道一个问句涉及到的实体是什么实体，属性是什么，带有什么约束条件。比如“3 元 5G 流量快餐包如何办理”这个句子中，实体是“流量快餐包”，属性是“办理方法”，约束条件是“流量=5G，资费=3 元”。传统智能问答在回答这类问题时，将不同档位的流量快餐包的答案聚合在一起输出给用户，答案冗长。KBQA 在精准理解的基础上，可以给出更细致的回答，直接回复“3 元 5G 流量快餐包的办理方法是：……”。

精细管理：当知识数量较大时，增、删、改、查将会成为一个难题。通过知识结构化，从实体、属性、CVT 子属性等多个维度对知识进行系统的梳理，知识的增、删、改、查将会更加便捷。

推理计算：在结构化知识的基础上可以进行推理和计算。比如，对于“阿里巴巴几岁了”这样的问题，在知识图谱中存储的具体知识三元组是（阿里巴巴，创立于，1999 年），KBQA 经过一个从创立时间到年龄的计算过程：年

龄=当前年份-1999 年，即可计算出“阿里巴巴的年龄是：19 岁”。不需要每隔一年去修改一次答案。

总结

最后再小结一下今天报告的主要内容：

- 需求，分析了知识结构化的需求，包括知识管理和语言理解；
- 思路，知识结构化为分为知识图谱构建、语义剖析和答案展示三个方面；
- 方案，知识结构化的方案包括 KAMR 语义表达体系、KAMR Parser 及知识图谱问答平台三个部分；
- 收益，知识结构化解决方案带来的收益主要是高效复用、精准理解和精细化管理。

知识就是力量，结构化的知识更有力量。今天的分享到此结束，谢谢大家！

DataFunTalk

阿里云小蜜对话机器人背后的核心算法

嘉宾：唐呈光　编辑：刘汝洲

导读：阿里小蜜智能对话开发平台是智能服务事业部推出的面向各行各业的对话构建平台，此次分享将结合平台，对小样本下的语言理解、用户模拟器和基于模型的对话管理的算法研究和落地进行介绍。

0. 对话系统简介

对话系统的一般架构如图：

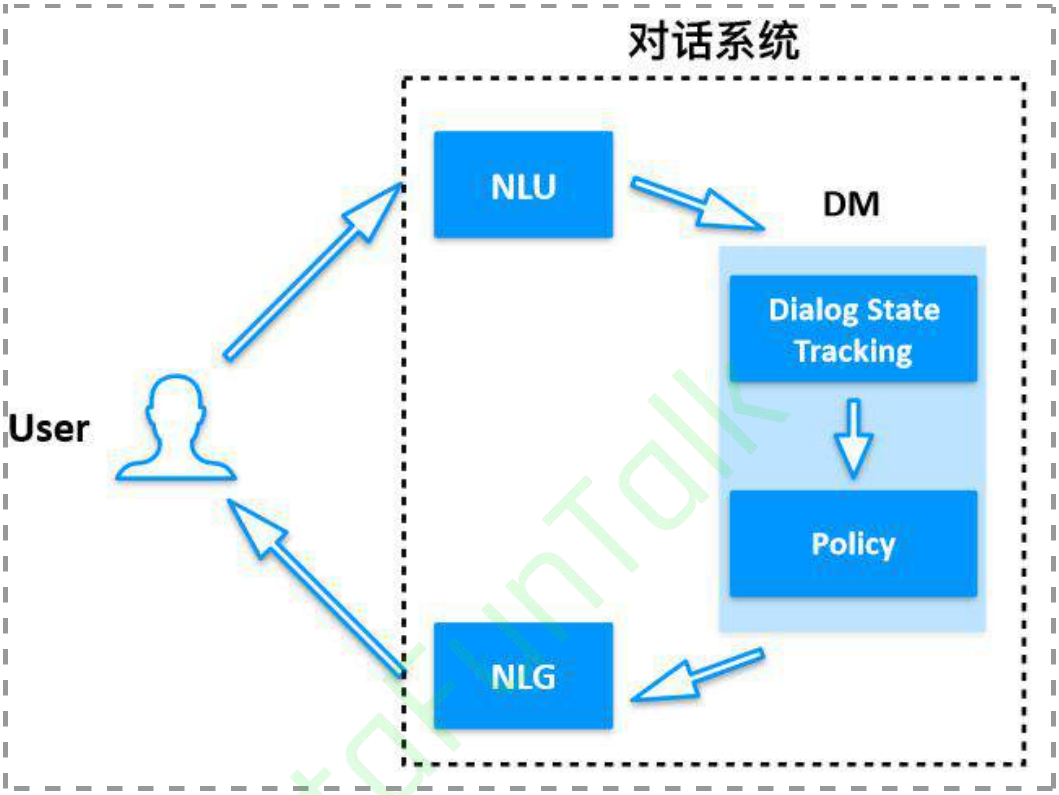


图 1：对话系统一般架构

这是我们所熟知的对话系统框架 ,这里面主要有 :NLU 自然语言理解 ,DM 对话管理，NLG 自然语言生成 3 个主要模块，DM 里面有 dialog state tracking 用于对话状态追踪，policy 用于对话策略管理。

当我们在执行一个对话任务时，例如“开发票”，系统不仅要识别用户的需求，还需要与外部系统对接，进行订单号的合法性校验，调用开发票接口等，这时候 DM 不仅要完成与用户的交互、管理槽位信息 ,还需要访问外部接口，管理调用的结果。

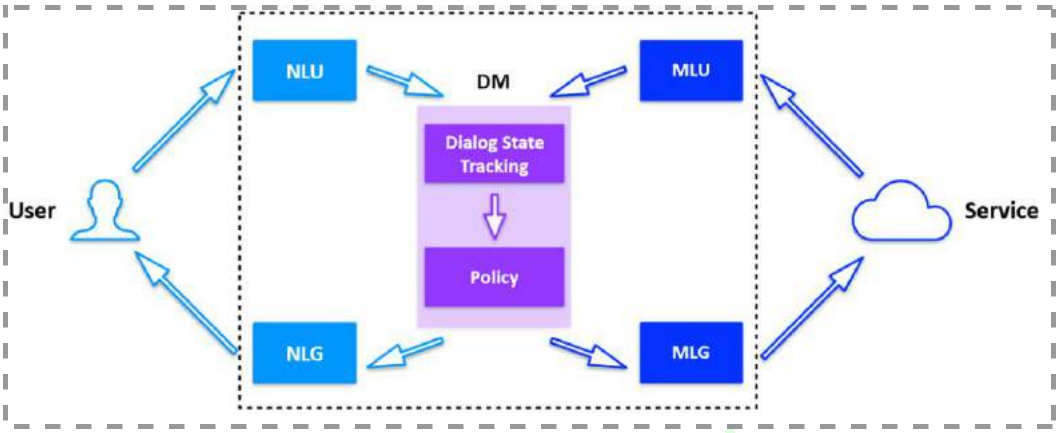


图 2：完整的对话系统架构

当构建好一个机器人后，还需要进行对话的诊断，效果评测，才能发布上线，如果这些工作全都让人来完成，整个过程会非常费力，于是我们引入了用户模拟器来提高整体交付效率。

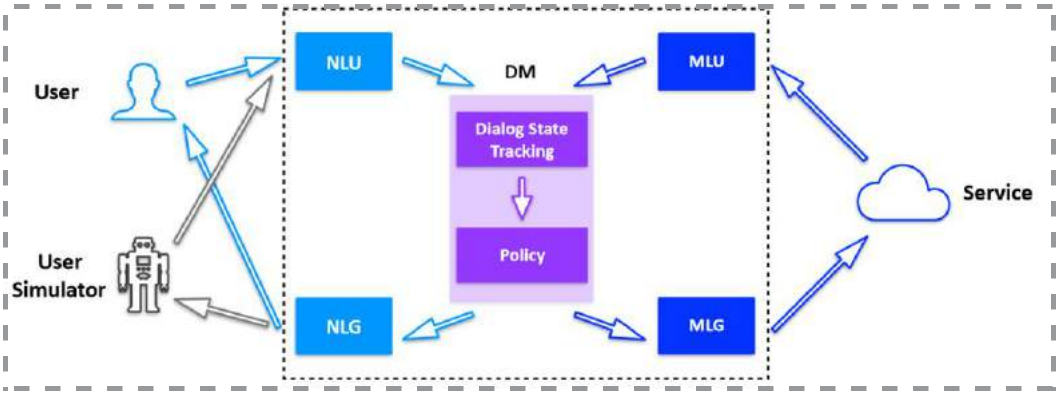


图 3：用户模拟器应用到对话系统

云小蜜对话机器人核心算法主要包括三部分 :1. 自然语言理解 ;2. 对话管理；3. 用户模拟器。

1. 自然语言理解

由于云小蜜对话机器人需要满足各行各业各种场景下的对话服务需求，所以我们的自然语言理解是平台视角下的自然语言理解。根据训练样本的多少，我们把它分为 3 种不同的情况：**无样本、小样本、多样本**。在没有样本的情况下，我们提供了一套简单易懂的规则表示语法，帮助用户实现快速冷启动。以查天气为例，用户只需写 1 条规则，就能表示 100 多个句子。

DataFunTalk

图 4：无样本自然语言处理解决方案

在小样本的情况下，比如共 10 个类别的意图，每个意图下有十多个样本，这种情况下还不足以训练一个有监督模型，但我们可以借助平台数据积累的优势，当只有少量样本的情况下，也可以做出比较好的结果。

实现思路：我们先整理出一个大数量级的数据（十万级别），每一个类目几十条数据，为它建立 meta-learning 任务。对于一个具体任务来说：构建支撑集和预测集，通过 few-shot learning 的方法训练出 model，同时与预测集的 query 进行比较，计算 loss 并更新参数，然后不断迭代让其收敛。这只是一个 meta-learning 任务，我们可以反复抽样获得一系列这样的任务，不断优化同一个模型。在线预测阶段，用户标注的少量样本就是支撑集，将 query 输入模型获得分类结果。实验表明，few-shot learning 的效果优于无监督相似度匹配的方法。

DataFunTalk

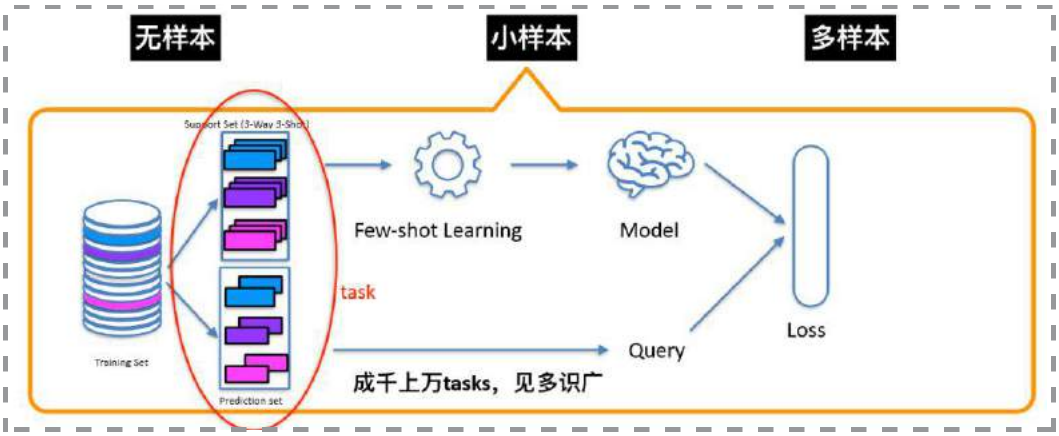


图 5：小样本自然语言处理方案

具体是怎么实现的呢？我们借鉴了图像领域的工作，图像领域大多数工作都只考虑了样本的信息，但是在 NLP 领域样本的信息可能会存在噪声或冗余，比如说“开发票”场景，用户在表述开发票这个事情，除了常见的“我要开发票“、”帮忙开一张发票“以外，他还可能会说：“你好，我前两天在你们店里买了一条裙子，请问现在能帮我开下发票吗？”而这样的句子是普遍存在的，我们需要对这些句子进行归纳，得到类别的信息，然后再与要预测的 query 比较语义相似度。它的神经网络结构分为 3 部分 ,首先是 Encoder 将句子变成句子向量，然后再通过 Induction Network 变成类向量，最后通过 Relation Network 计算向量距离，输出最终的结果。

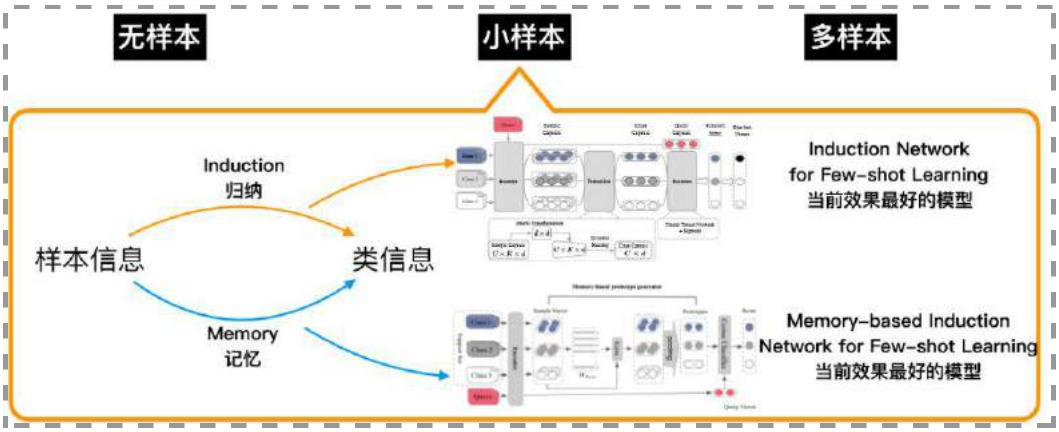


图 6：小样本训练算法

Memory-based Induction Network 是我们在 Induction Network 的基础上引入了 memory 机制，目的是模仿人类的记忆和类比能力，在效果上又有进一步提升。

Induction Network 很关键的一部分就是怎么把样本向量抽象到类向量，我们采用的是 matrix transformation 的方法，下图显示的是 1 个 5-way 10-shot 的数据，转换前，几个类很难区分，类中心不够内聚，转换后，类边界更清晰，更利于下游 relation 的计算。

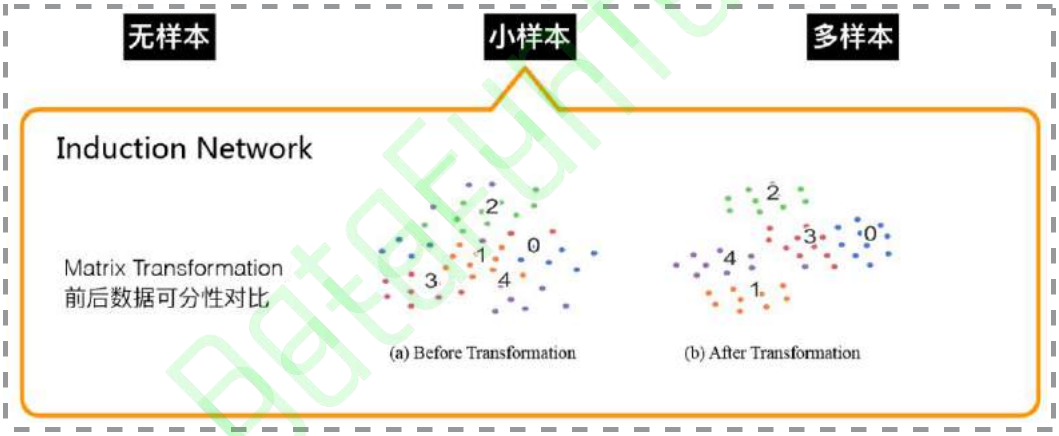


图 7：样本空间转换

当业务方有一定标注数据的情况下，我们就考虑上监督模型了。在云小蜜实际业务场景中，企业相关的标注数据的获取成本是比较高的，因此有监督模型的目标是希望让业务方能够在标注数据量不是很大的情况下达到很好的效果，因此我们构建了一个三层的模型，最底层是具有较强迁移能力的通用模型 BERT，在此基础上构建不同行业的模型，最后用相对较少的企业数据来训练模型。这样构建出来的企业的 NLU 分类模型，F1 基本都在 90%+。

这种模型也有缺点，就是它的结构比较复杂，在线预测的时候延时会比较长，在真实生成环境中应用落地有困难，所以我们通过知识蒸馏的方法来进行模型压缩，在效果相当的同时预测效率更快了。

在实际业务场景的多数情况下 ,任务型对话和 FAQ 型问答一般都是同时存在的，我们也引入了多任务学习（ multi-task learning ），能让任务共享底层的信息并互相增强，使得模型具有更强的泛化能力。在政务场景里，我们通过多任务学习， acc 提升两个点以上。

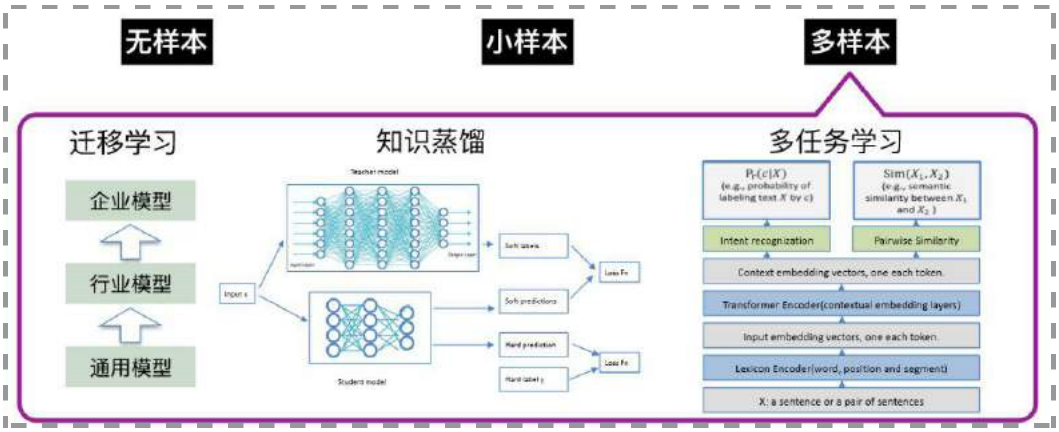


图 8 ：大规模标注集上的多任务学习

简单小结一下，这是 NLU 的整体能力输出的能力版图：

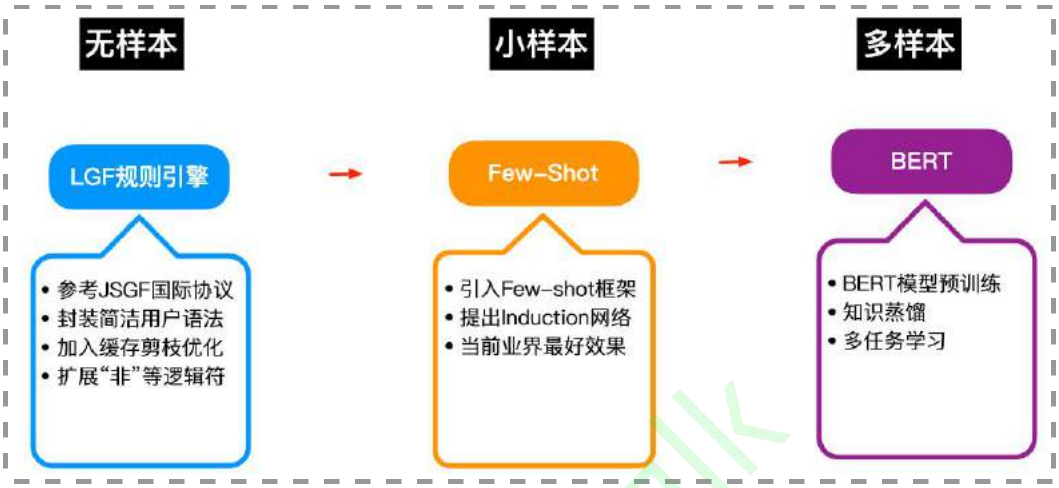


图 9：NLU 的整体能力输出

2. 平台视角下的对话管理

对话管理成功的三要素：

DataFunTalk

1. 业务建模：能够对不同行业不同场景的业务进行抽象，能够用一套统一的表示体系建模，保证业务逻辑的正常运行；

2. 具备鲁棒性：能够很好的处理业务未定义的通用对话需求和各种异常情况；

3. 持续学习的能力：能够在与用户交互的过程中，不断的学习，不断适应新场景，根据用户的反馈调整系统的对话策略。

图 10：平台级对话管理的应用需求

2.1 基于 TaskFlow 的业务建模

我们以“火车票”场景为例，一个有经验的卖火车票的售票员在指导一个新售票员的时候，他会把常见的对话样例描述出来，让新售票员知道用户怎么问，我该怎么答，同时他还会告诉新售票员需要查询哪些系统才能获得票务信息，以及有票和无票的情况下分别怎么回复用户。

基于对上述真实场景的观察，我们对整个交互过程进行抽象，我们认为对话的基本单元是一个 turn（ 一轮 ），它可以拆解为 3 部分：用户说、机器人思

考和机器人回复，分别对应三个基础节点：触发节点、函数节点和回复节点。上面说的是单轮的情形，如果把所有的后一轮的触发节点接到前一轮的回复节点后面，就构成了一个多轮交互（multi-turn）的对话。

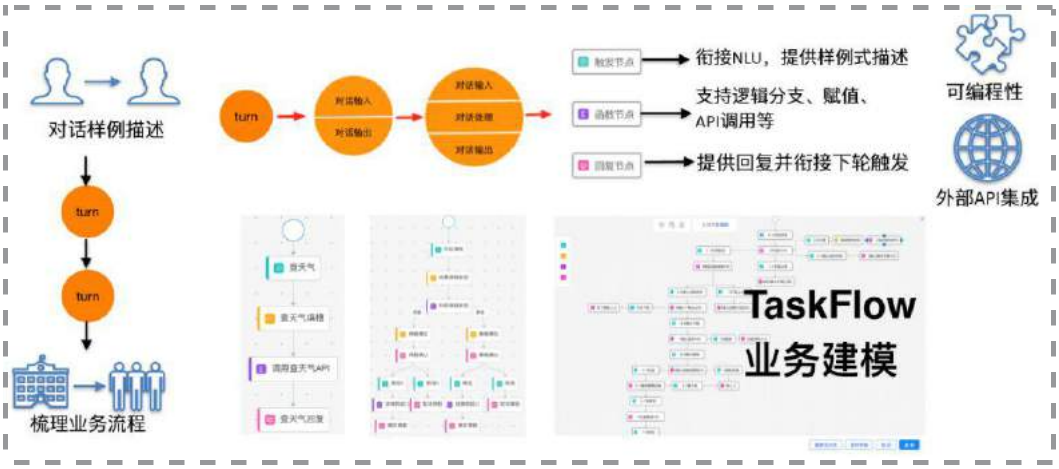


图 11：基于 TaskFlow 的建模框架

为了让 TaskFlow 在平台上执行，我们设计了一种双层状态机的方案，上层是对话逻辑，底层是一套通用的对话引擎，通过这种解耦的设计，不论上层的业务逻辑如何变化，下层都用一套统一的引擎在支撑，如果想赋能上层业务，只需要不断升级底层的能力，上层的所有业务都会受益。

DataFunTalk

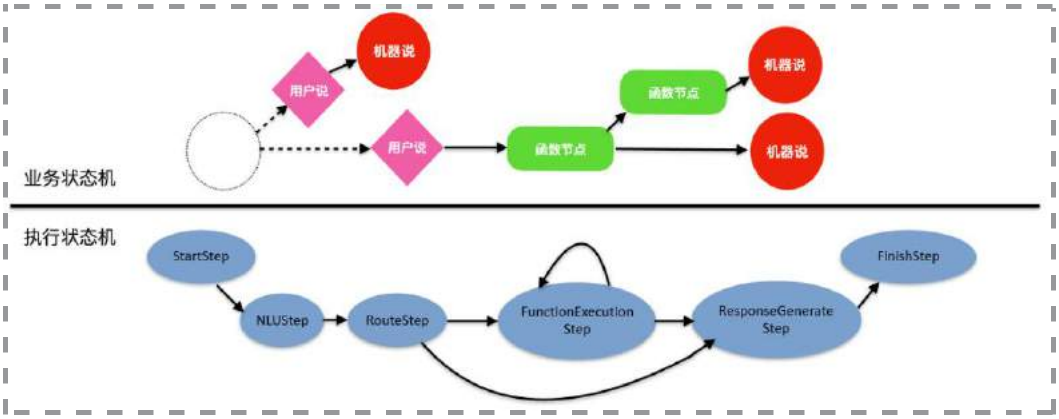


图 12：基于双层状态机的方案

2.2 具备鲁棒性

如果对话管理只能保证用户已定义的业务逻辑正确运行，它的鲁棒性还不够，还需要考虑更多情形，包括：通用对话能力和异常处理能力，如图：

图 13：对话系统的鲁棒性

总体的实现思路是，通过系统内置 TaskFlow 实现对话鲁棒性。以“流量包业务”的对话中模糊澄清为例，我们采用插件化的方式实现，在用户定义的 TaskFlow 基础上，增加澄清 TaskFlow，它的功能包括：

1. 判定是否需要触发澄清；

2. 选择澄清的策略：是隐式澄清还是显示澄清，是单意图澄清还是多意图澄清；

3. 澄清话术的生成。

在上线之前，系统会把这两部分的 Taskflow 进行编译和链接，变成一个可执行的 Taskflow，然后放进执行引擎提供在线服务。其他的功能，比如重听，个性化拒识等都是采用同样的方法实现。

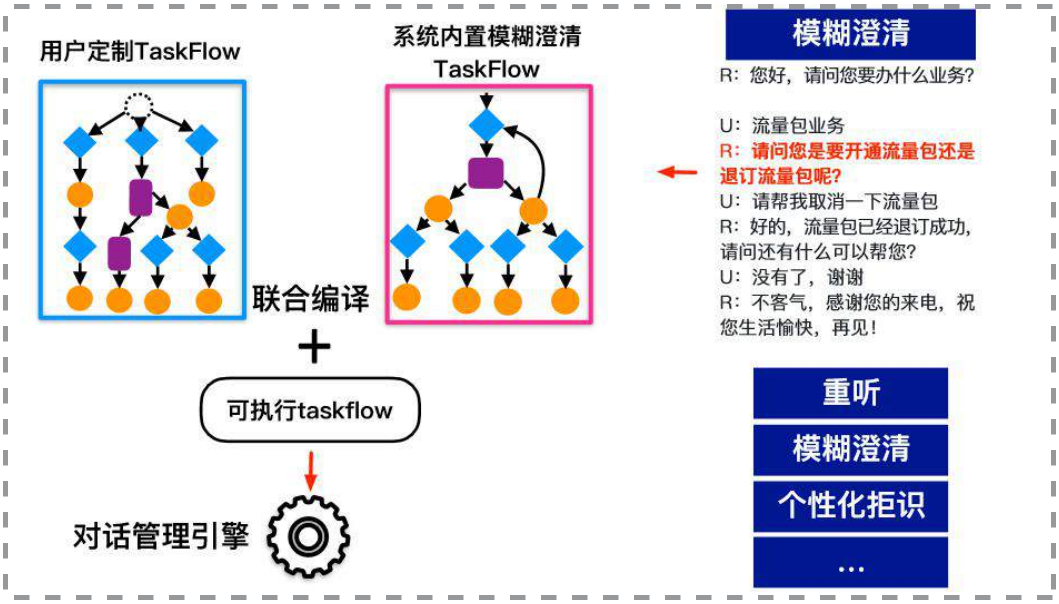


图 14：通过系统内置 TaskFlow 引擎增强系统鲁棒性

2.3 可持续学习

以上的两部分能力都是解决对话中高频、确定的部分问题。如果希望用户在实际对话中越聊越好，仅有以上部分是不够的，需要利用好对话数据，建立

DataFunTalk

对话模型，去 cover 中长尾对话行为，并且基于反馈快速调整对话策略，从而获得更好高价值的智能。

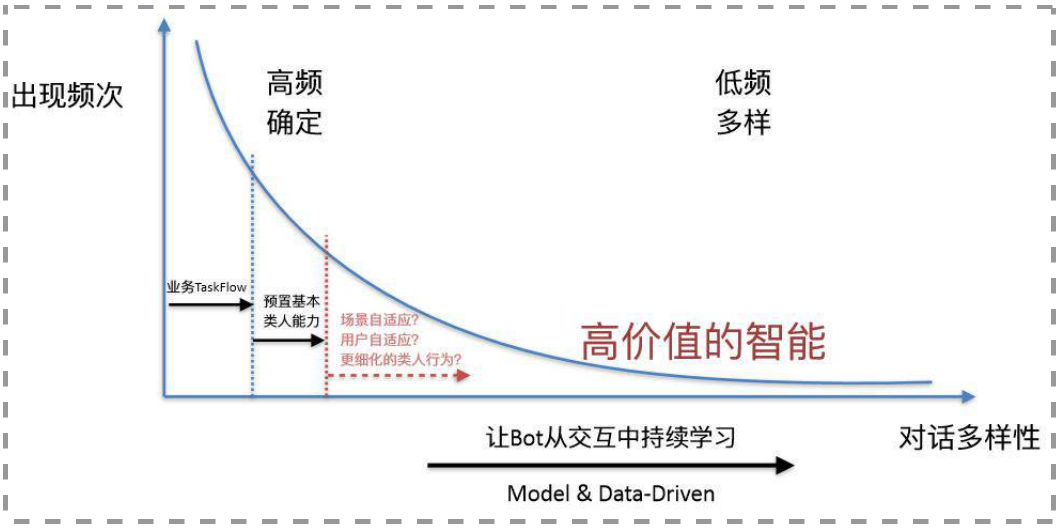


图 15：可持续学习的对话系统

持续学习：

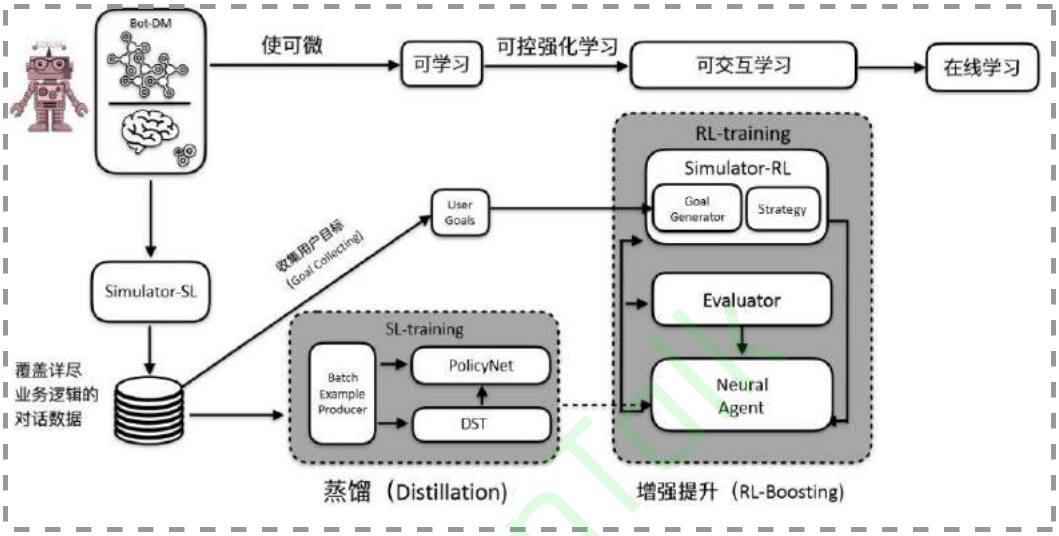


图 16：对话系统增强学习技术架构

总体分为 3 步：

1. 构建 DM 模型，保证它是可学习的；

2. 让模型可交互学习；

3. 支持在线学习。

训练模型得先有数据，我们构建了一个用户模拟器，让它与机器人对话，从而获得大量的带标注的数据，然后分别训练 DST 模型和 Policy 模型，这一步完成了机器人知识的蒸馏，可以获得一个与规则系统效果上等价的 DM 模型。接下来，对 User Goals 进行采样，通过用户模拟器对 DM 模型进行交互，利用 Reward Evaluator 模型进行 Reward 打分，从而获得大量的 Transition 四元组：

<state(t),act(t),state(t+1),reward(t)>

利用增强学习 A2C 算法训练，直至收敛。然后发布到线上，进行在线学习。

2.4 Dialog State Tracking (DST)

DataFunTalk

	学术界	我们
交互主体	人和对话系统（two agents）	multi-agents
state定义	slots	variable（含slots）
slot value假设	离散可枚举	无约束
一轮对话中tracking次数	1次	多次

图 17：阿里与学术界在 DST 的差异

真实场景下，我们的 DST 与学术界相比，有以下几点主要差异：

1. 多智能体建模：DM 不仅跟用户交互，还跟多个外部服务交互，这些服务都可以看做是一个个智能体；

2. 追踪变量：外部返回的结果会存储在变量中，我们的 DST 扩展了数据形式；

3. slot-value 假设：学术界假设 slot value 都是离散可枚举的，哪怕是“时间”类型的值，会通过静态化处理进行简化，这显然和实际情况不符，我们不做这样的假设；

4. 追踪次数：我们对每轮对话中 tracking 的次数不做约束。

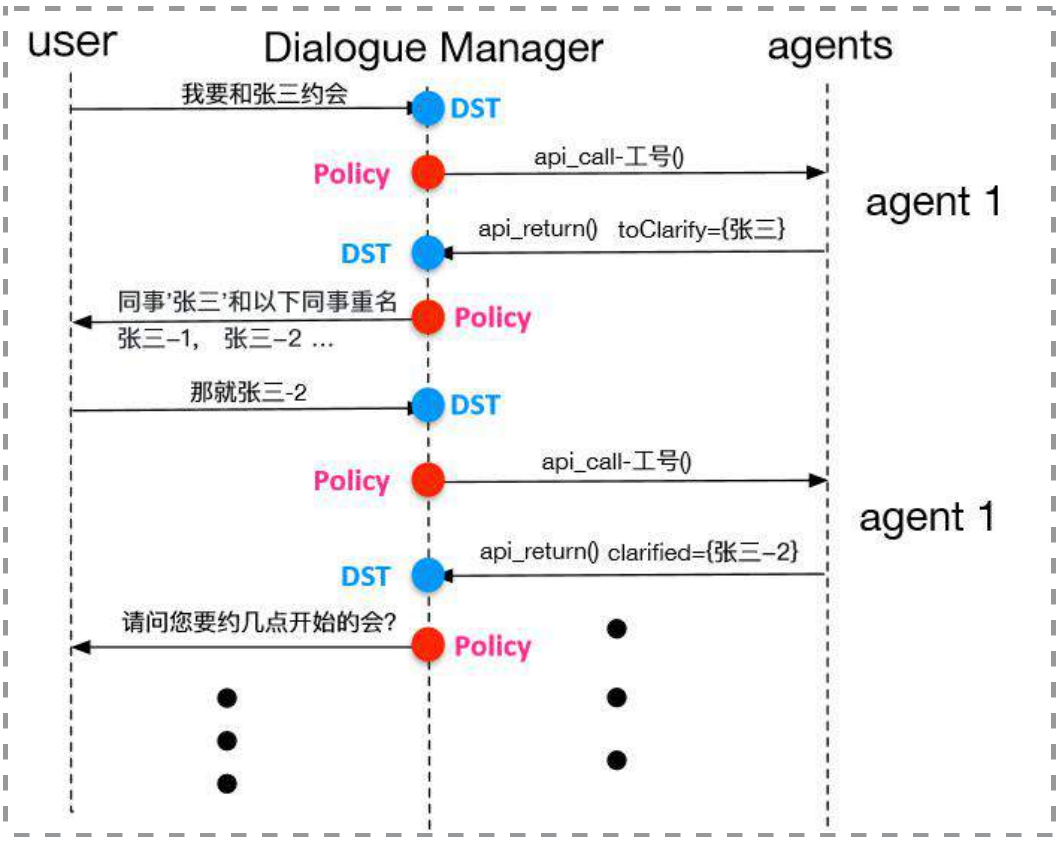


图 18：约会议的追踪过程

如上图所示，一共有两轮对话，每一轮有两次对话追踪。

DST 模型的输入是上一轮对话的状态、上一轮系统行为、当前轮用户 utterance、API 返回，输出是当前轮的对话状态。

DataFunTalk

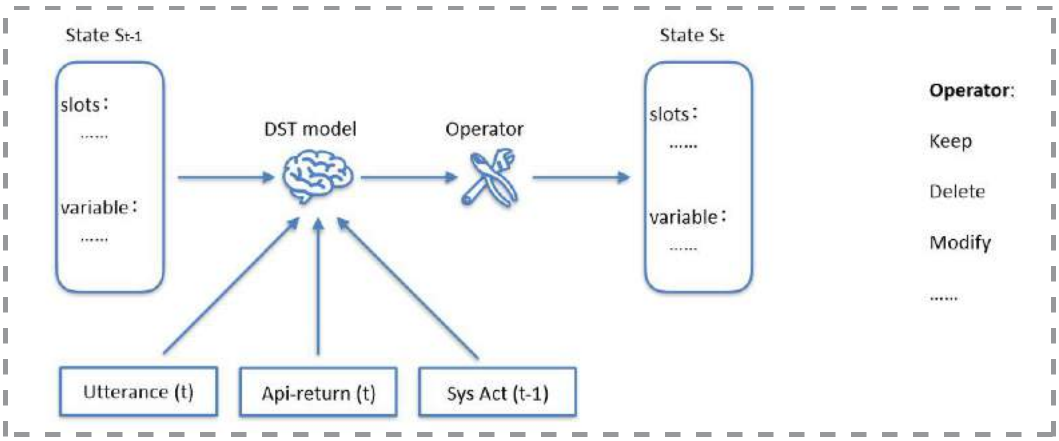


图 19：DST 的状态跟踪流程

这个模型的核心是 operator。在“约会议”场景中，工号的 slot value 一直在变化，开会时间也是不可枚举，如果把 slot value 加入到模型中训练，在预测的时候就会遇到未知的 slot value，此时模型效果就变差。operator 的操作对象是 slot 本身，它只关心信息的流转，而不关心具体 value 是什么，它摆脱了对 value 具体值的依赖，因此具备更强的适应能力。

2.5 Policy: A2C-ER with TaskFlow bootstrapping

下图表示的是增强学习的数据收集和训练的过程。首先，用户输入 user act(t)，进入一个对话状态 state(t)，经过 feature generator 模块，得到 b(t)，接着把它输入到 policy 网络获得 act(t)，这个 act 会和对话状态一起输入到一个 Reward Evaluator 模型进行打分，得到 reward(t)，然后进入下一个对话状态 state(t+1)，这时候会得到一个 experience，它包括：

<state(t),act(t),state(t+1),reward(t)>

四元组，然后通过 off-policy 的方式进行训练，而这里收集到的样本分布和 policy 真正输出的样本分布存在偏差，可以利用重要性采样（Importance Sampling）的方法进行修正，这样能训出一个比较好的 policy 模型。

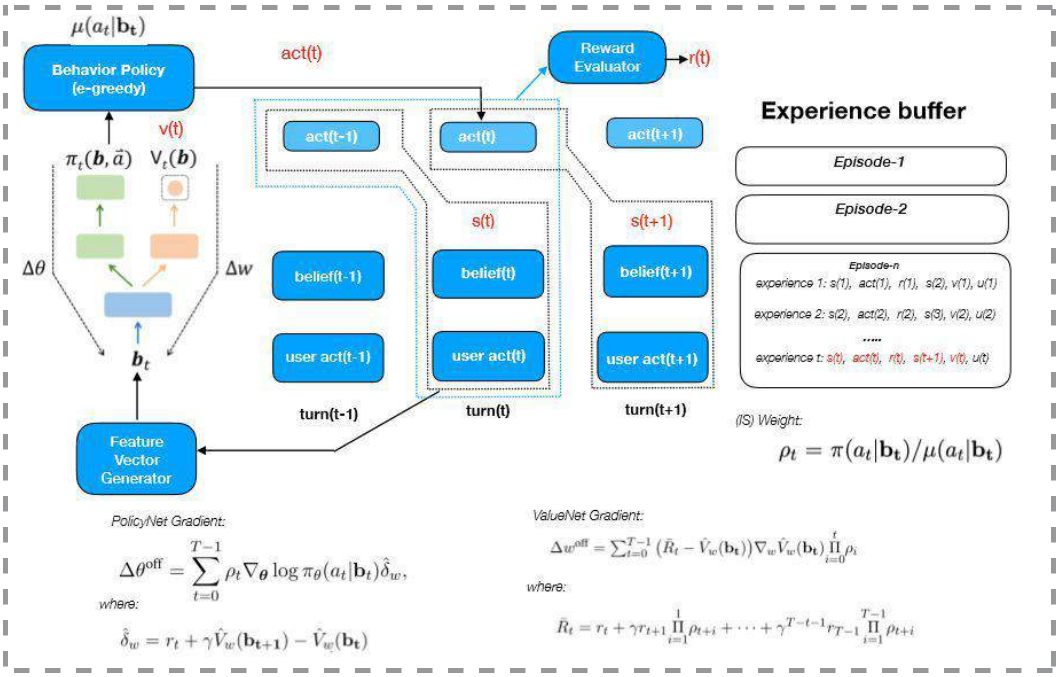


图 20：Policy 的学习过程

3. 用户模拟器

3.1 Simulation System

DataFunTalk

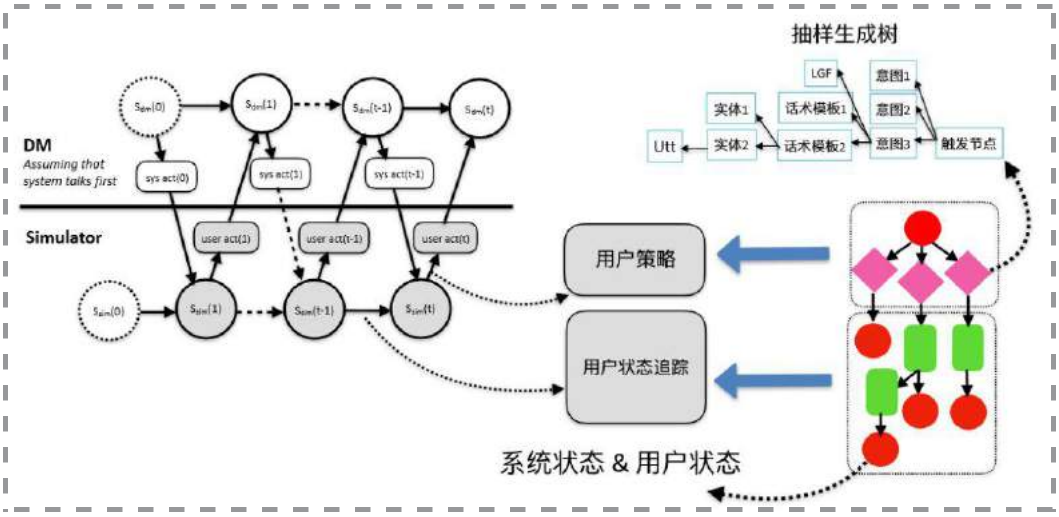


图 21：Simulation System 的实现原理

从上面所说的 DM 模型的训练过程，我们可以看到用户模拟器起到非常重要的作用，这里我们介绍一下用户模拟器的实现原理。

我们的用户模拟器是以 Taskflow 为根基构建的，触发节点展开后形成的抽样树对应用户策略，函数节点+回复节点的串联对应用户状态管理，这是实现用户模拟器的基础。

它由 3 部分组成：User State Tracker、User Policy 和 User Model。其中 User Model 可以针对不同的任务设定不同的参数，比如：生成对话数据，对话评测，它们的 Goal 和 Profile 都可以不一样，这样既保证是一套统一的建模框架，同时又保证了系统的灵活性。

DataFunTalk

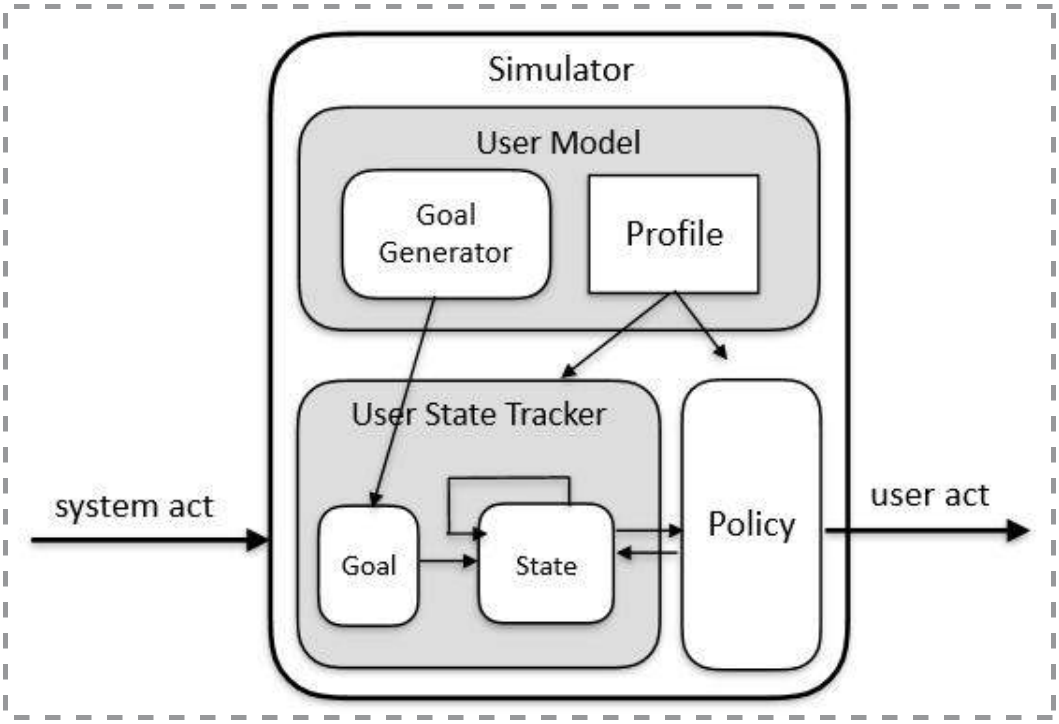


图 22：User Simulator - 实现框架

具体的应用场景如下：

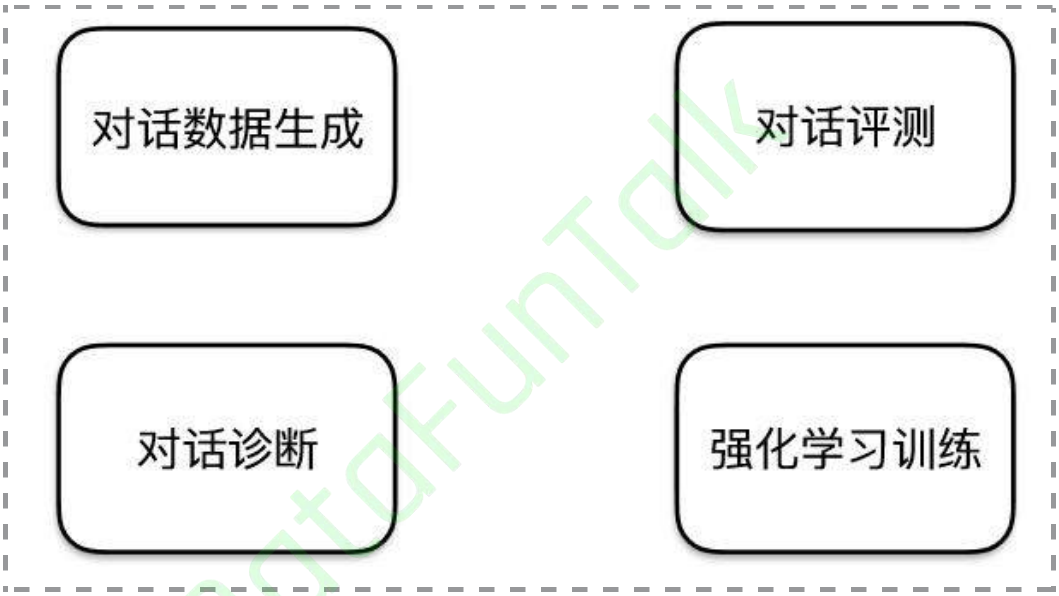


图 23：User Simulator - 应用

3.2 User Simulator - 对话诊断

这是一个公积金查询的对话流，它一共有 175 条路径，如果要去覆盖需要输入 1000 次以上，这个过程耗时耗力。

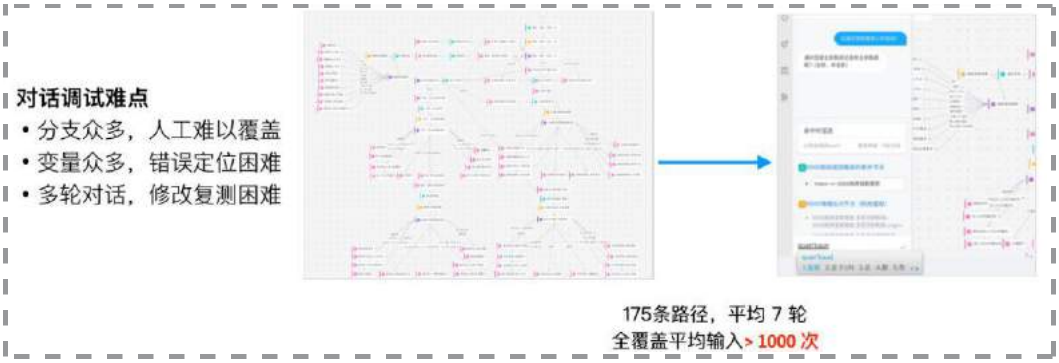


图 24：对话系统的对话诊断难度

为了解决这个问题，我们提出了利用一个机器人诊断另外一个机器人的想法，具体的实现架构如下图所示：

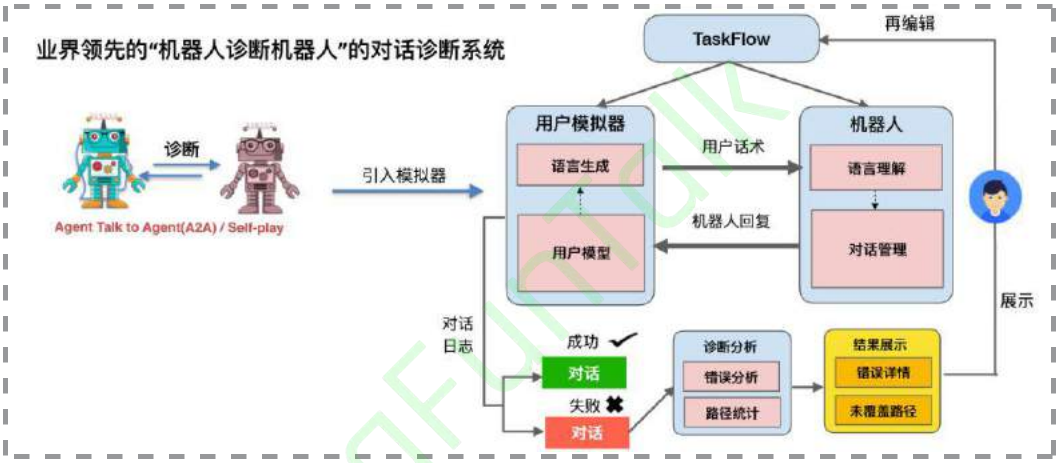


图 25：对话诊断机器人架构

它的基本逻辑是，首先有一个对话系统，它加载的是 Taskflow，对话系统与用户模拟器交互，产生对话日志，这里面有成功的 session 也有失败的 session，如果是失败的，会把它送到一个诊断分析的模块，进行错误分析和路径统计，得到错误详情及问题产生原因，反馈给业务人员，业务人员根据提示修正 Taskflow，然后重新诊断，以此往复直到所有的问题都解决。

4. 总结

总结如上，这是本次分享的主要内容，谢谢大家！

DataFunTalk

UC 国际信息流推荐中的多语言内容理解

嘉宾：汪昆　编辑：Hoh Xil

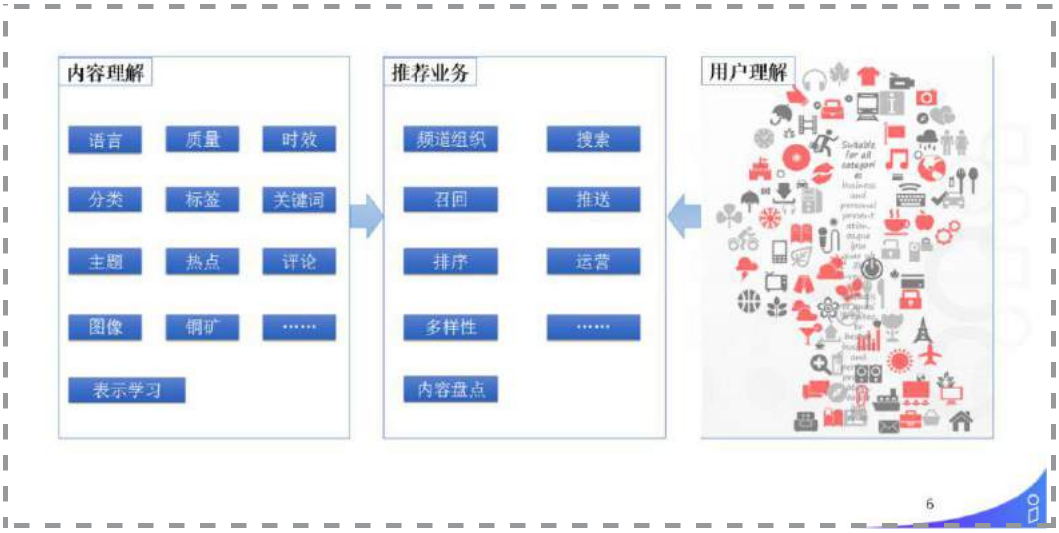
导读：本次分享的题目是 UC 国际信息流推荐中的多语言内容理解，讲一下我们在国际信流推荐场景下的多语言内容理解方面的一些工作和思考，主要分为：

- 多语言内容理解的需求和挑战
- 多语言内容结构化信号建设方法
- 总结

多语言内容理解的需求和挑战

1. 结构化是推荐的基础

DataFunTalk



结构化是推荐的基础，对于大多数做推荐的同学应该会比较有感受。可以把推荐系统分为三部分：内容理解和用户理解两个大的离线模块；在线的推荐业务分为召回、排序、多样性等，主要利用离线的 NLP 结构化信号以及离线的用户理解信号来完成推荐的工作，本文主要介绍内容理解的部分。

对于内容理解，我们面临的是多个国家、多种语言的新闻推荐场景：

从 item 的维度，我们会做语言的识别、质量的控制、时效的控制（由于我们是新闻推荐的场景，跟商品以及其他物品的推荐不太一样，对于时效的控制会比较严）。

从用户兴趣的维度，我们会做分类、标签、关键词以及主题等内容信号的建设。

热点挖掘，因为我们是新闻客户端，所以会做多源的新闻热点挖掘。

互动评论，由于我们还有一些用户互动的评论，所以会做评论的分析和反作弊。

图像，因为我们的新闻是图文并茂的，并且视频会带有图像，所以在图像这块也会做一些图像的表征学习或者特征的挖掘。

铜矿，我们不同于其他新闻客户端的一点是，我们的新闻推荐主要是在浏览器中，除了用户在信息流场景下的消费行为，还会利用到用户在浏览器中的其他的一些浏览行为、搜索行为等等的一些行为，我们把它称之为铜矿，来挖掘用户相关的一些兴趣。

表示学习，因为我们面临的语言比较多，所以我们还会用表示学习的方式来做很多的工作。

2. 多语言的业务需求

UC 国际化信息流推荐面临的多语言的业务需求：

多个国家：已在印度和印尼等多个国家上线十多个语言的信息流推荐服务；

多种语言：仅在印度已经涵盖印地语、英语、泰米尔语、泰卢固语、马拉地语、古吉拉特语等多个小语种，对 NLP 而言存在天然的多语言诉求；

DataFunTalk

多种样式：包含图文、短视频、小视频、GIF 和 Memes 等多种形式的新闻推荐，除了多语言外，还提出了多模态的诉求。

3. 技术难点

由此带来的挑战，包括：

语言多：需要覆盖十多个小语种；

任务多：基本的词法分析，每个语言的分类、标签等多个任务，都需要建设；

样本少：不是每个语言都有丰富的标注资源，对每个语言都需要进行标注，标注样本成本高，建设周期长；

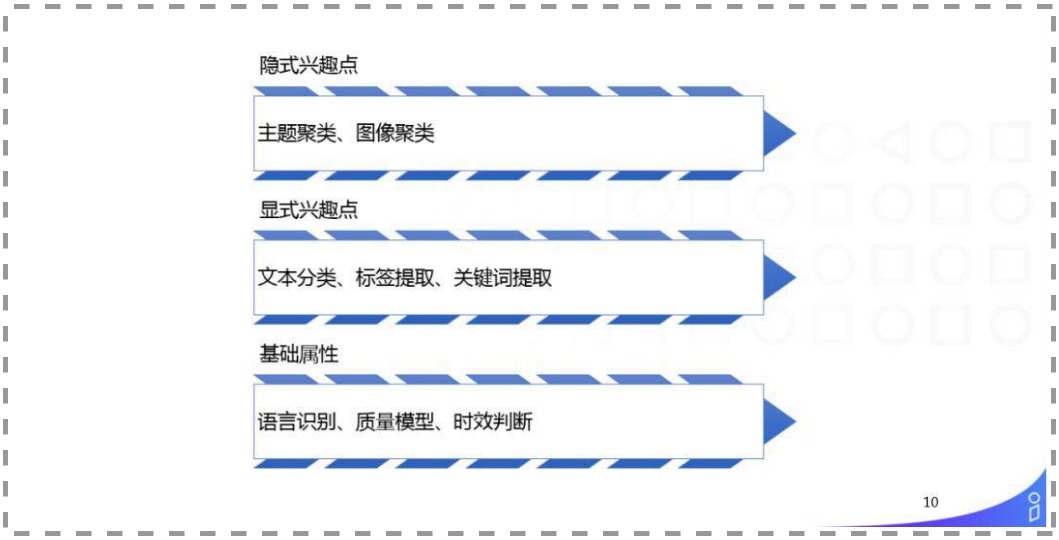
人才少：在国内懂小语种的人比较少，算法迭代分析困难。

所以，在我们的场景下，这是一个典型的资源匮乏型（Low Resource）的 NLP 问题。

多语言内容结构化信号建设方法

1. 内容结构化的发展路径

① 内容结构化信号



从内容结构化信号整体上来讲，我们从底向上做了很多不同类型的内容结构化信号：

基础属性：因为我们是多语言的场景，首先要做语言识别，然后做质量的控制以及时效的判断；

显式兴趣点：从用户显式的兴趣角度来讲，我们要做文本的分类、标签提取、关键词提取；

隐式兴趣点：由于用户的显式兴趣点往往受制于样本的规模、算法的精度，所以我们还会做一些用户隐式兴趣点的挖掘，如：主题聚类、图像聚类、用户表征或者基于表示学习的隐式兴趣挖掘。

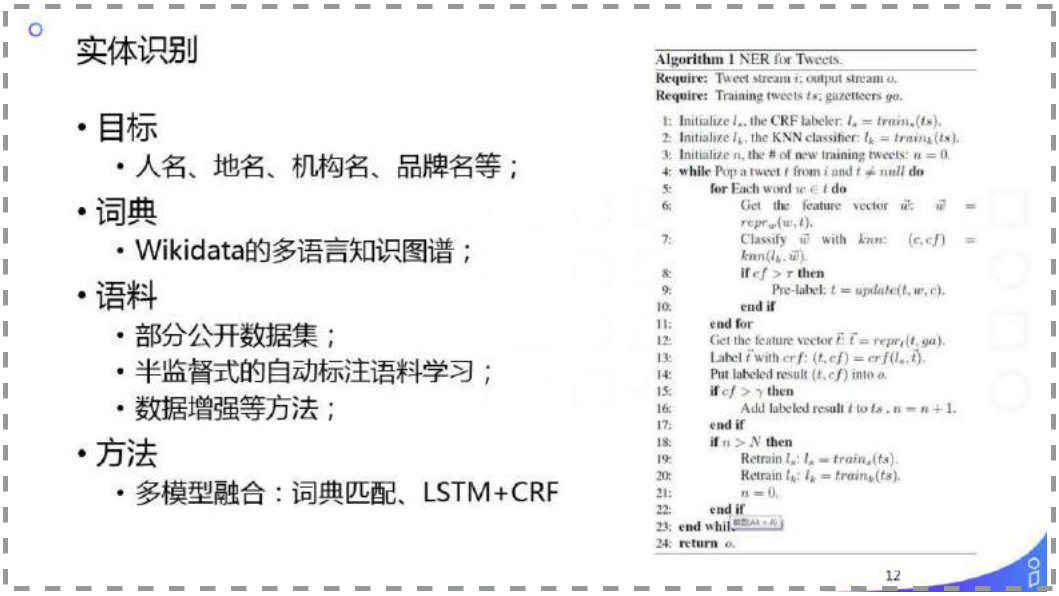
Treebank	Mode	Words	Sents	UPOS	XPOS	UFeats
Arabic-PADT	Raw text	94.60%	82.10%	90.20%	83.80%	83.90%
Chinese-GSD	Raw text	90.30%	98.80%	84.20%	84.00%	89.10%
English-EWT	Raw text	99.10%	77.10%	93.50%	92.90%	94.50%
English-GUM	Raw text	99.70%	80.70%	93.20%	92.70%	93.90%
English-LinES	Raw text	99.90%	87.20%	94.60%	92.40%	94.80%
English-ParTUT	Raw text	99.50%	99.00%	93.90%	93.50%	93.50%
Hindi-HDTB	Raw text	100.00%	98.20%	95.80%	94.80%	90.30%
Indonesian-GSD	Raw text	100.00%	94.10%	93.00%	92.20%	93.90%
Marathi-UFAL	Raw text	90.10%	90.70%	71.30%	-	60.60%
Russian-GSD	Raw text	99.90%	96.20%	94.40%	94.20%	84.30%
Russian-SynTagRus	Raw text	99.70%	98.80%	97.90%	-	93.60%
Russian-Taiga	Raw text	98.00%	87.20%	86.40%	98.00%	75.80%
Tamil-TTB	Raw text	95.00%	100.00%	81.80%	76.90%	81.00%
Telugu-MTG	Raw text	99.80%	97.60%	90.50%	90.50%	98.70%
Vietnamese-VTB	Raw text	85.40%	93.70%	76.20%	74.10%	85.10%

② 基础词法分析——去语言化

因为我们面临了 10 多种语言，多种不同的任务，对后续的算法，如：分类、标签，以及下游的推荐和用户画像来讲，我们期望尽量把语言相关的东西在 NLP 中解决，下游的一系列任务可以去语言化，算法能够通用。所以，我们尽量在基础的词法分析阶段就把语言的鸿沟消除掉。

我们面临的难点是覆盖语言种类比较多，标注标准比较多。所以我们尽量采用的是 Google 开源的 Universal Dependency 数据库 ,它覆盖了 70 多种语言，100 多个 Treebank。我们期望在所有的语言和任务上采用同样的标准，在词法分析阶段都处理成标准的 schema 以后，下游即使写规则，也不需要换标签集。我们在开源工具的基础上，做了一系列的迭代工作，覆盖了 10 多种语言的词法分析。

③ 实体识别



我们会做多语言的实体识别，主要利用的是 Wikidata 多语言知识图谱，来挖掘高质量的词典，同时使用半监督或者数据增强的方式标注一部分资源，然后还有一部分人工校对的工作在里面。用的算法和中英文上的并没有太大的区别，主要是 LSTM + CRF 序列标注算法。但是我们在资源方面投入了很大的精力，以数据增强为例，我们可以从学术圈或者公开数据集上拿到很多各个语言的 NER 标注语料，结合我们挖掘出来的多语言的知识图谱，可以直接做数据增强的一些办法（在图片和视觉方面，做数据增强可能很容易，但是在 NLP 角度做数据增强并不是很容易）。比如“巴黎是法国的首都”，如果把“巴黎”换成“伦敦”对于人来说是不可理解的，是错的，但是对于机器训练 NER 来说这是没有问题的，所以我们会用规则或者知识图谱来限制这种数据增强的规模，取得了比较明显的效果。

④ 多语言分类发展阶段

我们在多种语言上都经历了这样的分类发展阶段：

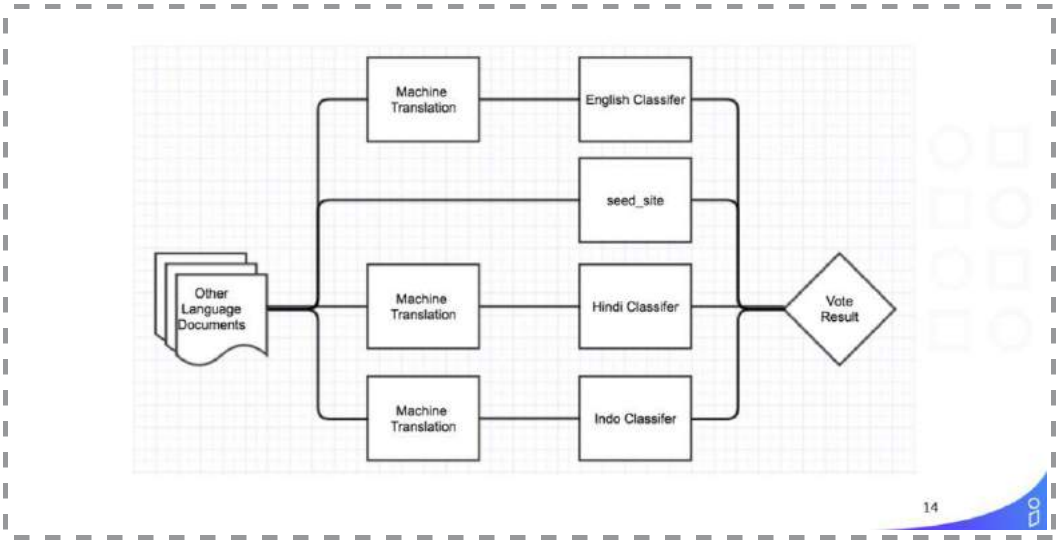
起步：我们刚开始做国际化的时候，在做分类时完全没有标注语料，我主要是通过种子源、种子词，人工总结一些规则或者绑定（一些成熟的站点，我们会进行绑定）。

发展：到了发展阶段，我们累积了一些样本，会去做迁移学习，包括样本的迁移和特征的迁移。

深入：从去年开始，包括 ELMo、BERT、GPT 等（先做预训练，再做后面的一些迁移学习）在 NLP 中成为了一个基本范式，在这个阶段，我们开始做多语言的表示和小样本的学习，能够提升分类的精度。

简单举几个例子：

基于翻译模型的分类样本迁移

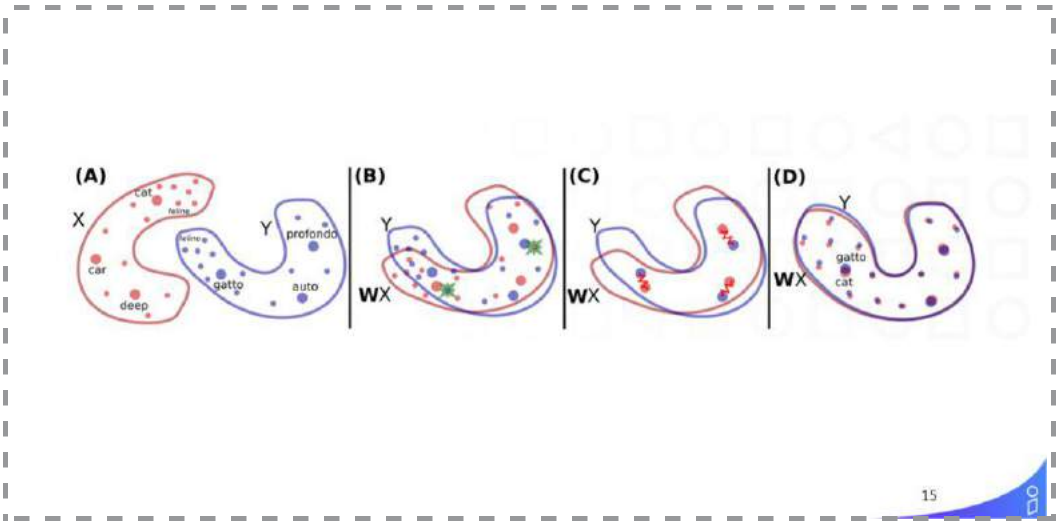


我们所有的分类都是多种策略并行的系统，没有一个语言是单纯的系统或者策略。最开始我们在英文、Hindi、Indo 上投入了很多资源，所以结构化的信号建设基本成熟，因为在后面开展一些小语种的时候，会采用多种的办法做样本迁移和特征的迁移。

比如样本迁移，我们会采用图中的办法：把小语种翻译为三个大语种，用大语种的 model 做分类。

基于词向量对齐的分类特征迁移

DataFunTalk

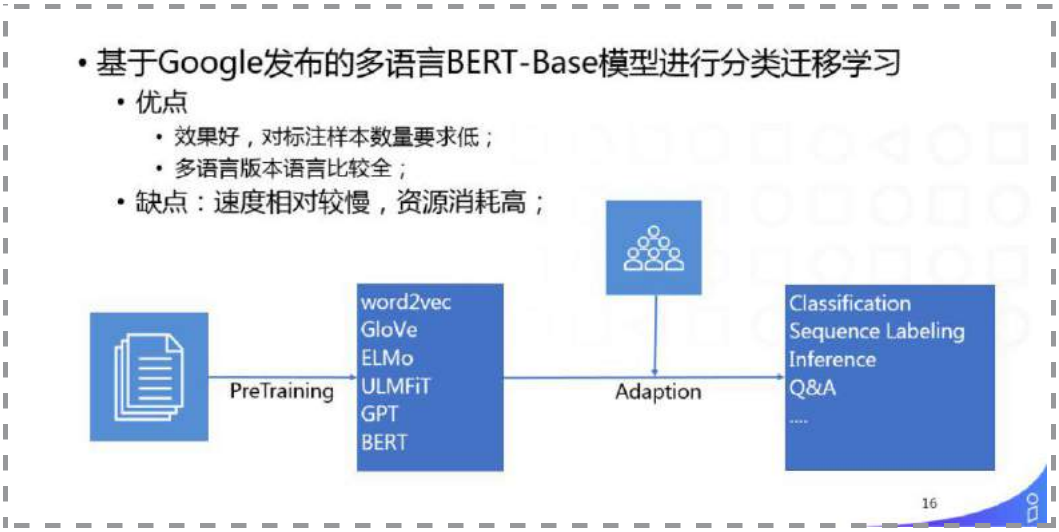


这是 Facebook 提出的多源词向量对齐的办法 MUSE，MUSE 的基本思想是用单语言的训练语料去训练每一个单语种的词向量，然后再加上一个双语词典，可以把不同语言的词向量映射到同一个空间，他们的目标是映射到同一个空间时，表示相同语义的词距离更近。这种方法在一些有共性的语言上，比如印尼语和英语，效果会比较好，但是在语言差异较大时，效果会弱一点，整体上这种方法还是给我们带来了很多可选的样本。

去年 Facebook 又提出了基于句向量对齐的 LASER，由于需要双语机器翻译的语料，应用起来的话限制条件会比较多。

基于 BERT 的迁移文本分类

DataFunTalk



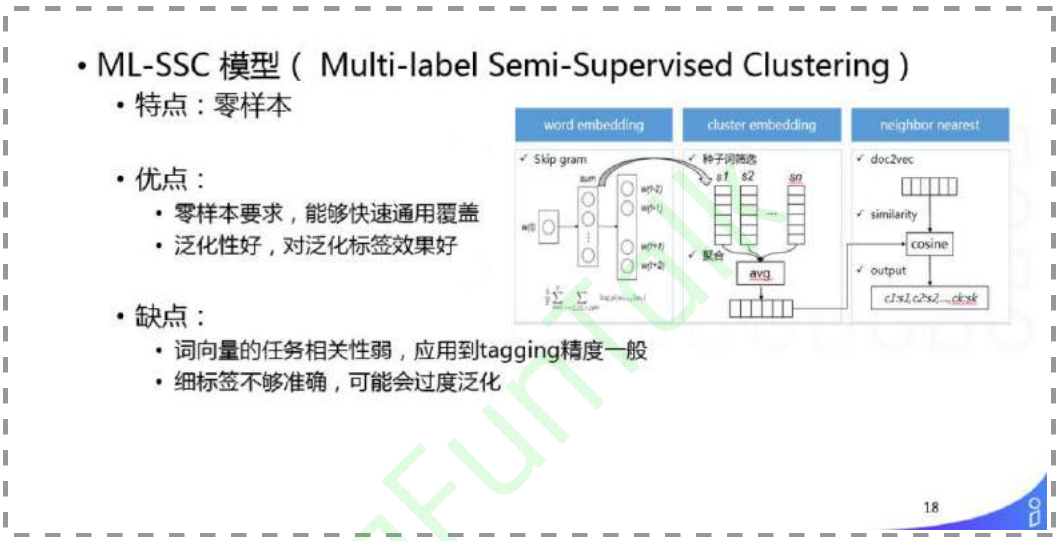
从去年开始，先用大规模的单一语料做 PreTraining，使用 ELMo、GPT 或者 BERT，得到一个单语的文本表示，后面再接一些你自己的任务做 Adaption，已经成了 NLP 的一个基本范式。我们也在很多的任务（如：分类和标签）中采用这种办法，其优点是对标注样本数量要求低，在一些小样本上就可以得到很好的效果，但是也存在着缺点就是速度相对较慢，资源消耗比较高，因为我们的语种比较多，所以还会做一些半监督和知识蒸馏相关的一些探索性工作。

⑤ 语义标签

DataFunTalk

除了分类以外，我们还会做标签，我们的总体思想是一样的，先基于专家经验去累积样本，累积一定样本之后会做一些大规模的有监督学习模型。

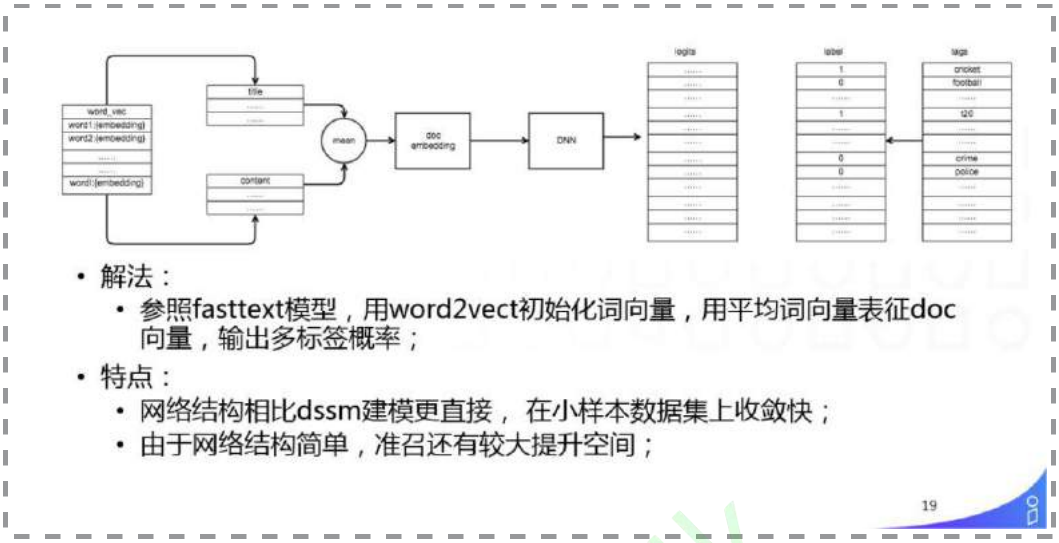
弱样本依赖模型



举个简单例子，我们如何做弱样本的依赖模型，它的特点是零样本。如果定义一些语义标签之后，通过 word embedding 先相关很相似的词，然后把

这些词做聚类，聚类以后才作为一个语义标签簇的表达，之后直接和 doc 的 embedding 做相似性计算，我们就可以不需要训练样本，完全单语的语料就可以给文章打标签，这种方法对粗粒度的标签效果还可以，但是对细粒度的效果还不行，只能保证一些比较粗的标签的召回，提供一些基础的样本，供后续的人活着算法做二次校验。

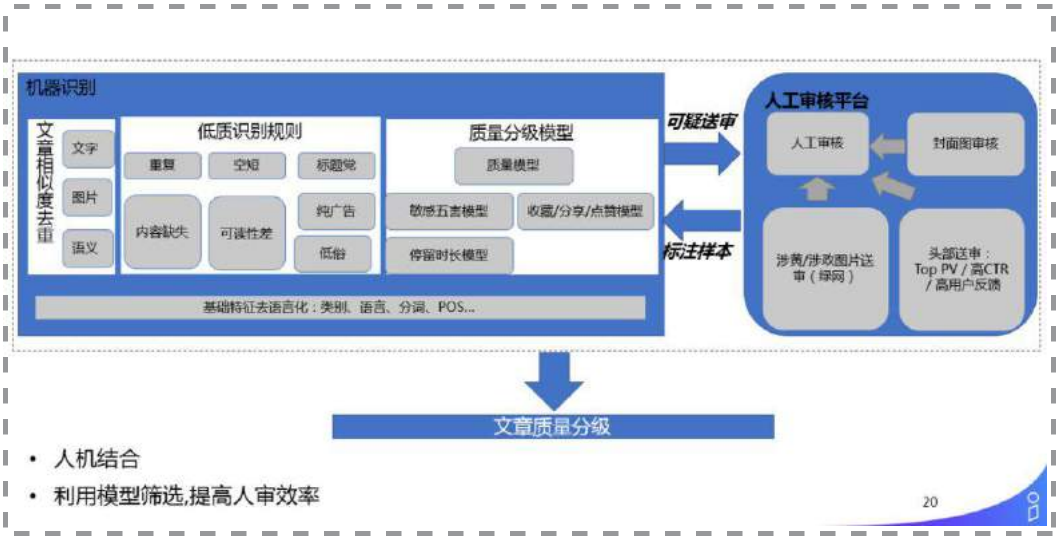
大样本监督模型 ML-DNN



通过各种策略挖掘和人工总结大量样本之后，我们可以使用一些大样本的有监督模型，我们的做法比较简单：参照 Fasttext 模型，用 word2vect 初始化词向量，用平均词向量表征 doc 向量，输出多标签概率，就是把刚刚的做法直接迁到有监督的框架下。它的优势是模型比较简单，结合之前做的一些工作，可以快速的覆盖头部的语义标签。

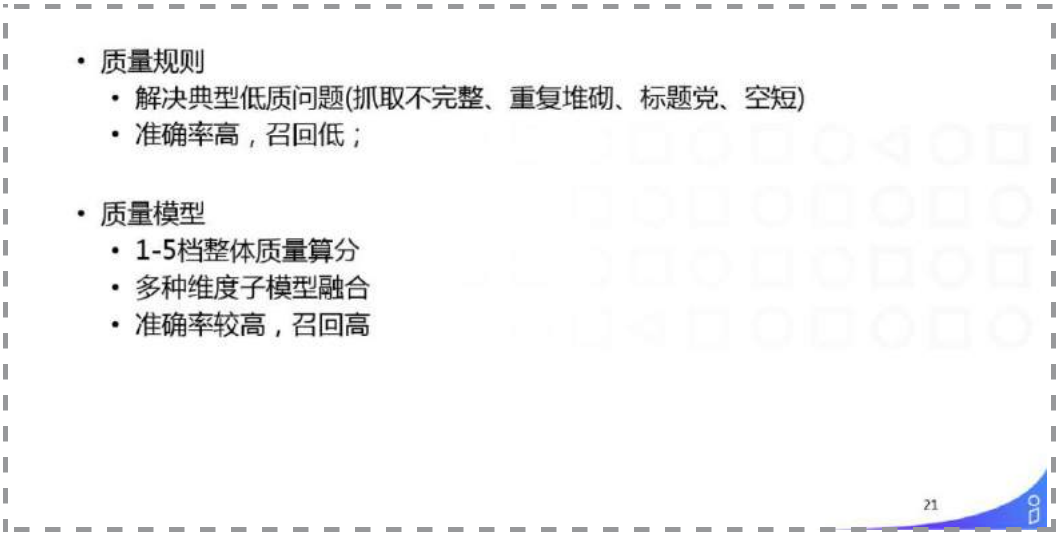
2. 内容质量模型

刚刚讲的是分类和标签，这种结构化的信号，我们再来讲下内容质量：



因为质量是新闻推荐的一个基本红线，在国内可能体会更深，像今日头条、快手等有段时间就因为一些质量的问题被下架了，我们在国际化时，也会遇到这样的问题。

我们的质量模型，基本还是一个人机结合的框架。主要通过一些低质的规则，能够把一些高危的 item 直接检测出来，直接干掉；我们还有一个质量的分级模型，会从多个维度做多个子模型，然后再有一个大的模型来融合这些子模型，质量平台输出的一些不太确认的结果会送到人工审核平台去审核，人工审核平台审核后的结果又会返回到质量模型做 training。在这个过程中，整个框架是去语言化的，我们输入的是 NLP 基础结构化的信号（类别、语言、分词、POS 等），后面的算法不会对语言做特殊的处理。



对于质量规则，我们的准确率会比较高，召回低一点，主要是为了保证检测出绝对低质的东西。

对于质量模型，我们主要分为 5 挡，然后做多维度子模型的融合。

3. 图像信息的应用

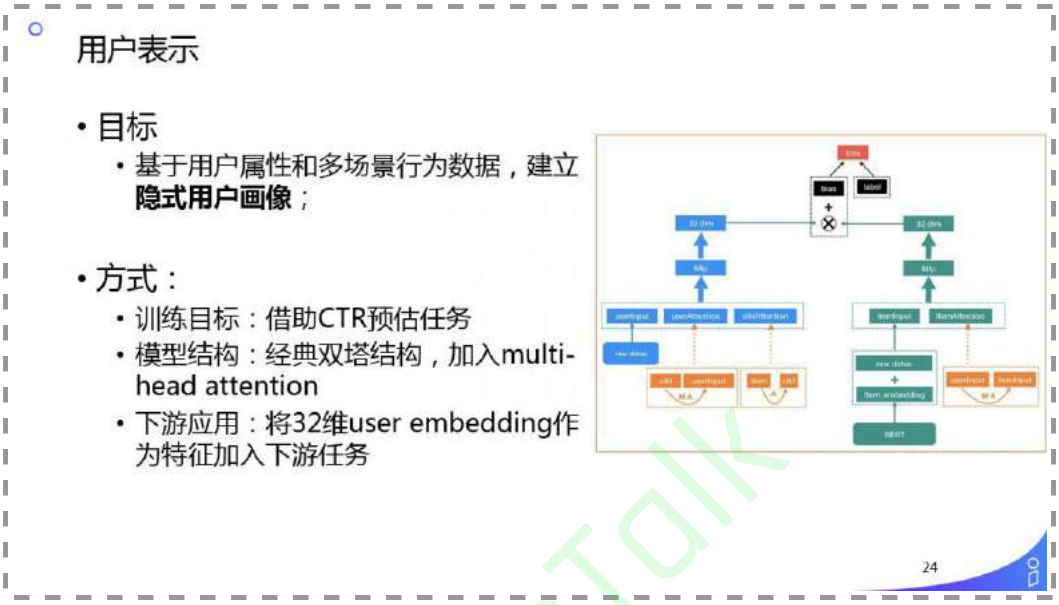
刚刚讲的都是一些用户显式的结构化信号，比如语言、分类、标签等人可理解的一些显式信号，显式信号在不同语言的发展进度和阶段是不一样的，为了让一些小语种追赶上大语种的节奏，我们还会做一些不依赖于 NLP 基础建设的一些用户显式的兴趣挖掘，就是内容结构化上的一些隐式的信号。比如图像方面的一些工作：因为语言在不同国家不同语种上会有资源的问题，对于图像，除了上面的一些文本，其它方面是没有太大语言鸿沟的，所以我们会做一些图像方面的工作，比如图像聚类。

具体的做法是：基于开源图像表征模型 ResNet 和 VGG 抽取图像表征，使用 K-means 聚类选取适当合适的聚类簇固定下来，然后对于新入库图像基于固定下来的簇进行打标。这样会得到什么结果呢？

这是从我们库中找的一些样本，对印度的一些封面图做了一些基于这样表示的聚类，最左边是聚类的编号，从这些样本中可以看到一些共性，比如第 23

个 topic 都是关于狗的，24 基本都是钱等等，我们可以直接把这些信号用到 rank 或者召回中，比如印度人比较喜欢的我们的一个场景叫 Memes（用户发一张图，上面会加一些文字，其它的什么都没有），这时用图像信息做召回效果会比较好。我们通过这种隐式的聚类的编号，跟分类的一些信号做交叉 mapping，发现这里很多类别直接可以对到二级分类，并且准确度高，这样就可以快速的建立很多分类的样本。

4. 表示学习在内容理解和用户理解中的应用



刚刚讲的都是内容理解方面的内容，接下来简单介绍下用户表示。我们在做内容结构化信号时，用户显式可以理解的信号，在不同的语种上，特别是小语种上，我们的迭代速度以及精度跟不上推荐的诉求，这时我们需要用一些隐式的信号，提升推荐的效果。我们的想法是基于预训练在具体任务做

fine-tuning 这种范式已经在 NLP 和图像领域验证过了，所以我们在推荐领域做了这样的尝试：

我们利用用户的消费数据直接建立双塔，类似于 CTR 预估，一边是 user，一边是 item，item 直接用 BERT 的 embedding 作为初始输入，然后两边做双塔结构，这样做的好处是可以引用更多的用户在其他场景下的一些行为，比如用户在浏览器中的一些搜索 query，这时再离线做双塔，user 侧一直往上走，然后 item 侧一直往上走，最后分别得到 user 和 item 的隐含表达，然后再做相似度计算，预估 ctr，但是这不是我们的目的，我们的目的是直接用 user 的表达给下游用。这样在一些稀疏的场景效果提升明显，比如 Push。

总结

简单总结下本次分享的内容：

- 多语言内容理解业务需求广，单一语言的经验不可完全复用，所以我们要去语言化，在基本的词法分析阶段把语言处理成标准的 schema，除了样本之外，期望后面所有的算法都跟语言没有太大的关联；

- 迁移学习是我们解决多语言内容理解问题的核心抓手，因为我们天然的会遇到样本和特征的问题，所以我们会做样本和特征等的迁移；

- 表示学习特别是基于预训练的表示学习为迁移学习，特别是我们的推荐场景，带来了新的无限可能，我们在 NLP 的基础结构化信号中有一些成功的应用，并且在用户表示上也有一些成功的范式。

本次的分享就到这里，谢谢大家。

DataFunTalk

用人工智能技术打造虚拟生命

嘉宾：邵浩　编辑：Jane Zhang

导读：近年来，作为人工智能时代的入口级产品，聊天机器人得到了快速的发展，然而从用户体验上来看，大多数产品都远低于大众预期。本次分享，将分析聊天机器人产品所面临的技术和落地难题，引出聊天机器人的下一代范式-虚拟生命，并详细阐述如何使用人工智能技术，包括语音、图像、自然语言处理以及知识图谱技术等，打造具有认知智能的虚拟生命。

分为 4 个部分：

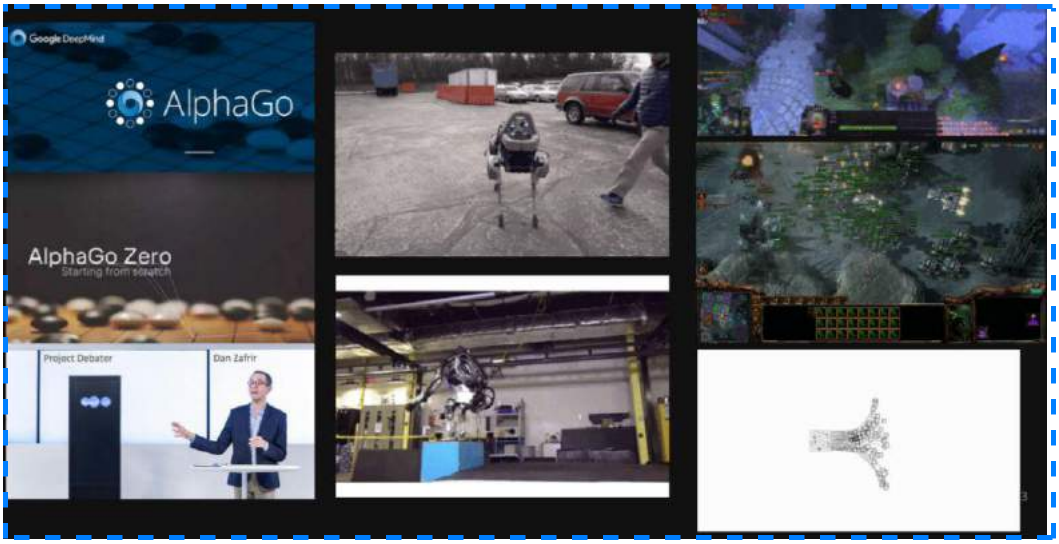
- 聊天机器人困境及破局
- 虚拟生命核心技术及落地
- 工程实践中的其他问题
- 结束语

聊天机器人困境及破局

1. 分享两个观点

1.1 人工智能技术真的很牛

DataFunTalk



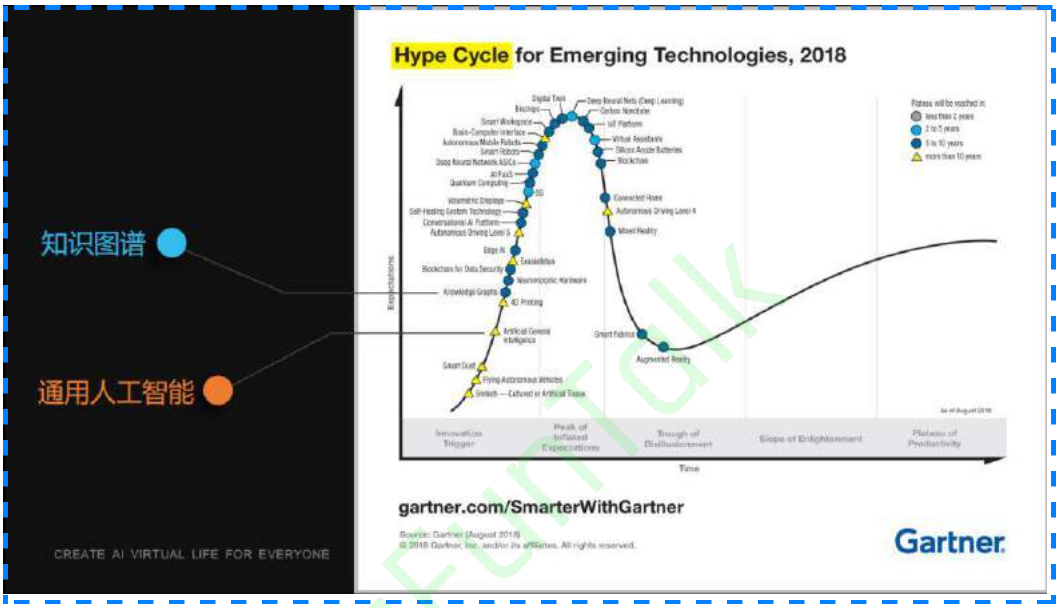
左上角列出的是 DeepMind 出品的 AlphaGo，它打败了世界最顶尖的选手，后续又出了很多新的版本，又打败了柯洁；左下角是 IBM 新出的辩论机器人，辩论机器人可以和人类辩手针对某个问题进行辩论，战绩也非常不错；中间是大家熟悉的波士顿动力机器人，可以完成各种复杂动作；右边，是把人工智能技术用在游戏中，比如星际争霸二。所以大家就会有种感觉，人工智能技术真的是上天入地，无所不能。既然人工智能技术和深度学习技术已经如此优秀了，是不是所有问题都可以解决呢？答案当然是否定的。比如 AlphaGo 可以下围棋，但是不能端茶倒水。有人开玩笑说，战胜人工智能最好的办法就是把它的插头拔掉。

1.2 聊天机器人好傻

举例来说，前段时间亚马逊爆出“隐私门“ -- 亚马逊音箱 Echo 在半夜突然笑了一声，把用户吓得半死。也就是说，Echo 在不停的监听用户的语音。市

场上还可以看到其他的智能音箱产品，如小米的“小爱同学”，我们自己做的“公子小白”，“叮咚音箱”，“天猫精灵”，还有百度带屏的“小度在家”，还有很多叫不上名字的。市场上充斥着大量的机器人产品，但无一例外都是“不智能”的，尤其是在多轮对话中。虽然小冰已经做到了第六代的情感引擎技术，但我们在与小冰聊天的过程中，还会发现很多矛盾的、上下文不一致的点。为什么聊天机器人做不好？这就是我们想要聊的核心。我们期待机器人能够模拟人类对话，在这个过程中，需要什么样的技术积累呢？我们看一下 2018 年最新的 Gartner 技术曲线

2. Gartner 技术曲线



蓝色的点，表示 5-10 年可以达到成熟的技术；黄色的三角形，表示超过 10 年才能成熟的技术。知识图谱技术和通用人工智能技术，是达到机器人和人

类自由对话中必不可少的技术。知识图谱目前处于上升初期，还需要 5-10 年才能成熟；通用人工智能，也处于初步阶段，需要 10-20 年做到成熟。

可以看出，在做产品过程中，技术的发展是不匹配的，远没达到人类需要的期望值。尤其是在做聊天机器人的过程中，需要用到自然语言处理（NLP）技术，目前也没有办法实现真实的自然交互。

3. NLP 为什么这么难

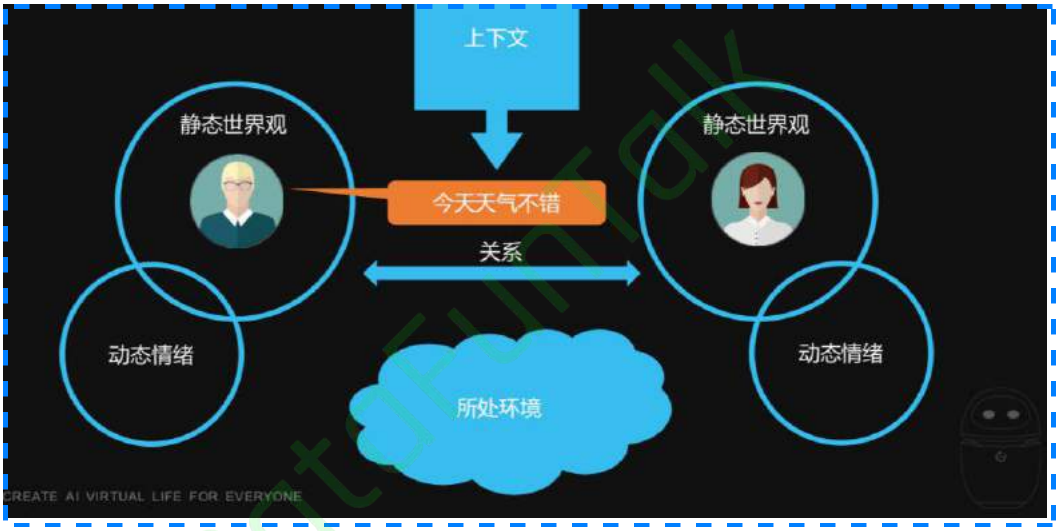
周明老师说过，语言智能是人工智能皇冠上的明珠。人工智能技术在深度学习的加持下，在图像、声学中都取得了重大突破，但在语言智能技术上却有很多阻碍。

举例来说，《绿林俊杰》是一本武侠小说，但百度贴吧里林俊杰粉丝却开始炸锅，说林俊杰最近做了什么事情，为什么被“绿”了？这也是 NLP 较难解决的问题。

DataFunTalk

再如右边的例子，这是一本讲什么的书？还有：男生和女生表白，“我好喜欢你哦，可不可以亲亲你”，女生说“不要脸”，男生说“那我亲嘴好了”，这就是在理解过程中，人类都容易犯的错误，存在很多歧义。这就是为什么 NLP 如此难做的原因。

4. 人是怎么对话的



我们想模拟人类对话的过程，但人类是怎么对话的呢？

上图是我梳理的人类对话要素。比如我是男生，我对一个女生说，“今天天气很不错”，这句话代表的意思和很多因素有关：这句话的上下文是什么；我和这个女生是什么关系；我自己的静态世界观是什么；我现在的情绪；我们现在所处的环境等。如我上午坐电梯的时候，电梯非常挤，对女同事说，“今天天气很不错”，是为了化解尴尬；如果这个女生是我的女朋友，我的意思可能是，“今天天气很不错，要不我们出去玩儿”；如果是分手多年的女友，在大街上偶遇，“今天天气很不错”，就很可能代表“你最近过的好吗”。在不同的场景、上下文中，同样的句子会代表不同的含义。

目前的聊天机器人，是通过文字来猜测人类对话意图的。而人类有强纠错功能，如在回答提问者的问题时，会根据接下来的回复，来纠正对意图的理解偏差，但机器人做不到。机器人回复效果做的最好的，是用 IR（ 信息检索 ）技术，从大量语料库中，检索出比较合适的回复语。这也是为什么单轮对话可以做的很好，多轮对话做的差的原因。

微软亚研的宋老师，从事小冰的研发，有一天问他母亲（ 非 AI 相关从业者 ）说，“如果机器人可以打败人类最顶尖的围棋选手，你觉得机器人厉害吗？”。他妈妈说，“厉害”。又问说，“如果小冰机器人可以和你聊天，那这个技术厉害吗？”妈妈说，“不厉害”。宋老师问为什么，母亲回答“不是每个人都会下围棋，但每个人都会说话啊”。可以看到，人类已经是非常擅长说话了，让机器来模拟人类对话，是非常困难的一件事。

5. 时代 VS 入口

作为一个 80 后，我经历了 4 个时代：

1. IT 时代，我当时有一台 386 电脑，16M 内存，玩 DOS 版的仙剑奇侠传。

2. 互联网时代，有了 Google 搜索，可以和世界各地的人聊天，当时用 MSN 和网上的人练习英文。

3. 移动互联网时代。它的到来，是随着 iPhone 的出现到来的，发生了两件事情的变革：效率的提升和交互方式的改革。效率的提升，使得很多事情在手机端解决，可以实时点餐，实时叫车，这是在互联网时代无法实现的事情。交互方式的改变，我们习惯使用触摸、语音的方式和机器交互。

4. AI 时代，以微软 CEO 纳德拉代表性的一句话说，人工智能时代的入口是语音技术、语音产品。人类是最擅长用语音聊天的，AI 时代，一定会出现一款以语音交互为入口的一款产品。

每个时代都有一个入口：

- IT 时代的入口：PC 电脑
- 互联网时代的入口：浏览器

- 移动互联网时代的入口：触屏手机
- AI 时代的入口：语音交互类产品

每个时代都会诞生一家伟大的公司：

- IT 时代：微软
- 互联网时代：Google
- 移动互联时代：Apple
- AI 时代的伟大公司，现在还不知道。现在都是在争抢入口，无论是天猫精灵还是小米，都是想占据入口

6. 聊天机器人体系

DataFunTalk

下面跟大家聊一聊聊天机器人，从聊天机器人的体系开始说起。包括三个方面：1.框架；2.产品；3.平台。

- 框架，用来实现产品的框架，比如用亚马逊的 alexa 实现了 Echo 音箱；
- 产品，我们能看到或者使用到的终端，比如小冰或者 siri；
- 平台，小冰这款产品是没有硬件形态的，所以需要一个平台来承载这样的产品形态，所以会诞生微信、微博这样的平台，来承载小冰。把这些平台称为聊天机器人的平台。

从产品的分类看，产品分为主动交互和被动交互：

DataFunTalk

- 主动交互是更进一步的智能化产品，可以主动推荐、关怀、提醒，这部分不在我们今天的讨论范围，现在每家都做的不好；

- 被动交互方面 ,是大家见到的主流聊天机器人 ,被动交互是一问一答的形式，又可分为闲聊、任务对话、问答三个部分。

下面聊这几部分产品是如何实现的：

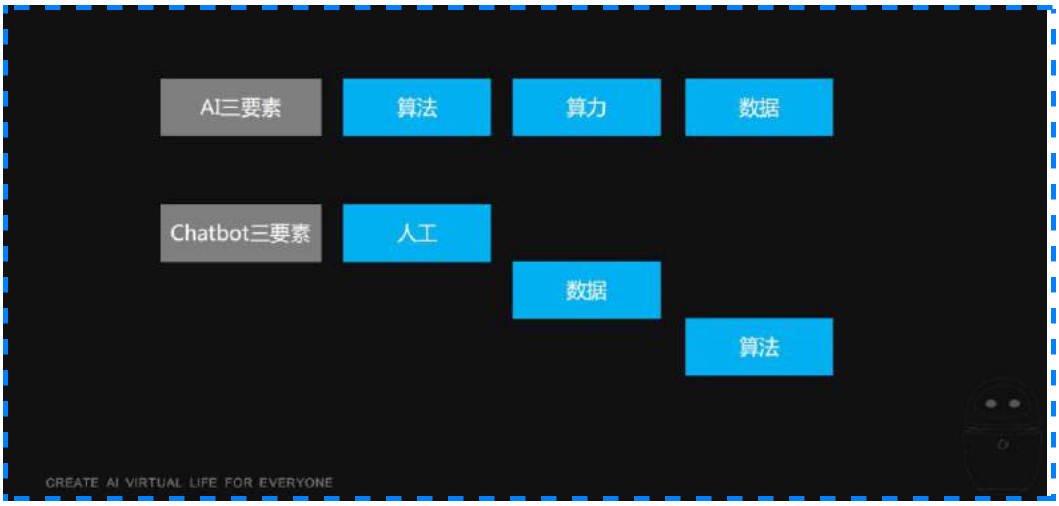
- 闲聊很简单，比如我跟机器人说“今天天气不错”，机器人回复，“今天天气也很不错”。

- 任务对话，比如订票，是为了去完成某个目的，“给我订张票“， “查下某个人的电话号码”，“今天天气怎么样”。

- 问答，是想要得到具体的答案，比如“特朗普的妻子是谁？” “姚明的女儿是谁？” “珠穆朗玛峰有多高？”基本上都是一问一答。

问机器人，“你知道大川的电话号码吗”，机器人回答“知道”，再问“能告诉我大川的电话号码吗？” 机器人答 “可以”。这句话本来是任务型对话的，想要大川的电话号码，但机器人用安全的回复来闲聊，这是无意义的对话，也是聊天机器人中存在的问题。

7. 三要素



人工智能三要素：算法、算力、数据

chatbot 三要素：人工 > 数据 > 算法

人工堆一些规则，用大规模的 QA 数据，就能做到非常好的效果。完全不需要任何算法的加持。也有可能会需要检索的算法，怎样用 index 的方法使数据库的检索更快。但目前还是以人工和数据为主，算法排在最下面的层次。

8. 人工智能的人格化和 IP 化

DataFunTalk

既然 chatbot 的效果这么差，用什么方法来破局呢？怎么让 chatbot 更加有吸引力？这就是人格化和 IP 化。如在 chatbot 中植入智能的 IP ,比如王俊凯，吴亦凡，比如大白，这对于粉丝和非粉丝的吸引力都会提高。

9. 多模态交互

另外，可以通过多模态交互，使得交互更生动，如手势交互、视觉交互、语音交互，还可以看到动作、视频、形象，用这种多模态的方式来吸引用户，这也是可以优化机器人表现的一种方式。这就是为什么在讲聊天机器人的下一代范式，一定是虚拟生命的形态。

DataFunTalk

虚拟生命核心技术及落地

虚拟生命，是用人工智能技术来描述虚拟生命的主要特征，利用多形态和多模态进行交互，具备强大的感知和认知能力，并进一步实现自我认识和自我进化。虚拟生命，最早提出是在 2017 年 CCF 的 paper 中，讲虚拟生命应该具备的能力。

1. 人工生命：人造生命 VS 虚拟生命

DataFunTalk

生命分为两种：生物学上创造的碳基化合物；另一种是计算机技术创造的硅基生命体。这是在生命的定义上所走的两条不同的路。

2. 虚拟生命能力范畴

可以看下虚拟生命应该具备的能力，包括看、听、说、动作，进一步会拥有思考的能力。

3. 虚拟生命技术范畴

分解到细节，有两种范围：

- 基于认知的多模态交互；
- 具有生命感的表达。

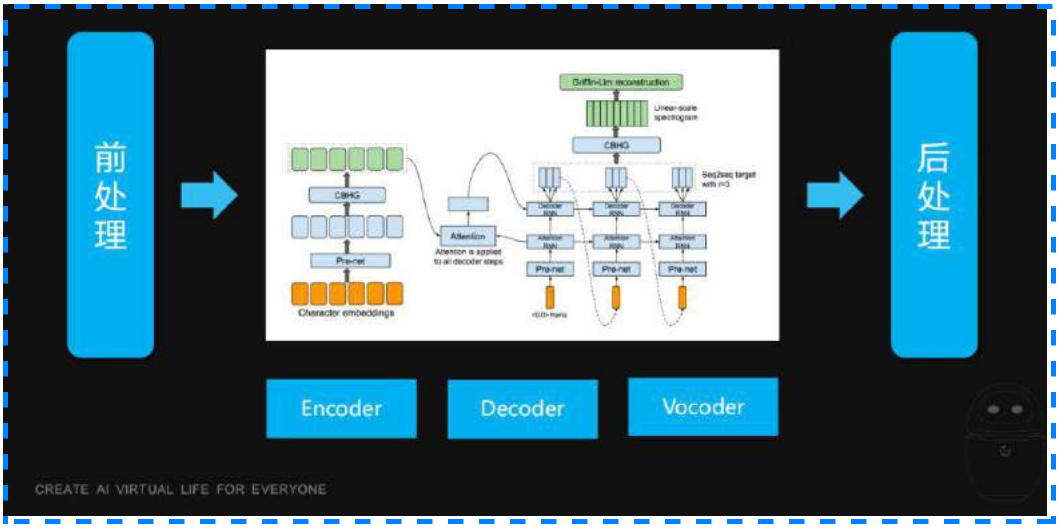
接下来的内容重点讲图像、声学、知识图谱、NLP，以及在歌曲上的技术落地。

3.1 语音技术：耳朵和嘴巴

首先看语音技术。语音技术相当于虚拟生命的耳朵和嘴巴，因为要听人类的语言并转化成文字，这也是大家熟知的语音识别，ASR。国内有多家公司，讯飞是目前较好的一家语音识别公司。语音合成也涌现了很多用深度学习做语

音合成的公司。在深度学习之前，一直在用传统的方法进行语音的合成，随着深度学习方法的出现， 语音合成技术的门槛大大降低，相关资料很多，这里不细讲。

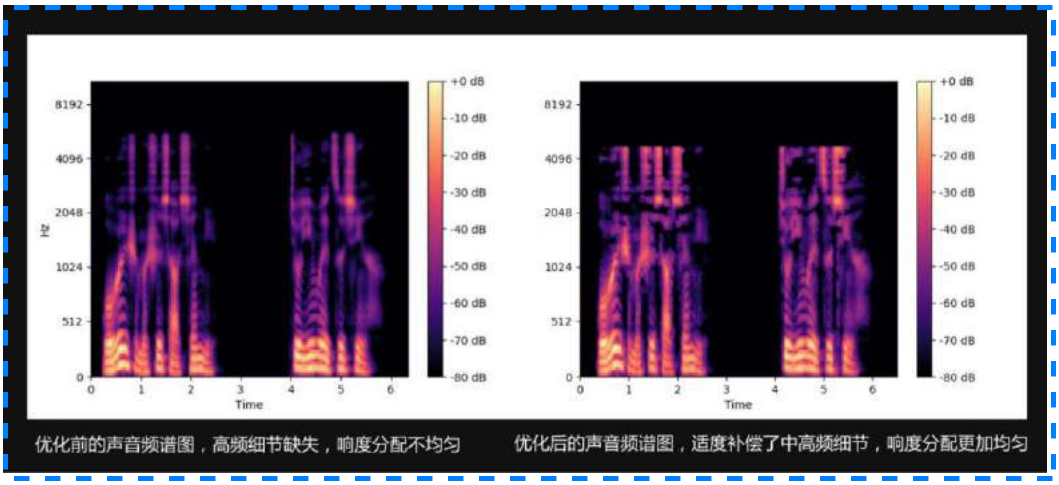
3.1.1 个性化语音合成



个性化语音合成。语音合成分成三个部分，第一部分是前处理，最后一个是后处理。中间的是算法，分两个部分：端到端的深度学习算法，vocoder 技术。图中是 seq2seq 的语音合成技术的算法，称为 Tacotron，现在已经有了第二代。百度的 Deepvoice 也有类似的算法，功能实践中，现在看到的还是以 Tacotron 的技术为主。在语音合成的技术中，虽然能快速搭建框架，但真正在做语音合成的过程中，需要花费 3 个月的时间，才能真正做好。在聊天机器人的实现过程中，对语音合成速度有极高的要求，如对不同的语音秒数，5 秒，10 秒，有不同的要求，如低于 50ms、100ms 的语音合成技术，这就

考验了整个 seq2seq、声码器的处理效率；还有在前处理过程中，如何处理更多的多音字，比如“你好”，这个词中，你读三声（汉语声调有 6 种，升平阳平上声去声，外加轻声和 3 转 2)。你在“你好”中的发音为 2 声。

3.1.2 后处理优化

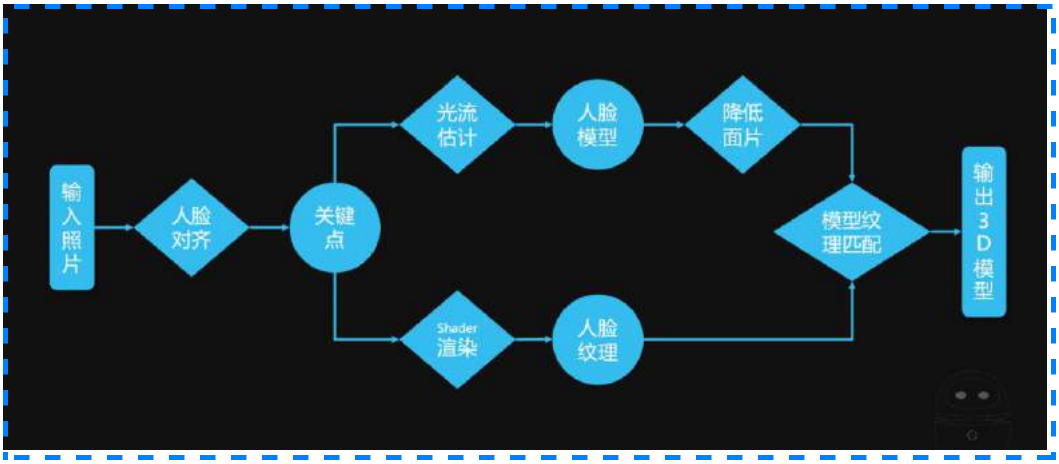


可以看到汉字中有很多变调的情况出现，所以要针对这个情况做处理，还需要针对不同的设备，如在手机端或者电脑端播放，做不同的后处理。这也需要花大量的时间调整。如在后处理优化的时候，左边的图，是优化前的图，在合成男声的图时，发现男声比女声的难合成，通过一些处理方法，把左边的声谱图，变成右边的声谱图，使声音更加均匀。所以称声音处理师为百万调音师。类似于演唱会上的调音师的功能，声音调整成到和硬件最匹配的状态。这些是在声学上的落地。

3.2 图像技术：眼睛和身体

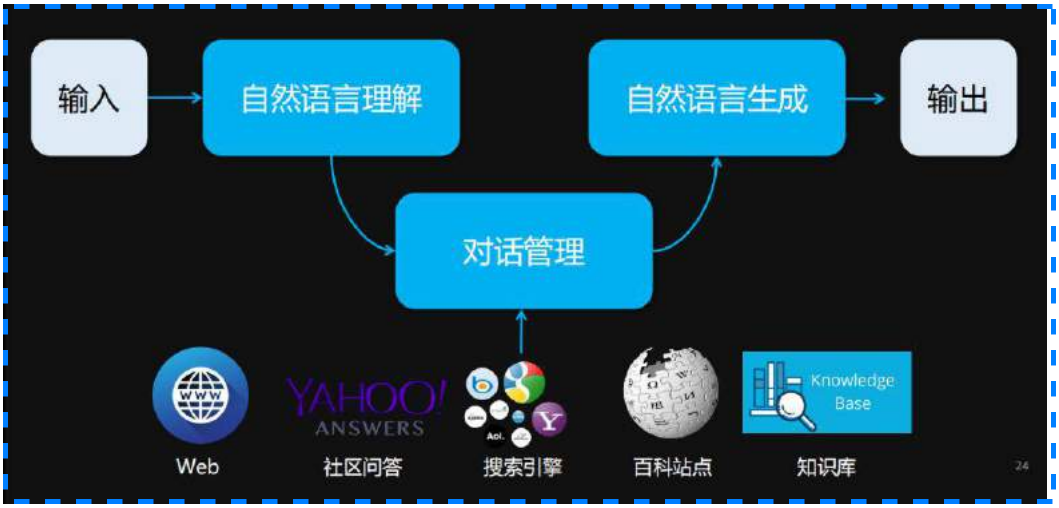
再看图像技术，图像技术相当于虚拟生命的眼睛和身体，因为图像技术中有计算机视觉，可以通过摄像头看到场景和用户，这是在人脸识别、场景识别用到的技术。还有个技术是 3D 重建技术，可以根据动漫的原型图，或者照片，来自动生成 3D 模型，通过一定的加工，可以把身体生成出来。由于 3D 也属于图像技术，非常成熟了，不再多说。大概说下在 3D 重建中使用的方法。

3.2.1 人脸重建



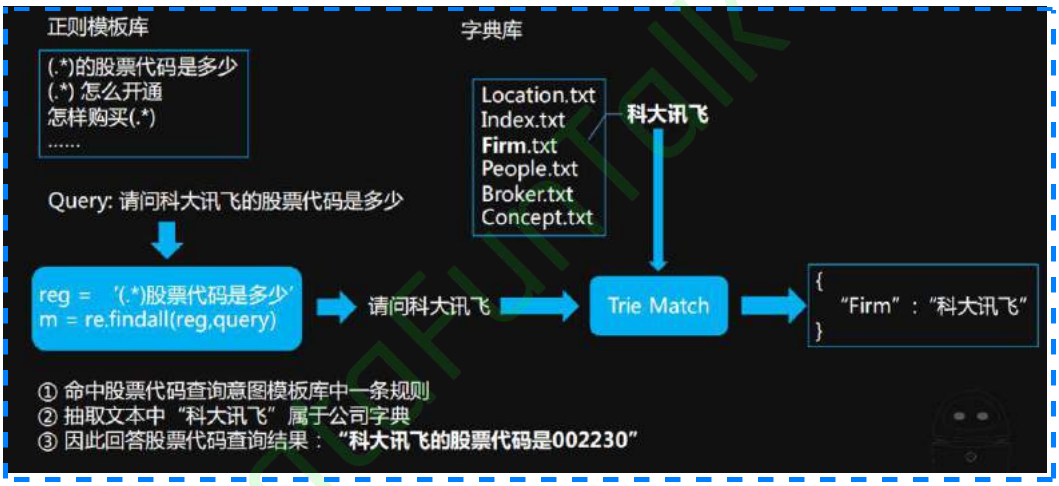
人脸重建中，目前用的最主流的方法是这样的，分两个工作流，上面的是在做 3D 模型，下面的是在做 shader 渲染（想象石膏像，希腊图像中有很多白色石膏像，如果我们想把石膏像中用颜色把鼻子、眼睛、耳朵涂出来，这就是 shader 渲染）。石膏像生成、把纹理贴到石膏像上，就可以生成最终的 3D 模型。Git 上有开源的 PRNet，3DMM 等经典算法，大家可以看下。

3.3 NLP 技术：基本交互



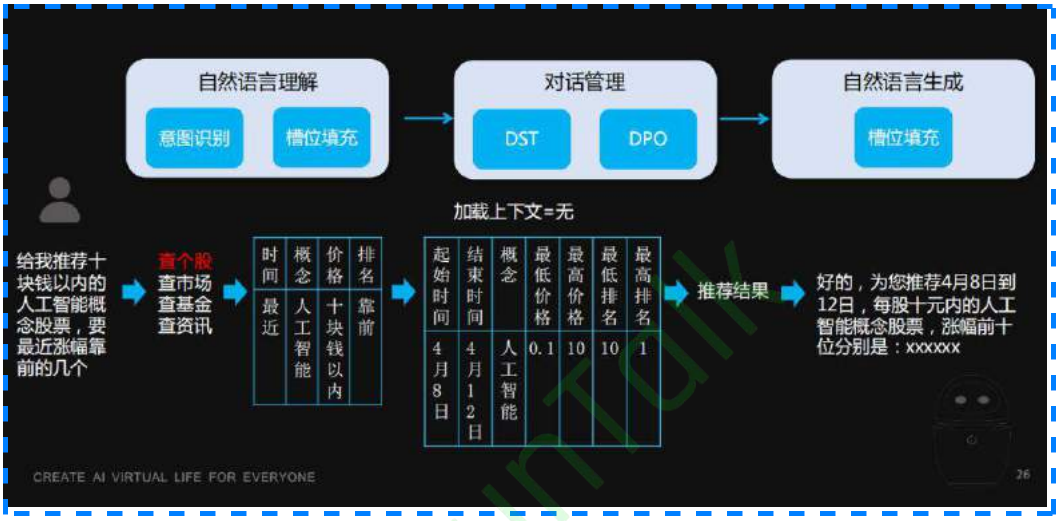
NLP 技术解决的问题是聊天机器人的基本交互，分为 5 个模块，核心的有 3 个：NLU、NLG、对话管理。整个交互模式是通过一个输入，到理解判断意图，到对话管理中解决槽位填充，到 NLG 中生成比较自然的回复，输出给用户，这是我们在 NLP 基本交互技术中的经典框架。

3.3.1 问答



问答时，聊天机器人分为 3 类：闲聊、问答、任务对话。问答是一问一答，如果是用最简单的方法正则模板，可以在代码中写很多正则模板，来一个 query 在正则模板中匹配，然后在建立好的字典库中，查找“科大讯飞”属于哪个字典库，最后查到属于 Firm 的名称，直接把科大讯飞的实体抽取出来，然后查已经建立好的数据库，比如建立了一个 key 和 value 的数据库，分别对应每个公司的股票代码，就会查到答案是 00230。这就是问答的最简单的做法。在做问答的过程中，还有语义解析、文本解析等方法。检索性的做法是在工程中用的最多、也是最有效最简单的方法。

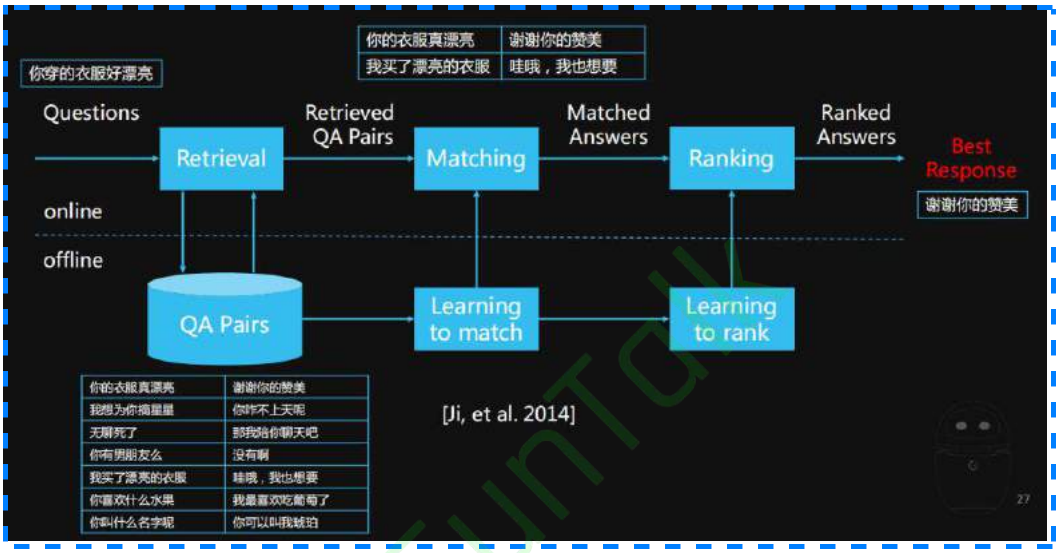
3.3.2 对话



对话是一个任务型的对话，是为了解决带有目的性的任务。比如用户问“给我推荐 10 块钱以内的人工智能概念股，要最近涨幅靠前的几个”。这句话带给我们的信息，需要 NLU、DM 和 NLG 这几个模块来解决。在 NLU 中需要

判断意图，假设用分类模型判断出来意图是查询个股，然后开始填槽，包括时间、概念和排名。查找到了时间是“最近”，概念是“人工智能”，价格是“10 块钱以内，涨幅靠前”。把这些信息变成标准槽位，起始时间是 4 月 8 日，对话管理中，根据 DST，接下来要做的事情是，给用户推荐排名最高的结果。这是 DPO 的事情。DST dialogue state tracking，DPO：dialogue policy optimization。最终推荐的结果是通过 NLG 给用户返回一句话，给用户返回一句话：为您推荐 4 月 8 日到 12 日，每股十元以内的人工智能概念股票，涨幅前十位分别是：XXX。

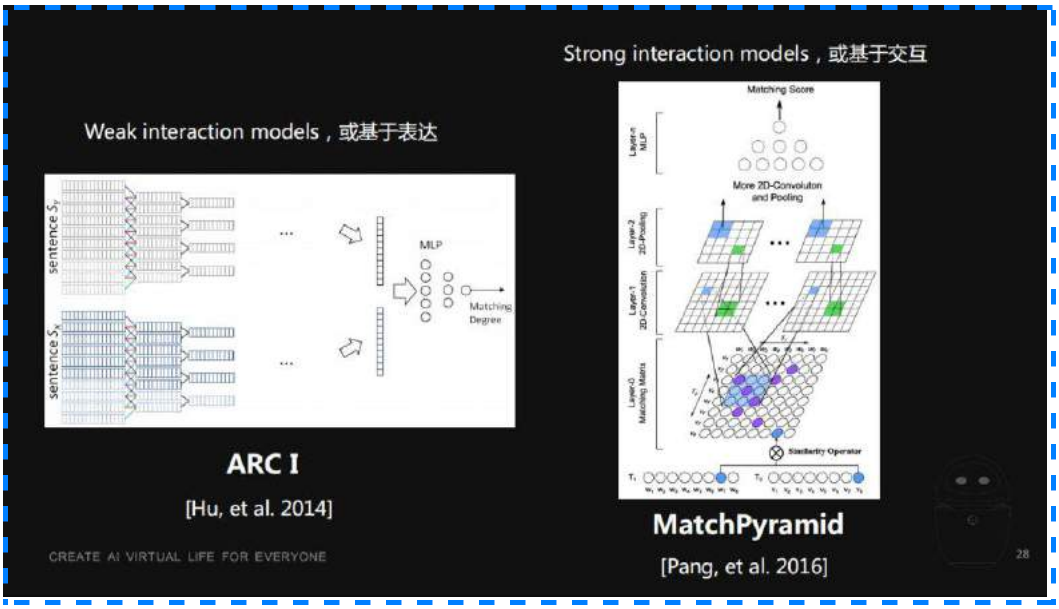
3.3.3 闲聊



闲聊做的最好的是检索型的，怎么做的呢？比如，用户有一个语料，“你穿的衣服好漂亮”，我们已经建立好了很多的 QA pairs，现在要做的事情，就是把这句话放在数据库中，进行相似度匹配，把最相似的句子挑出来，得到

上面表格这两句，对这两个句子做 ranking，得到最好的。这里最关键的技术是 matching 技术，怎样从数据库中找到相似句子？

3.3.3.1 Matching 算法



TF-IDF，最大子串，基于字的相似度匹配，这里想提的是计算语义相似度的 matching 算法。有两种典型做法：基于表达的算法，和基于交互的算法。这是想解决基于语义相似、句子不相似时，怎样做好 matching。

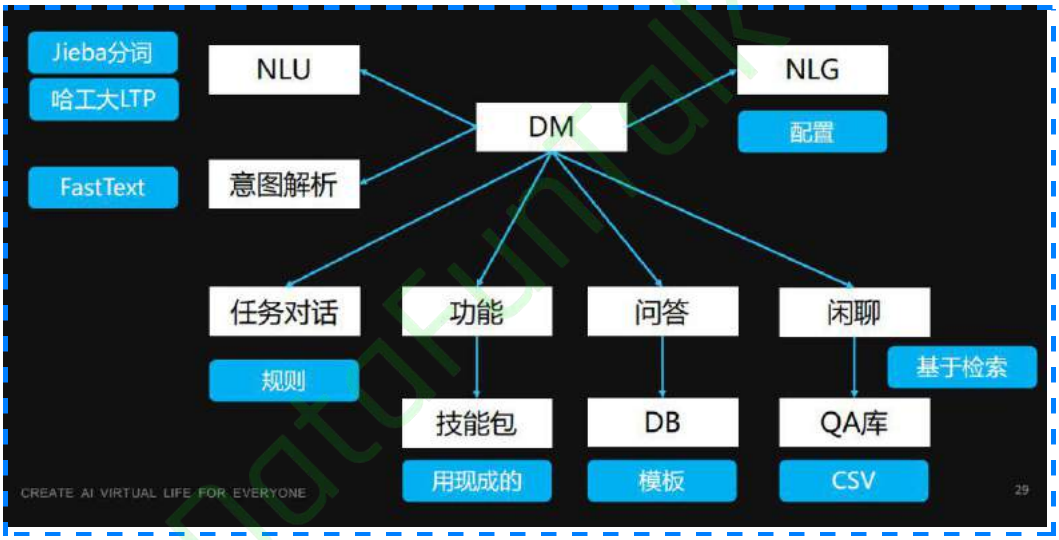
先看基于表达怎么做，基于表达最早的论文是诺亚方舟实验室在 2014 年发表的，提出 2 种算法：ARCI 和 ARC II。把两个句子分别进行向量化，向量化的做法：word2vec，或 embedding 技术。把词嵌入，把词表示为一个向量，句子的向量矩阵，对它们做卷积操作，在每个句子上做卷积操作，最后得到

定长向量后，做多层感知机，计算匹配度，在分别句子操作后，最后一层做相似度计算。这是基于表达的方法。

基于交互的经典算法，是在 2016 年发表的，在各大比赛中都取得了较好的成绩。怎么做的呢？一开始就把两个句子揉在一起，对两个句子的每一个 word，都进行一次 similarity 的操作，这有多种方法，可去看论文，其实就是把两个句子，揉成一个矩阵。在这个矩阵上做卷积操作，最后经过多层感知机，输出 score。

这两种方式最典型的区别，一个是分开做的，一个是一开始就揉一起了。感兴趣的可以看下北大的严睿老师，对 matching 做过专题分享，网上可以搜到 PPT。

3.3.4 工程方案汇总

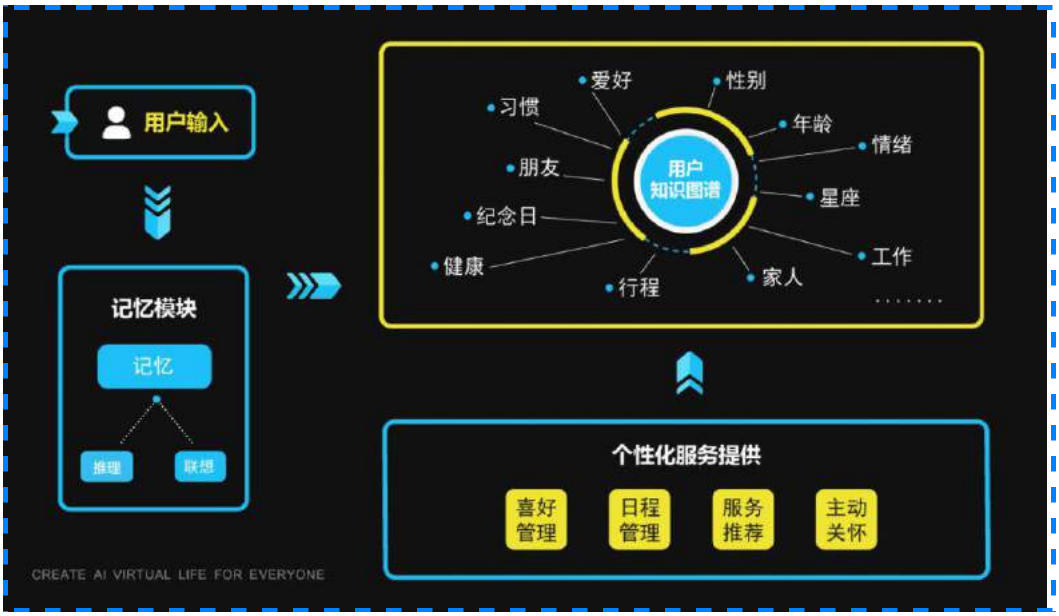


想要做 chatbot 蛮简单的，在工程方案汇总之后，NLU 不用自己做，可以使用 jieba 分词，或者哈工大的分词，意图解析可以使用训练数据跑一个 fastText 或者 BERT、transformer，得到解析；人物对话，需要自己写一些规则，功能也可以不用自己做，可以用海知智能的如意平台上很多技能拿来用。问答技能可以用模板搜索数据库中的答案，就可以了。闲聊基于检索，把一问一答的 QA 库存在 ES 中，使用相似度匹配放在句子中抽取出来。唯一需要写代码的，是写规则。NLG 通过配置来实现。这就是最简单的工程方案汇总。这只是最简单的方案，如果要做成产品，还需要考虑很多时间响应的事情。比如在大规模的用户并发的情况下，如何保证每条语句的返回结果都是 500ms 内的，此时如果用深度学习技术，就达不到这样的效果。在工程实现过程中，需要考虑很多效率因素。

说到 NLP 之后，谈一谈知识图谱：

3.4 知识图谱：跨越感知和认知的桥梁

DataFunTalk

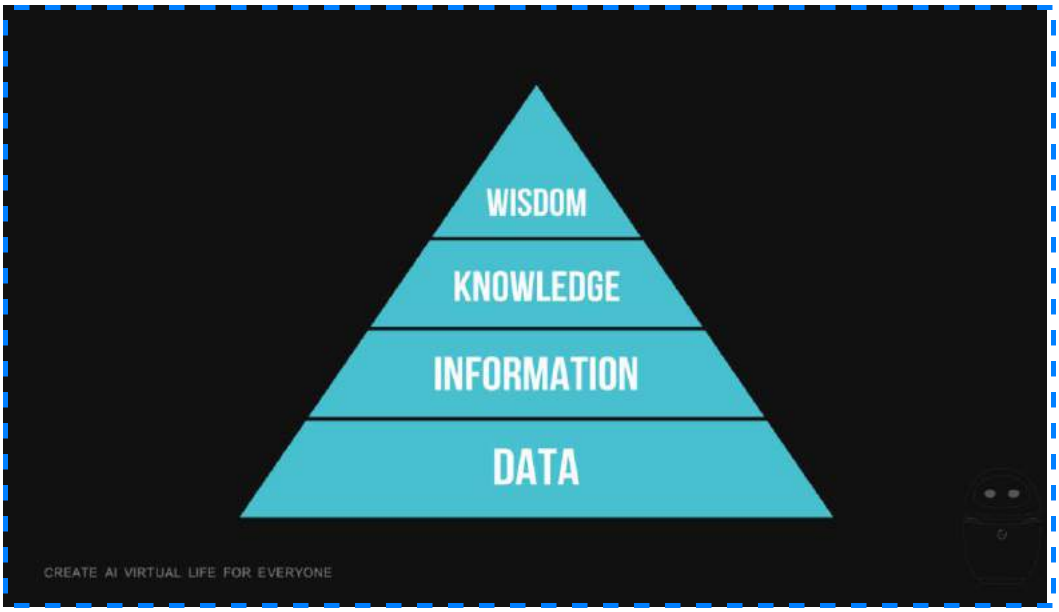


现在，知识图谱技术在学界、工业界，都公认是从感知智能通往认知智能的桥梁。是真正解决机器人认知智能的方法，来看下为什么这么说：

知识图谱技术，可以为聊天机器人带来什么进展？如果有了 KG 技术，可以通过记忆模块，抓取用户输入，下面举例说明怎么做。用户在与 chatbot 聊天的过程中，不断抽取用户的知识图谱，把它存在简单的 mysql 或者图数据库，存成用户的知识图谱，知道了喜好、性别、年龄，图谱可以动态更新。有了图谱就可以提供更多的个性化推荐服务，如日程、关怀等。比如今天被老板骂了，可以查到用户是做什么的，老板是谁，主动关怀一句“你们老板太抠门了“。这是 KG 在个性化推荐上可以做的事情。

简单聊一下 KG 的本质：

DataFunTalk

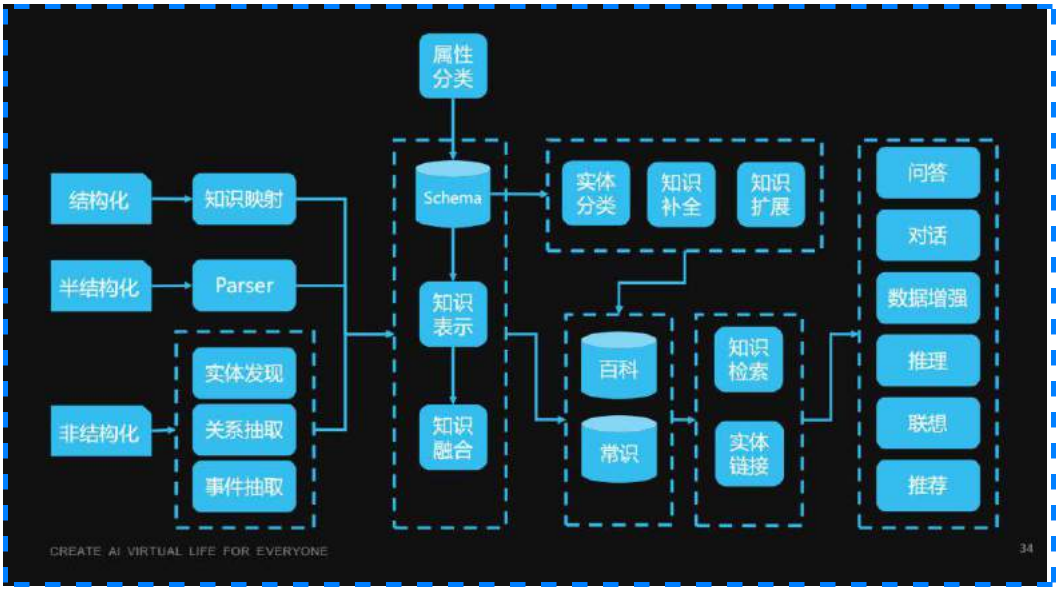


简单说，KG 是知识+图。这是最简单的解释。想象一下，传统的数据是到 information 级别的，从 data 到 information，到 knowledge，再到 wisdom，是不断凝练的过程。这是上世纪初学者提到的 DIKW 体系。什么是数据，“姚明身高是 2.26m“。2.26 是数据，“身高是 2.26”这是 information；把姚明用节点表示出来，节点周围加上属性，比如姚明的妻子是叶莉，姚明效力的球队是上海队、中国队等；把这些知识总结成图的形式，就是知识图谱的最简单的定义。把知识以图的形式来表示。

DataFunTalk

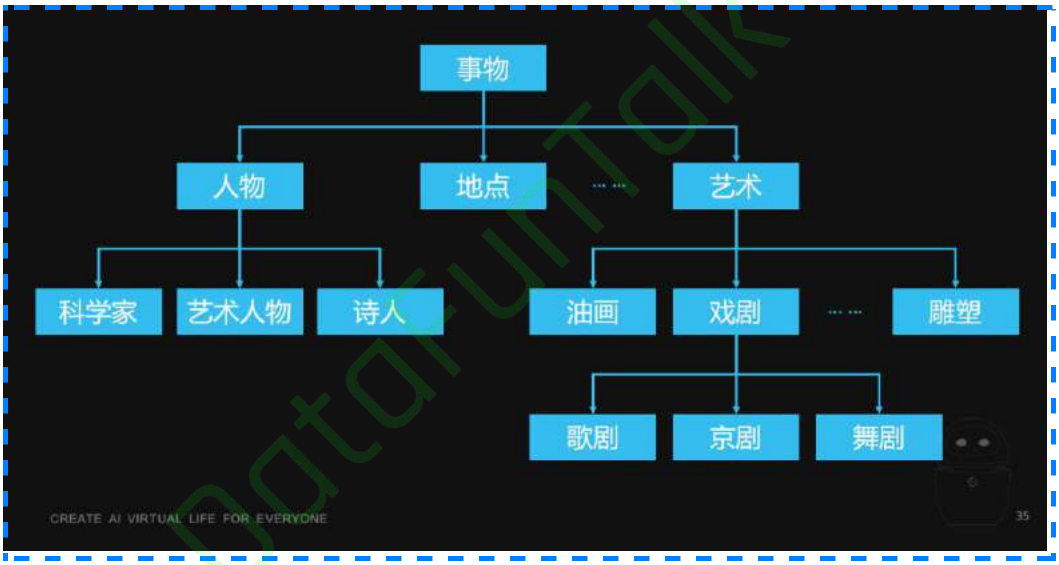
知识图谱实现的过程中，用到了这些模块，从知识来源到知识应用，共 5 层。从不同的知识来源，加工后，得到知识图谱，知识图谱之上，可以用知识映射的手段，做问答、推理、联想、推荐、数据增强等。比如问答，通过知识检索、实体发现、实体对齐、实体消歧，在百科知识图谱上做出来的结果。这是知识图谱的整个体系。

DataFunTalk



流程上看是这样的，数据，到知识映射，到知识表示，形成百科知识图谱，通过知识检索、实体链接，做成上层应用。这个图是最基本的知识图谱构建引擎。

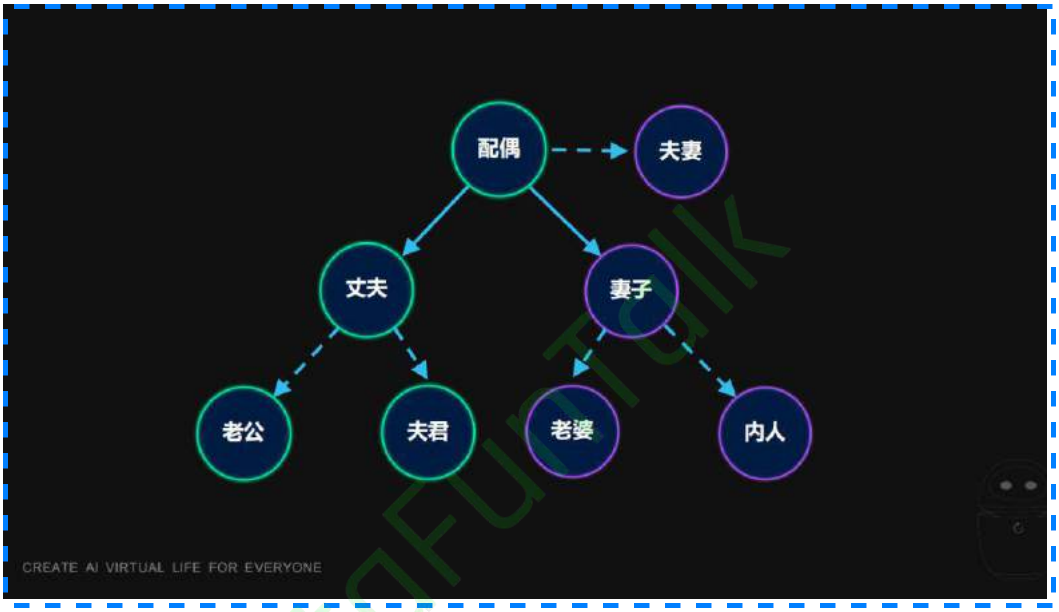
3.4.1 KG schema



知识图谱不得不谈 schema。

schema 是知识图谱的灵魂。一个形象的比喻，如果把 schema 看成是骨架，整个知识图谱就是一个人。如果把知识图谱看做一棵大树，schema 是它的树干树枝。如果没有 schema，数据就会比较散，没有标准。这里列出来的是一个简单的例子，事物下有人物、地点、艺术。艺术下有油画、戏剧、雕塑。这其实是一个分类体系。如果学编程，知道什么是类和对象。类相当于 KG schema，对象相当于实体。比如任务科学家，下面有个屠呦呦，他是一个实体，科学家是一个 class。

3.4.1.1 实体属性上下位



构建 schema 的过程中，有很多可参考的资料，比如 schema.org 以及 cnSchema，我们在这些 schema 的基础上，定义了自己的知识图谱。

构建 schema 的过程中，会发现很多关于实体属性上下位的事情。在拥有这个东西之后，做问答，可以很方便地进行语义解析，比如问“特朗普的老婆是谁”，可以通过老婆的同义词，查找到，“特朗普的老婆是谁” = “特朗普的妻子是谁”” = “特朗普的配偶是谁”。可以知道特朗普和梅拉尼娅属于夫妻关系，使用实体属性上下位的方法，会对问答推理带来作用。

3.4.2 KG 构建

3.4.2.1 数据来源

首先是数据来源，可以是半结构化、非结构化的数据，我们用的有百度、萌娘百科，二次元领域的聊天机器人需要用到萌娘百科，知乎、维基百科等。

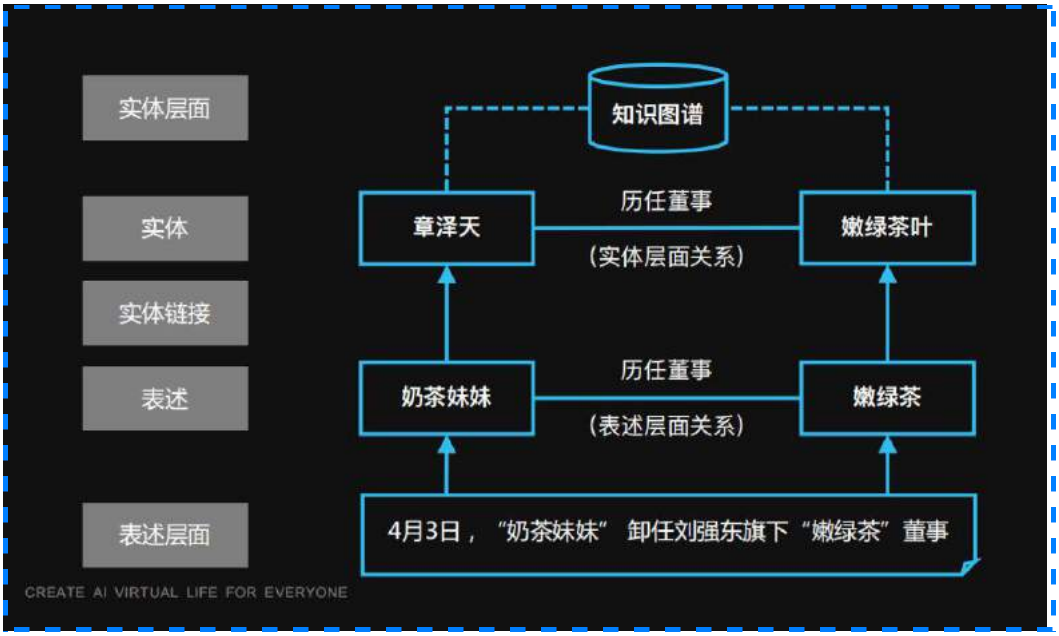
3.4.2.2 知识表示

用什么来表示知识？知识以什么形式存在，以及知识图谱的存储。可以用原生的图模式，也可以用 RDF（resource description framework，经典的三元组形式 ）。主谓宾的形式表示。

3.4.2.3 知识补全

利用 schema 做知识补全，比如在百度百科上发现，陈道明有一条边，妻子，指向杜宪，杜宪的百科中，没有提到陈道明是她的老公，此时用 schema 的定义，来反推，杜宪的老公是陈道明，用这种方法，可以把知识补全。可以直接把知识自动化补全。

3.4.2.4 知识扩展



知识扩展层面，举一个例子，4 月 3 日，“奶茶妹妹”卸任刘强东旗下“嫩绿茶”董事。“奶茶妹妹”和“嫩绿茶”两个实体，不太清楚指的是谁。这里可以通过 NLP 中的实体链接技术来判断，奶茶妹妹其实指的是章泽天，嫩绿茶其实是指“嫩绿茶叶有限公司”。可以把知识补全到知识图谱中，奶茶妹妹卸任，指不是嫩绿茶的现任董事，而是历任董事，可以更新到图数据库中。

3.4.2.5 新知识发现

DataFunTalk

新知识发现，是知识图谱要做的一个非常重要的工作。现在网络流行词很多，需要不断爬取、发现新知识。比如什么叫互联网创业公司，主营业务是 P2V，是 PPT to VC。如果项目是 Python 写的，就是一个 machine learning 的项目。什么是“不满式创业”，这都是知识发现要做的事情。

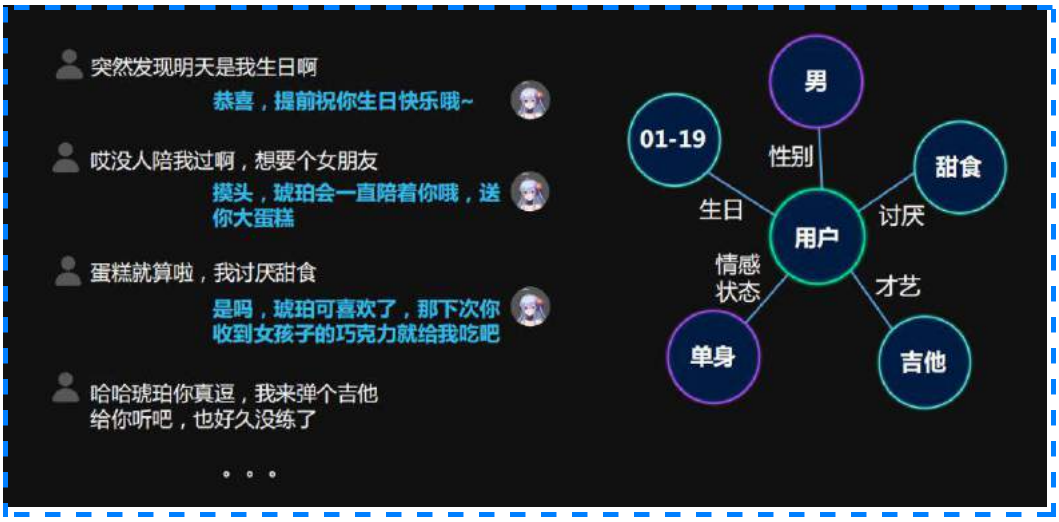
3.4.2.6 知识更新

刚才讲到章泽天，也是知识更新的层面。需要爬取这些热搜，不断增量更新整个知识库，这里面又牵扯到是否要全量更新，每个月做一次全量更新；增量更新，只需要更新节点，可以更新一跳关系，两跳关系，现任妻子变成前任妻子，现任董事变成前任董事，这都是知识更新要做的。

3.4.2.7 问答

问答方面，牵扯到问答领域几种典型的方法，有基于语义解析、基于模板、基于深度学习、基于检索的。融合了多种 NLP 技术，如问 CEO 国籍，狗尾草可能是一种植物，也可能是一个公司，CEO 可能是首席执行官，也可能是别的意思，通过基本的 NLP 处理和实体连接等技术把实体关系找出来，然后语义组合之后，变成 sparql 的查询语句，要找的答案其实是狗尾草的 CEO 是谁，是邱楠，邱楠的国籍是中国，然后通过比较自然的方式回复给用户。

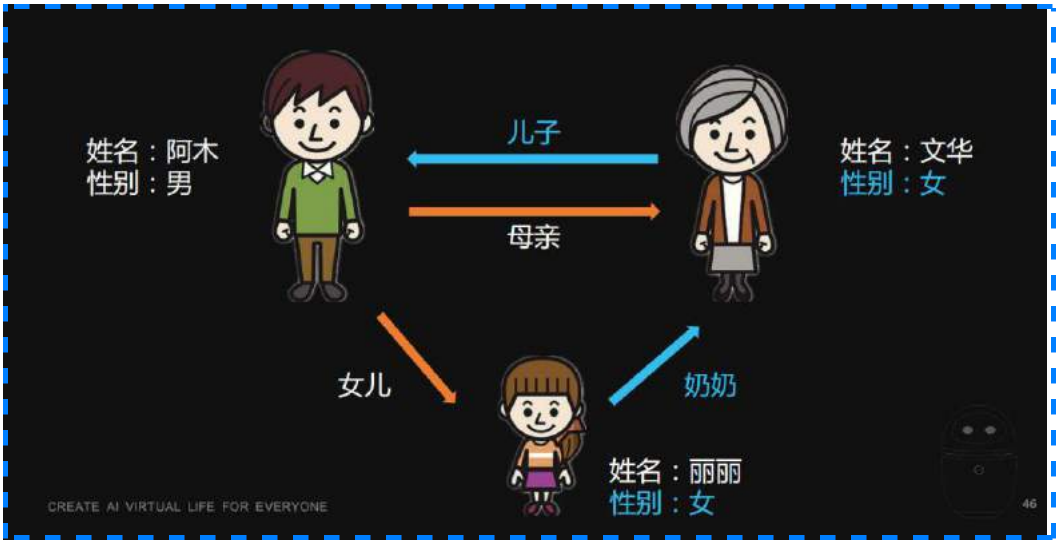
（1）理解用户



比如突然发现明天是我生日，可以抽取出我的生日日期。当时做 PPT 的时候是 1 月 18 日，记住用户的生日是 1 月 19 日。比如想要个女朋友，判断性别是男，情感状态是单身，讨厌甜食。“弹个吉他给你听吧”，才艺就是吉他。怎么抽取用户的信息，保存在图谱中，这个信息，可以给用户更多的推送。这是理解用户的过程。

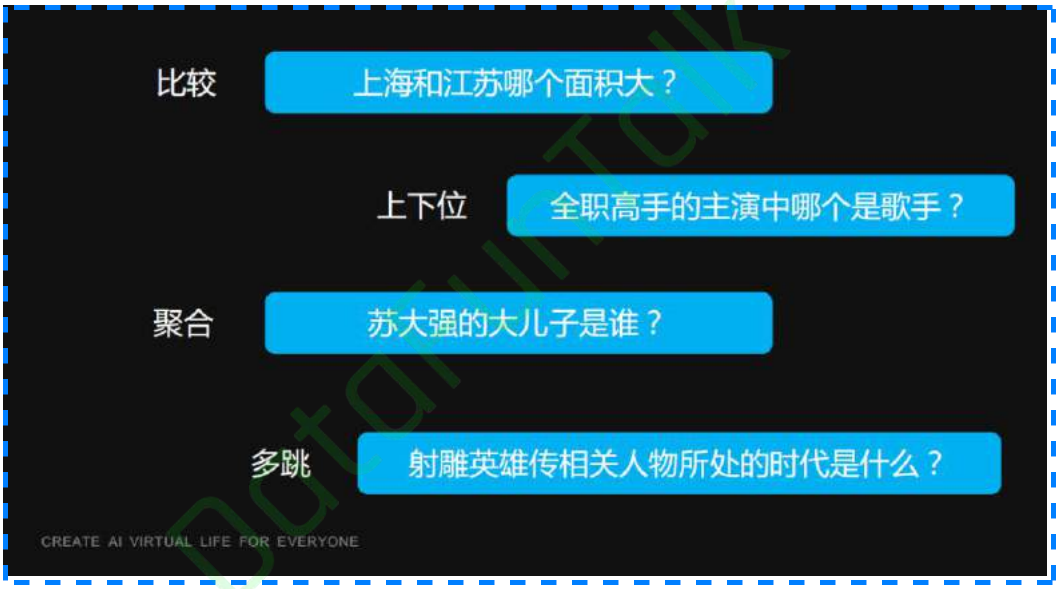
（2）推理机

DataFunTalk



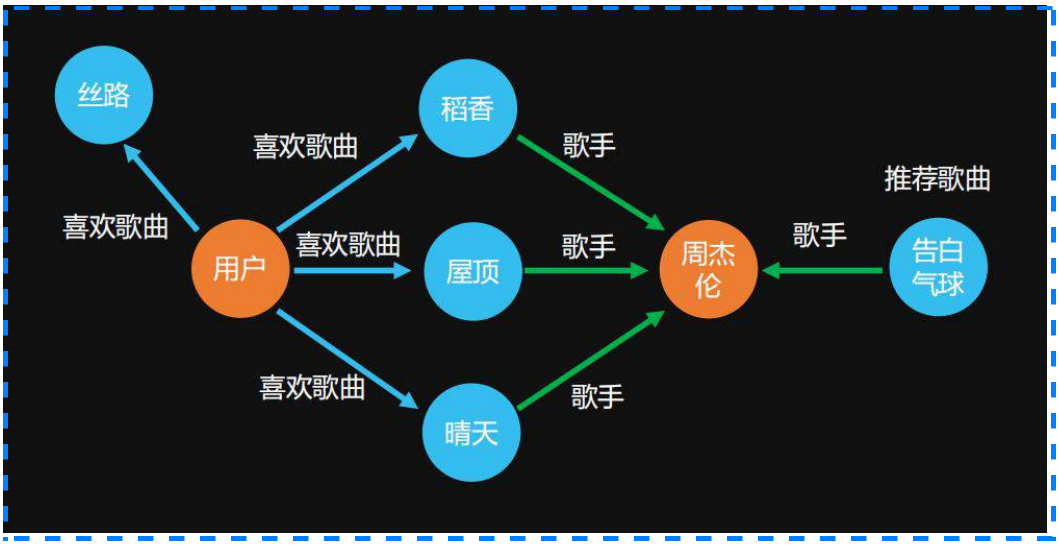
推理有推理机，类似于知识补全。比如知道了某个人，是一个男性的母亲，就可以知道性别是女，这里蓝色的是已经有的，黄色的是需要推理得到的知识。比如她的奶奶是这个人，她又是这个人的女儿，那么她的性别应该是女，这是推理中用到的方法。

（3）推理规则 QA



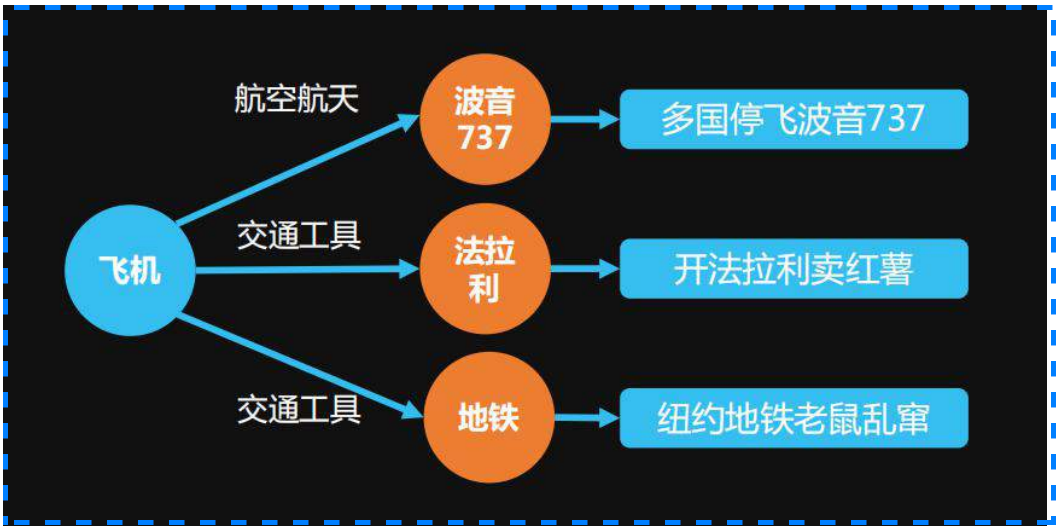
可以使用比较型、上下文、聚合型、多跳型，上面是规则推理的例子。

（4）联想：歌曲推荐



比如用户通过音乐播放记录，知道用户听了四首歌，都指向了周杰伦，下次要推荐歌手的时候，会以更大的概率推荐周杰伦的歌曲。这是在歌曲推荐中的应用。还有一些信息提供，比如用户提到飞机，可以通过上下位关系，查找相应的实体，提供信息。

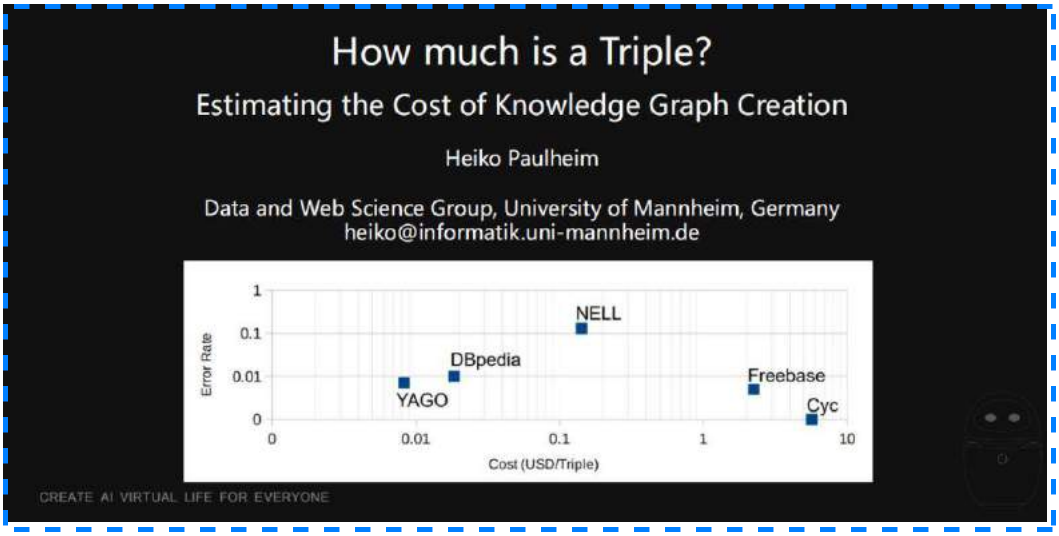
DataFunTalk



3.4.2.8 知识存储

根据实际情况看，传统关系型数据库能解决的问题，尽量不用图数据库，如 ES 是用到的较多的数据库，效率足够高；RDF 图数据库， jena 是有代表性的 RDF 图数据库；Neo4j 是有代表性的原生图数据库，可以根据不同的情况使用。

3.4.3 构建知识图谱的成本



是否需要构建知识图谱，是要考虑成本的。比如 Cyc，每一条知识图谱的成本是 5.71 美元， DBpedia，1.85 美分一条。因为它出现的比较晚，可以用深度学习技术提升构建知识图谱效率，来降低成本。

公开的知识图谱，放了部分数据在上面，也可用这个网址查询。

DataFunTalk

3.5 音乐技术：表现

这里播放一个视频。目的是根据不同的音乐跳不同的舞蹈，前面是机械舞，后面是古风舞。这里用到了音乐检测技术。下面是 onsite detection 的流程图：

DataFunTalk

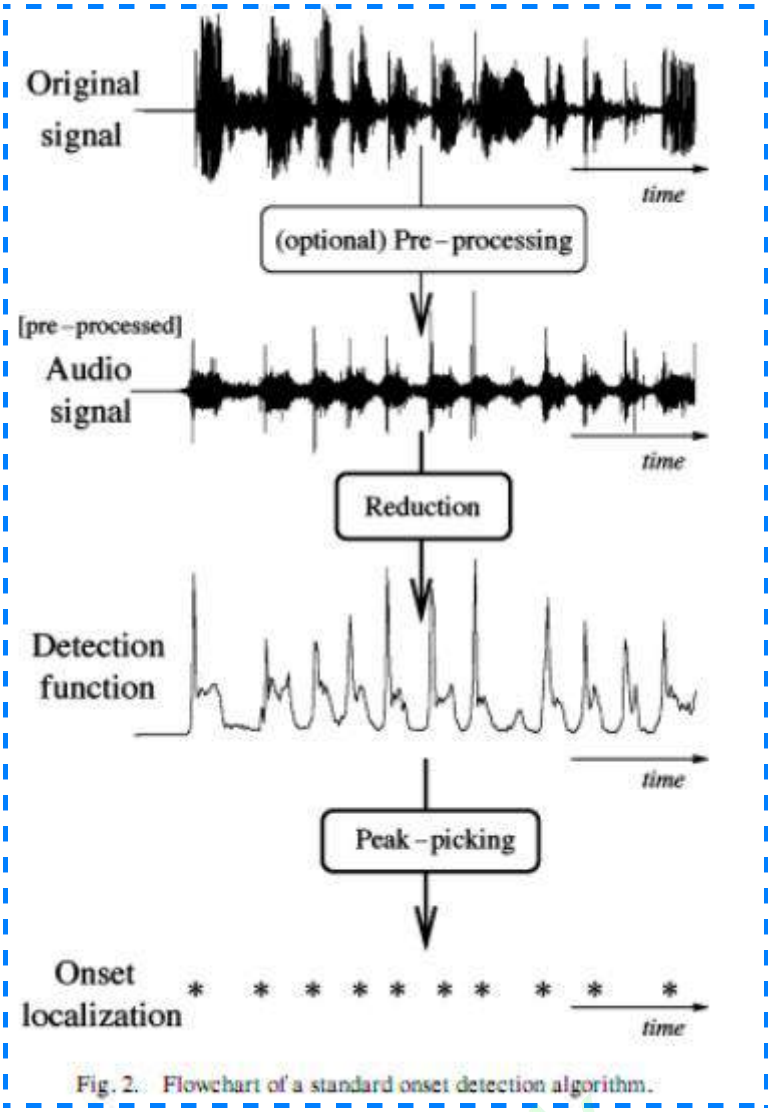
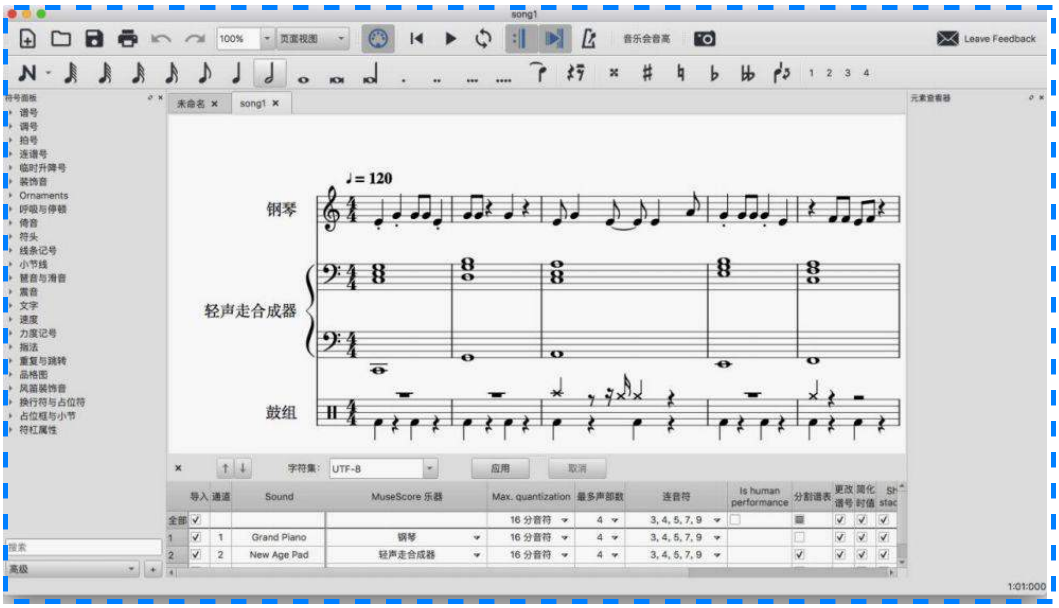


Fig. 2. Flowchart of a standard onset detection algorithm.

音乐属性检测包括多方面，如节奏、韵律、鼓点，有了这样的技术，产品在跳舞时，会把节拍卡的很好，表现会很强。



自动作曲方面 ,机器人可以自动作曲和演唱。在 Magenta 的技术上做了改进，使用 ABABB 的形式，主歌副歌主歌副歌副歌，韵律表现：C 跳到 G，再跳到 M，F，C，来体现音乐生成的感情色彩。这也是我们在做的有意思的尝试。

3.6 虚拟生命的发展阶段

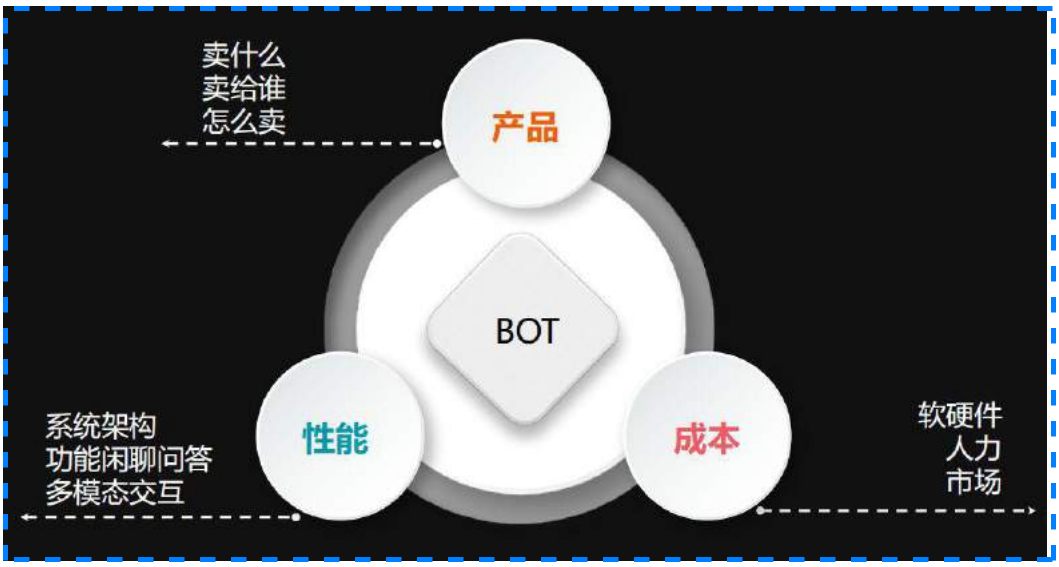
虚拟生命，现在还是在 1.0 的阶段，还有很长的路要走。

其应用场景，如车载，风控等：

工程实践中的其他问题

1. 多要素妥协

DataFunTalk



最后再谈下工程方面的问题，即使有很好的技术，也不一定能能做出好的产品。做好产品是多重要素的特性，不仅要考虑卖给谁、怎么卖、卖什么，还要考虑系统性能，以及投入多少的人力，以及整个市场是蓝海市场，还是红海市场。尤其是 chatbot 市场，已经不是红海市场，完全变成血海市场，chatbot 的产品是需要慎重考虑的，用什么形式进入市场。

2. 商业模式

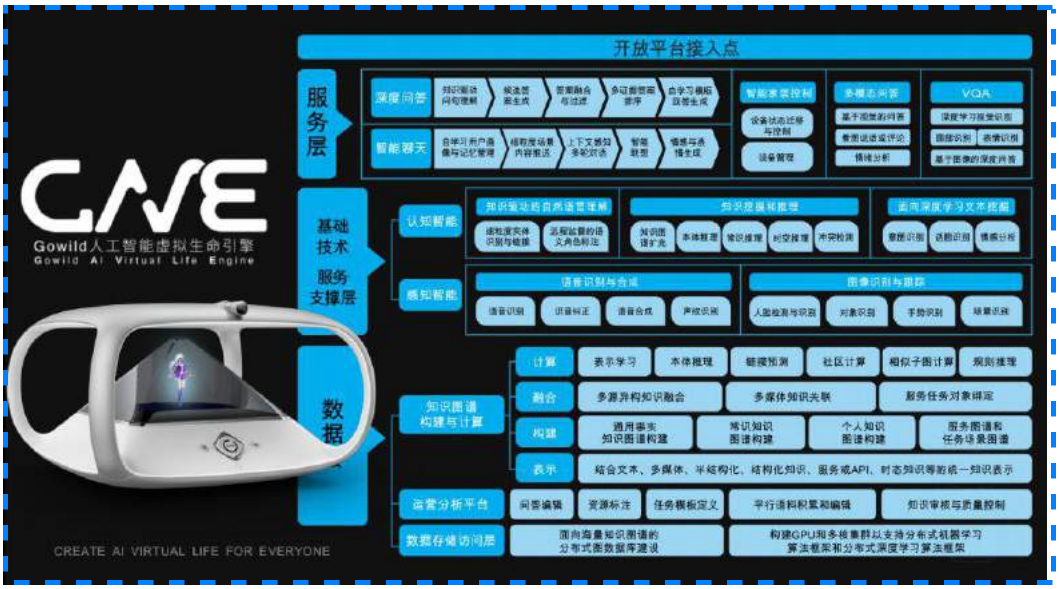
DataFunTalk

前段时间较火的，夸夸机器人，刘知远老师在清华的夸夸机器人群里的聊天记录。聊天机器人可以不停夸你。会有人买单吗？否定的。根本上来讲，这不是一个刚性需求。感兴趣的话我们可以线下继续讨论商业模式的问题。

最后，大家可以看下我们推出的 Gowild AI Virtual Engine（GAVE）人工智能虚拟生命引擎，已经应用到我们的产品中，也欢迎大家到我们的官网了解更多信息：

https://www.gowild.cn/

DataFunTalk



本次分享就到这里，谢谢大家。

DataFunTalk

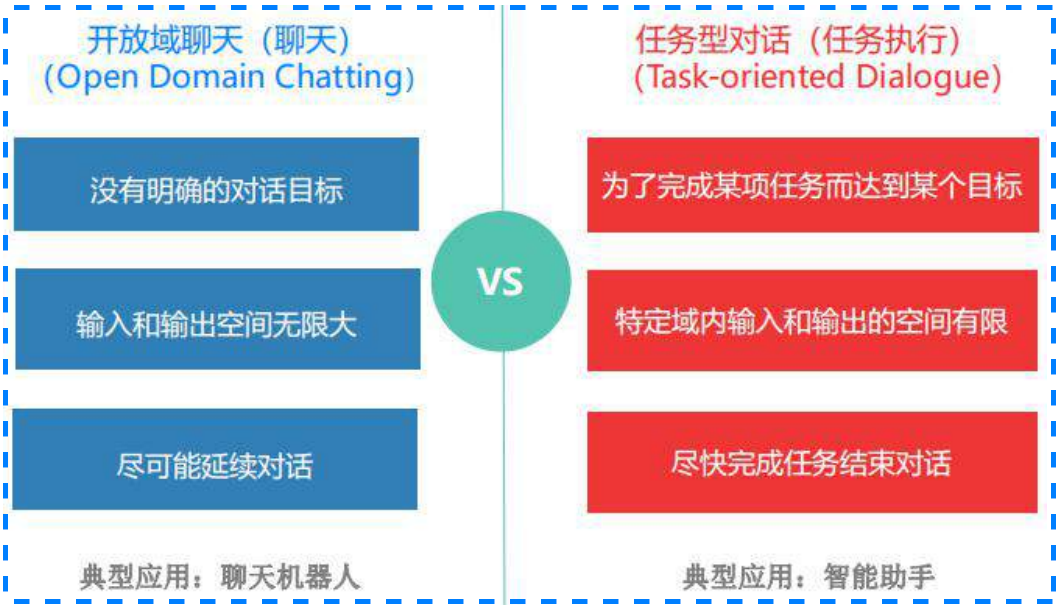
人机对话关键技术及挑战

嘉宾：张伟男　编辑：靳韡赟

导读：近年来，人机对话受到了学术界和工业界的广泛重视。在研究上，自然语言理解技术逐渐从序列标注向深度学习的方向发展，对话管理经历了由规则到有指导学习再到强化学习的发展过程，自然语言生成则从模板生成、句子规划，发展到端到端的深度学习模型。在应用上，基于人机对话技术的产品也层出不穷，如苹果公司的 Siri、亚马逊的 Echo 音箱、微软的 Cortana、Facebook M 以及谷歌的 Allo 等。同时，各大企业纷纷研发及收购 AI 平台，如微软研发 Luis.ai，三星、Facebook 和谷歌分别收购了 viv.ai、wit.ai 和 api.ai，百度自主研发了 Duer OS 以及收购 kitt.ai，以上种种迹象表明了各大企业将在今后着力发展人机对话技术。本次报告将针对人机对话中的核心关键技术的发展前沿和研究趋势进行综述，主要包括开放域人机对话和任务型对话的关键技术及挑战。

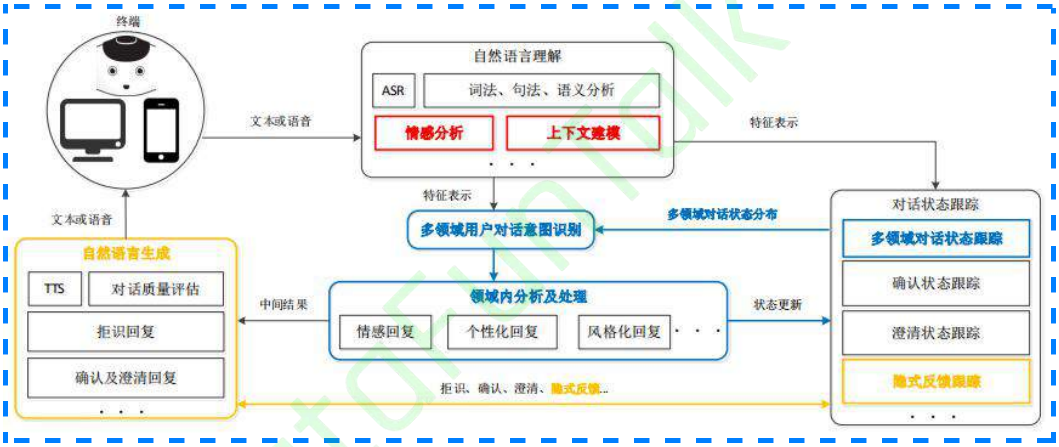
人机对话主要技术方向：

DataFunTalk



虽然人机对话主要包括四个功能：聊天、任务型对话、问答以及推荐，但主要集中在开放域聊天和任务型对话两方面。二者在目标、评价、最终目的上有一定区别，典型的应用系统也不大一样。

一．开放域聊天



开放域聊天的一个基础系统架构如上图所示。

1. 前沿研究点：回复质量

上图总结了 17 年到 19 年开放域的一些论文和一些相关的知识点，下面的部分主要关注回复质量。回复的话是通顺、流畅的。回复质量是一个基础，包含很多工作，也面临着多方面的挑战。

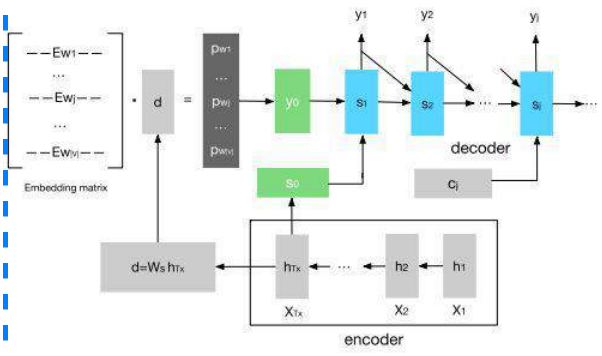
① 回复质量上存在的挑战

DataFunTalk

这里举两个例子，左边是单轮回复的例子，右边是多轮回复的例子。从中可以看出两个问题 ,第一个问题是在单轮回复中由于统计模型 loss function 的一个限制，通常采用的回复都是统计上占优的高频回复，叫做万能回复。而多轮回复可以看出是一个局部依赖的回复，就相当于它没有学到更高层次的上下文信息。这是一个局部学习，与上下文无关的问题。

② 优化初始的解码过程

• 动机：解码的第一个输出对后面的整句起决定性作用



Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Seq2Seq	42.870	3.226	0.382	0.000
Seq2Seq+LTS	**46.905**	**4.138**	**0.433**	**0.100**
ATT	**48.030**	3.884	0.450	**0.100**
ATT+LTS	45.634	**4.496**	**0.799**	**0.100**
HYP	**47.923**	4.403	0.516	0.000
HYP+LTS	47.488	**4.404**	**0.526**	**0.225**

Models	Mean	0	+1	+2	kappa
Seq2Seq	1.20	72.66%	14.66%	12,66%	0.309
Seq2Seq+LTS	**1.24**	71.33%	16.00%	12.66%	0.332
ATT	1.16	74.66%	12.00%	13.33%	0.398
ATT+LTS	**1.26**	68.66%	20.66%	10.66%	0.269
HYP	1.11	73.33%	16.66%	10.0%	0.340
HYP+LTS	**1.20**	70.00%	20.00%	10.00%	0.333

Qingfu Zhu, Weinan Zhang, Ting Liu. Learning to Start for Sequence to Sequence based Response Generation. CCIR 2018 24

我们在单轮回复对话中，积累了一些工作，首先是优化了解码器过程，通过观察真实的数据。如果第一个解码出一个高频词，比如我，你，我们，你们，那么通常来说回复不会太好，针对这个问题我们做了一个 learning to start for sequence to sequence 的 framework，后面大家可以把 learning to start 这个模块做进一步改进和提升。但目前来说对原有算法效果已经有了一定的效果。

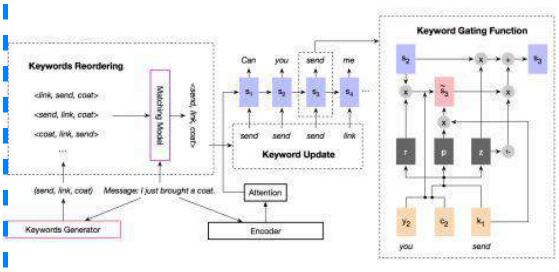
③ 更好地利用检索结果提高对话生成质量

其次，今年发表在 ACL2019 上面的一篇文章，检索式和生成式是互相对抗又互补的结果，检索式结果可以保证相关性、流畅性比较好，但是多样性受限，不论怎样检索都跑不出编辑好的话术。生成式的好处就是机器可以创造，但生成式模型面临着万能回复的问题。利用检索式结果和生成式结果互相增强的方式，提出这样一个模型。

④ 基于可控内容生成提升对话回复多样性

DataFunTalk

• 动机：万能回复的特点是无信息量，通过控制回复中生成的内容能够提升生成质量



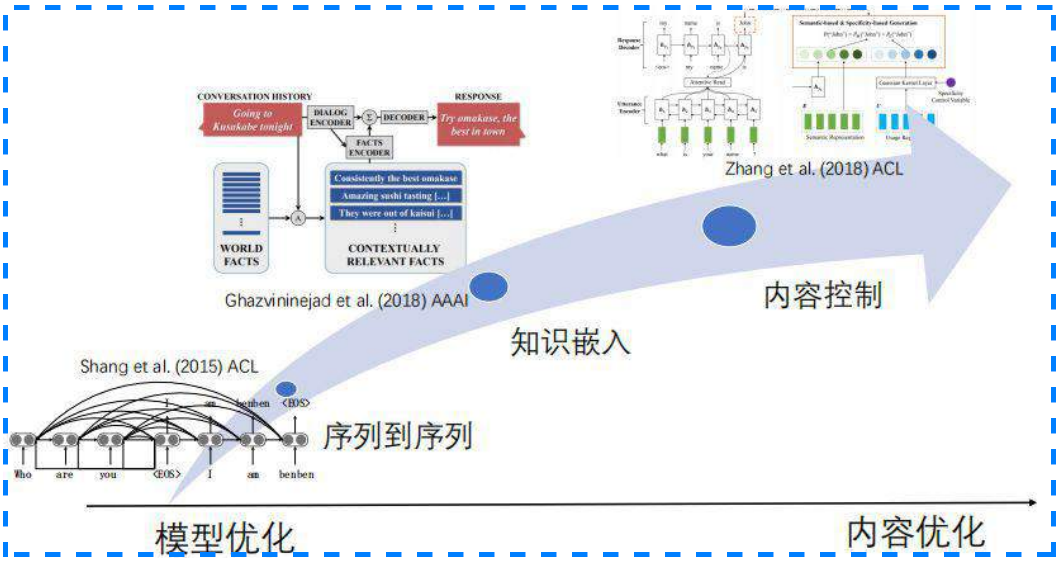
	Win	Tie	Lose	Kappa
Ours vs. Seq2BF	39.2%	30.5%	30.3%	0.29
Ours vs. TASeq	40.2%	28.7%	31.1%	0.45
Ours vs. Ours w/o reordering	39.2%	42.8%	18.0%	0.56

	N_k	# of UNI	Distinct-1	# of BI	Distinct-2	LEN	TRR	KRR
Ours w/o reordering	1	1,770	0.035	8,965	0.223	5.01	0.894	0.894
	2	1,840	0.028	12,768	0.234	6.46	0.977	0.915
	3	1,868	0.020	16,128	0.198	9.14	0.993	0.914
	4	1,920	0.016	19,281	0.172	12.20	0.996	0.906
	5	1,938	0.013	21,578	0.154	15.01	0.997	0.899
Ours	2	1,804	0.031	11,210	0.233	5.80	0.984	0.943
	3	1,822	0.023	14,043	0.198	8.08	0.994	0.937
	4	1,892	0.018	16,984	0.173	10.82	0.997	0.925
	5	1,897	0.014	19,068	0.153	13.45	0.997	0.920

Qingfu Zhu, Wei-Nan Zhang, Lei Cui, Ting Liu. Retrieval-Enhanced Adversarial Training for Neural Response Generation[C]. ACM TALLIP 2019

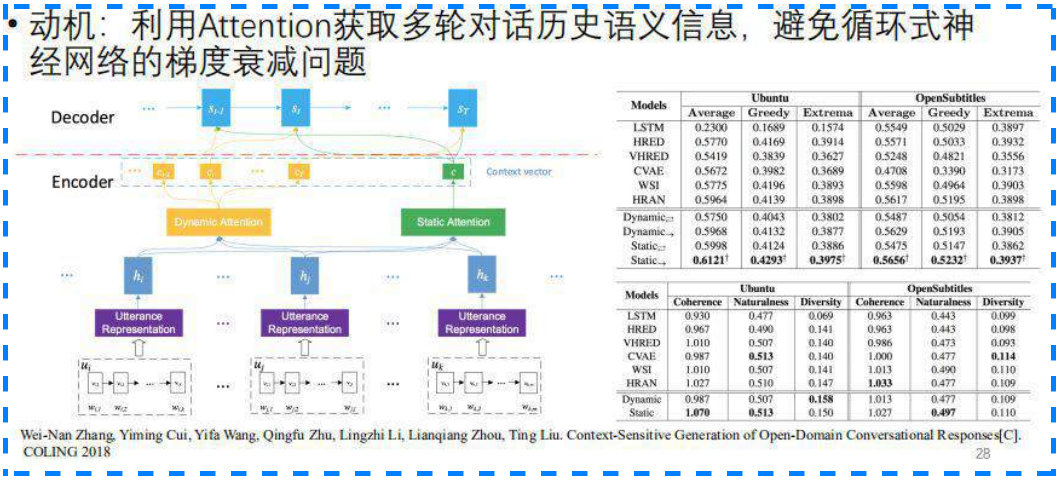
尽管我们可以把大量的对话数据交给模型，希望可以产生我们期望的结果。但是模型不一定会产生期望结果。所以我们考虑能不能通过一种控制技术让模型更加可控，使其生成包含指定内容的回复。首先在说一句话的时候 inference 出一系列的关键词，关键词不一定要有多个，可能是零个，比如你好，再见，回复里不需要关键词，但是如果是问姚明的身高，在回复中如果不出现一个数字表达式来表示身高就是不精确的一个回复。这里我们做了一个 keyword gating function 加到 sequence2sequence 的框架里，对框架性能有一定的提升。

⑤ 单轮聊天-回复生成技术



整个单轮聊天解决万能回复的问题的技术路线是从模型优化到内容优化的一个发展过程，其中有一定的知识嵌入。知识嵌入不是一件简单的事情，怎样把离散的知识加入到连续的模型中是目前存在的挑战。

⑥ 更好地建模多轮对话历史的语义



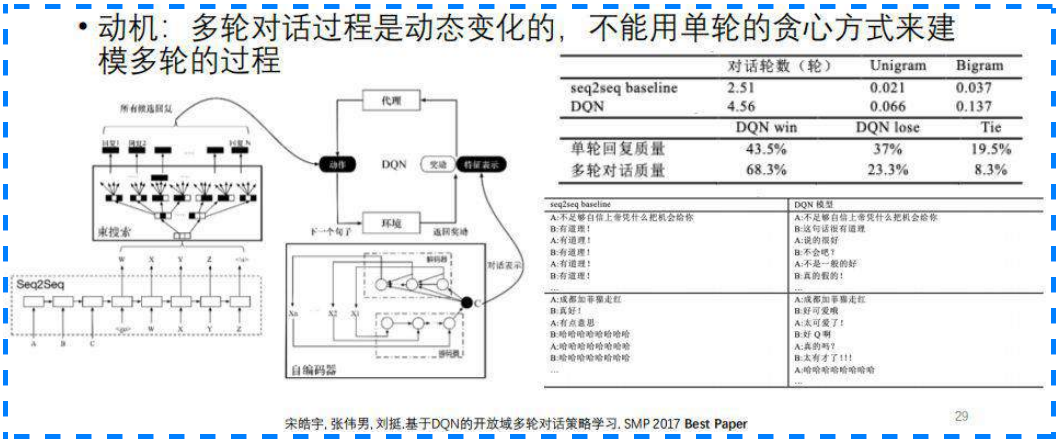
Models	Ubuntu			OpenSubtitles		
	Average	Greedy	Extrema	Average	Greedy	Extrema
LSTM	0.2300	0.1689	0.1574	0.5549	0.5029	0.3897
HRED	0.5770	0.4169	0.3914	0.5571	0.5033	0.3932
VHRED	0.5419	0.3839	0.3627	0.5248	0.4821	0.3556
CVAE	0.5672	0.3982	0.3689	0.4708	0.3390	0.3173
WSI	0.5775	0.4196	0.3893	0.5598	0.4964	0.3903
HRAN	0.5964	0.4139	0.3898	0.5617	0.5195	0.3898
Dynamic$_{\leftrightarrow}$	0.5750	0.4043	0.3802	0.5487	0.5054	0.3812
Dynamic$_{\rightarrow}$	0.5968	0.4132	0.3877	0.5629	0.5193	0.3905
Static$_{\leftrightarrow}$	0.5998	0.4124	0.3886	0.5475	0.5147	0.3862
Static$_{\rightarrow}$	**0.6121**†	**0.4293**†	**0.3975**†	**0.5656**†	**0.5232**†	**0.3937**†

Models	Ubuntu			OpenSubtitles		
	Coherence	Naturalness	Diversity	Coherence	Naturalness	Diversity
LSTM	0.930	0.477	0.069	0.963	0.443	0.099
HRED	0.967	0.490	0.141	0.963	0.443	0.098
VHRED	1.010	0.507	0.140	0.986	0.473	0.093
CVAE	0.987	**0.513**	0.140	1.000	0.477	**0.114**
WSI	1.010	0.507	0.141	1.013	0.490	0.110
HRAN	1.027	0.510	0.147	**1.033**	0.477	0.109
Dynamic	0.987	0.507	**0.158**	1.013	0.477	0.109
Static	**1.070**	**0.513**	0.150	1.027	**0.497**	0.110

Wei-Nan Zhang, Yiming Cui, Yifa Wang, Qingfu Zhu, Lingzhi Li, Lianqiang Zhou, Ting Liu. Context-Sensitive Generation of Open-Domain Conversational Responses[C]. COLING 2018

除了刚刚讨论的单轮聊天外，多轮聊天上下文无关也是一个比较棘手的问题。之前我们有很多工作，提出了两种 attention 的结构，尽可能保留对话历史的语义，由当前对话输入去决定哪个对话历史对它更有效，更有用，就是这样一个工作，在多个数据集上都取得了比较好的结果。

⑦ 动态建模多轮对话的策略

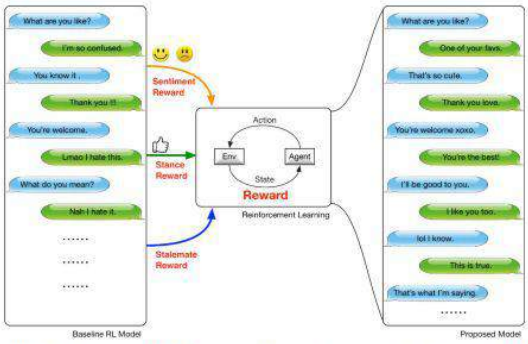


另外在多轮对话过程中，存在一对多的问题，给定一个对话的输入，其实有很多种对话的输出可以作为它的回复，那我们选择哪一个呢？我们想通过 n-best 的模式生成更好的结果，而且是更多样的一个结果，同时这个结果能够导致话轮的增长，所以我们说聊天的主要目的是延长话轮数目，然后我们以这样一个动机探索了 DQN 的一个方式，利用深度强化学习算法 DQN，使用深度价值网络对每一轮的候选句子进行评估，并选择未来收益最大的而非生成概率最大的句子作为回复的多轮对话策略学习方法。实验结果

表明，该文提出的方法相比较于 Seq2Seq 的方法将多轮对话的平均对话轮数提高了两轮。

⑧ 挖掘用户隐式反馈延长对话轮次

- 动机：用户的回复内容中通常包含了一些隐式反馈，如立场、情绪等的潜在信息，能够反映对话生成的质量



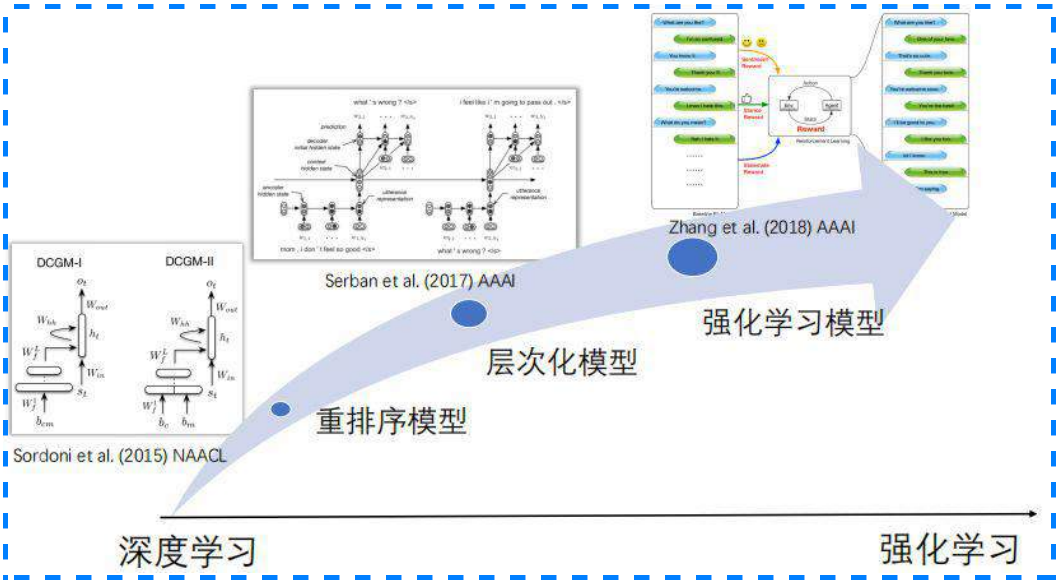
Model	# of simulated turns	
	OpenSubtitles	Twitter
Seq2Seq	2.39	5.82
RL-Seq2Seq	4.38	7.32
Ours	**4.64**	**7.79**

Dataset	Metric	Our-win	Our-lose	Tie
OpenSubtitles	S-Q	0.153	0.12	0.726
	E2R	0.147	0.1	0.753
Twitter	S-Q	0.16	0.073	0.767
	E2R	0.147	0.073	0.78

Wei-Nan Zhang, Lingzhi Li, Dongyan Cao, Lianqiang Zhou, Ting Liu. Exploring Implicit Feedback for Open Domain Conversation Generation[C]. AAAI 2018 30

下面这个工作是基于强化学习来做的，这里不是为了提高对话轮数。我们在观察“笨笨”（见最后的 demo 部分）的 log 的过程中有很多的用户在抱怨“笨笨”或者是夸赞“笨笨”，比如说“你怎么这么笨？”“你说的这是什么东西？”其实这些都是用户情绪宣泄和反应。但它反应的过程其实是上一轮回复的质量不好，所以我们把用户的反馈作为隐式的信号，加入到人机多轮的对话中。然后一旦我们发现用户对“笨笨”有不太好的评价，下一轮回复中我们要采用这种评价。考量上一轮生成的不好的回复，及时纠正下一轮生成的回复。这里面就是基于用户隐式反馈的一个工作，我们采用了三种隐式信号。

⑨ 多轮聊天-回复生成技术

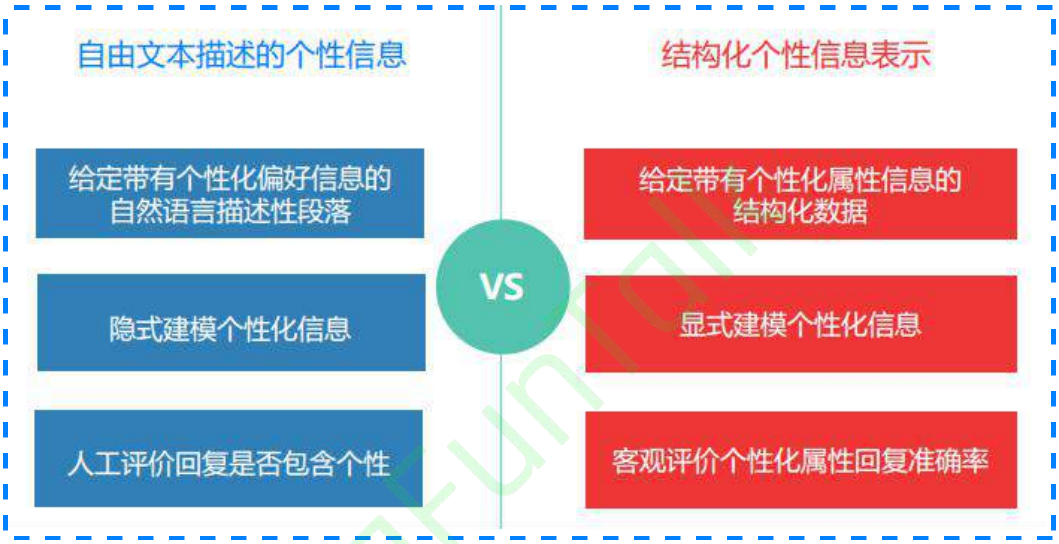


整个多轮多轮对话过程是从深度学习发展到强化学习的一个过程。

2. 前沿研究点：用户体验

除了对话质量，还有一部分影响用户体验的是上面特色的几个功能，包括情感对话和个性化，风格化，模拟机器人的画像等。

① 个性化回复生成



个性化分为自由文本描述的个性信息和结构化信息表示，自由文本描述的个性信息是外化气质内化的一个表现。比如说你养了一个宠物可能会间接反映出你是一个比较善良的人。结构化个性信息表示例如：你是否戴眼镜，你是否有房，你是否有车，你的年龄多大，你是男是女等等。

② 个性化回复生成代表性工作

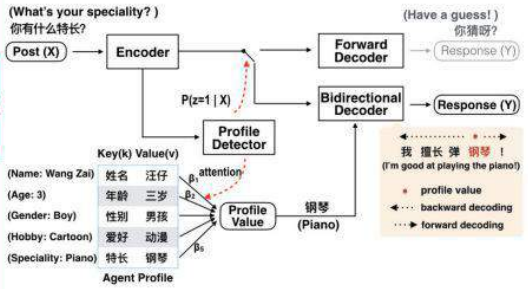
自由文本描述的个性信息

Persona 1	Persona 2
I like to ski	I am an artist
My wife does not like me anymore	I have four children
I have went to Mexico 4 times this year	I recently got a cat
I hate Mexican food	I enjoy walking for exercise
I like to eat cheetos	I love watching Game of Thrones

[PERSON 1:] Hi
[PERSON 2:] Hello ! How are you today ?
[PERSON 1:] I am good thank you , how are you.
[PERSON 2:] Great, thanks ! My children and I were just about to watch Game of Thrones.
[PERSON 1:] Nice ! How old are your children?
[PERSON 2:] I have four that range in age from 10 to 21. You?
[PERSON 1:] I do not have children at the moment.
[PERSON 2:] That just means you get to keep all the popcorn for yourself.
[PERSON 1:] And Cheetos at the moment!
[PERSON 2:] Good choice. Do you watch Game of Thrones?
[PERSON 1:] No, I do not have much time for TV.
[PERSON 2:] I usually spend my time painting: but, I love the show.

Zhang et al. (2018) ACL

结构化个性信息表示



Qian et al. (2018) IJCAI

我们说的这两个部分实际上都有相关联的一些工作，我们主要研究人性相关的一些信息，并且参加了去年的 NIPS ConvAI2 的一个评测。

③ NIPS ConvAI2 个性化聊天回复生成

Persona 1	Persona 2
I like to ski	I am an artist
My wife does not like me anymore	I have four children
I have went to Mexico 4 times this year	I recently got a cat
I hate Mexican food	I enjoy walking for exercise
I like to eat cheetos	I love watching Game of Thrones

[PERSON 1:] Hi
[PERSON 2:] Hello ! How are you today ?
[PERSON 1:] I am good thank you , how are you.
[PERSON 2:] Great, thanks ! My children and I were just about to watch Game of Thrones.
[PERSON 1:] Nice ! How old are your children?
[PERSON 2:] I have four that range in age from 10 to 21. You?
[PERSON 1:] I do not have children at the moment.
[PERSON 2:] That just means you get to keep all the popcorn for yourself.
[PERSON 1:] And Cheetos at the moment!
[PERSON 2:] Good choice. Do you watch Game of Thrones?
[PERSON 1:] No, I do not have much time for TV.
[PERSON 2:] I usually spend my time painting: but, I love the show.

Team Names	Perplexity	Hits@1	F1
1. Hugging Face	16.28	80.7	19.5
2. ADAPT Centre	31.4	-	18.39
3. Happy Minions	29.01	-	16.01
4. High Five	-	65.9	-
5. Mohd Shadab Alam	29.94	13.8	16.91
6. Lost in Conversation	-	17.1	17.77
7. Little Baby	-	64.8	-
8. Sweet Fish	-	45.7	-
9. 1st-contact	31.98	13.2	16.42
10. NEUROBOTICS	35.47	-	16.68
11. Cats'team	-	35.9	-
12. Sonic	33.46	-	16.67
13. Pinta	32.49	-	16.39
14. Khai Mai Alt	-	34.6	13.03
15. loopAI	-	25.6	-
16. Salty Fish	34.32	-	-
17. Team Pat	-	-	16.11
18. Tensorborne	38.24	12.0	15.94
19. Team Dialog 6	40.35	10.9	7.27
20. Roboy	-	-	15.83
21. IamNotAdele	66.47	-	13.09
22. flooders	-	-	15.47
23. Clova Xiaodong Gu	-	-	14.37
Seq2Seq + Attention Baseline	29.8	12.6	16.18
Language Model Baseline	46.0	-	15.02
KV Profile Memory Baseline	-	55.2	11.9

Table 3: Automatic Metrics Leaderboard.

我们和哈工大-讯飞联合实验室共同提交了两个系统，一个生成，一个检索，分别取得了第三和第四的成绩。

④ 引入个性化信息提高对话生成的多样性

- 聊天机器人的目标是“像”人
- 人的一个特性是具备个性化
  - 属性信息
  - 说话风格
- 机器人能否模拟人的个性生成回复
  - 将个性化信息加入到对话生成模型中
  - 实现生成回复的个性化和多样性
- 发表于IJCAI 2019

Exploiting Persona Information for Diverse Generation of Conversational Responses

Haoyu Song[1], Wei-Nan Zhang[1,2], Yiming Cui[1,3], Dong Wang[3] and Ting Liu[1,2]
[1]Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Joint Laboratory of HIT and iFLYTEK (HFL), iFLYTEK Research, Beijing, China
{hysong, wnzhang, tliu}@ir.hit.edu.cn, {ymcui, dongwang4}@iflytek.com,

Methods	Engage.	Variety	Per. Detection
Seq2Seq	48.91%	26.07%	44.57%
CVAE	55.43%	88.04%	41.30%
Per.-Seq2Seq	64.13%	29.35%	69.57%
GPMN	67.39%	30.43%	65.76%
Oracle. Copy	63.26%	32.61%	72.83%
Per.-CVAE	**71.74%**	**90.22%**	**78.26%**

Table 3: Human evaluation results (N=5). As a reference, the persona detection rate of the sampled testing data (human) is 90.83%.

Pref. (%)	Per.-Seq2Seq	GPMN	Per.-CVAE
Per.-Seq2Seq	-	41.86	32.56
GPMN	58.14	-	39.53
Per.-CVAE	**67.44**	**60.47**	-

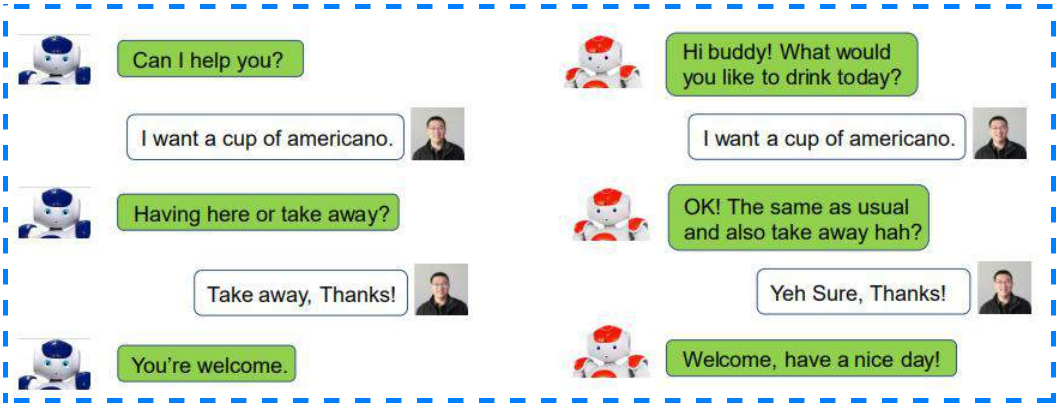
Table 4: Pairwise preference of three persona-based models (N=5).

Haoyu Song, Wei-Nan Zhang, Yiming Cui, Dong Wang, Ting Liu. Exploiting Persona Information for Diverse Generation of Conversational Responses[C]. IJCAI 2019

36

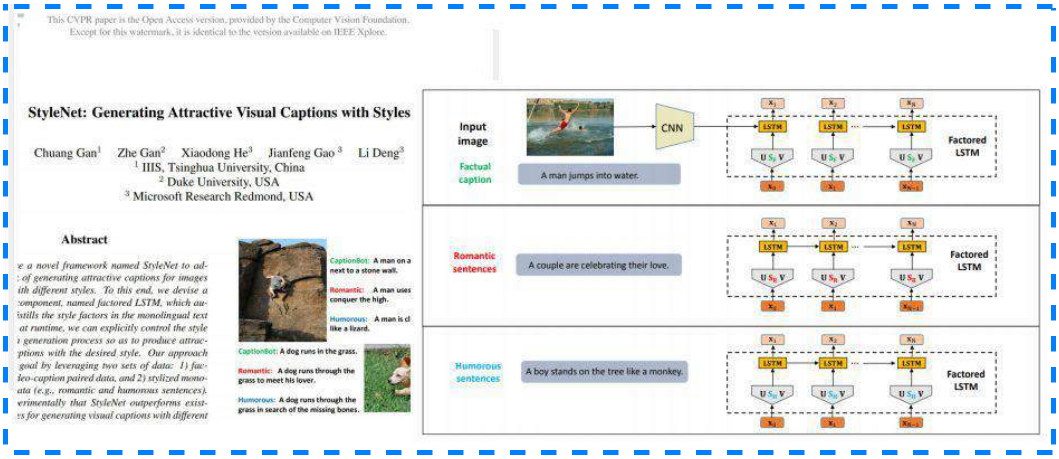
基于参赛评测，我们也做了一些工作：1. 引入个性化信息，提高对话生成多样性的过程。类似于机器阅读理解，每次回复的是描述用户个性化信息的自然语言的片段。我们也提出了 Per.-CVAE 的模型，将 persona 作为一个 condition 加入到原有的 CVAE 模型中，效果得到了较大的提升；2. 风格化的回复，语言的风格和内容是高度相关的，不像图像的风格和内容是能够分开的，我给你一幅河边的景色图，无论转化成毕加索风格还是梵高风格，一眼就可以辨识出来，语言的风格和内容实际上是绑定的，例如阿里巴巴换成亲，甄嬛传换成小主，换了一个词就换了一种风格。语言的风格和内容实际上是一个绑定的过程。

⑤ 风格化回复的直观印象



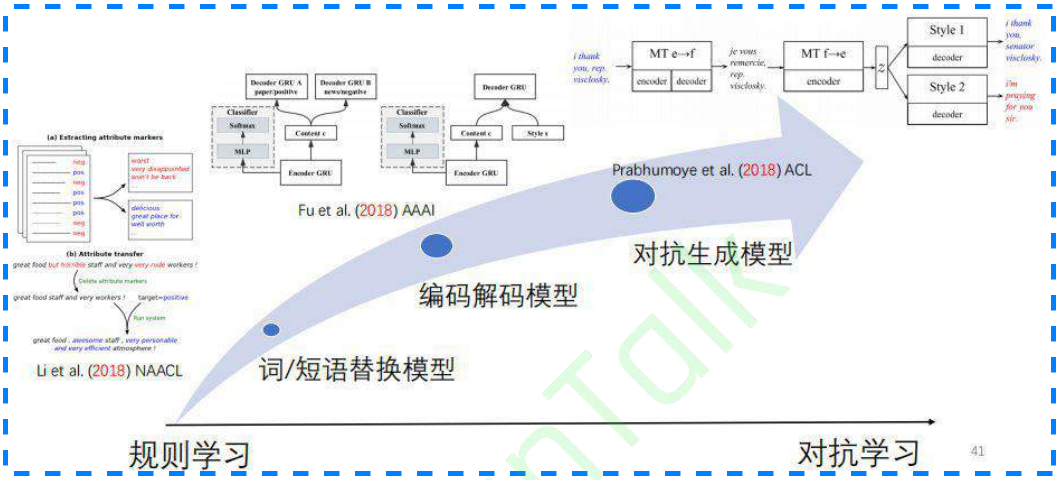
我们可以有一个直观化的比较，比如星巴克有聊天机器人可以帮忙点咖啡，下面两种点餐方式达到的目的是一样的，但是用户体验却不一样，之所以带来用户体验的不同是因为机器人说话风格存在差异。

⑥ 已有风格化文本生成工作



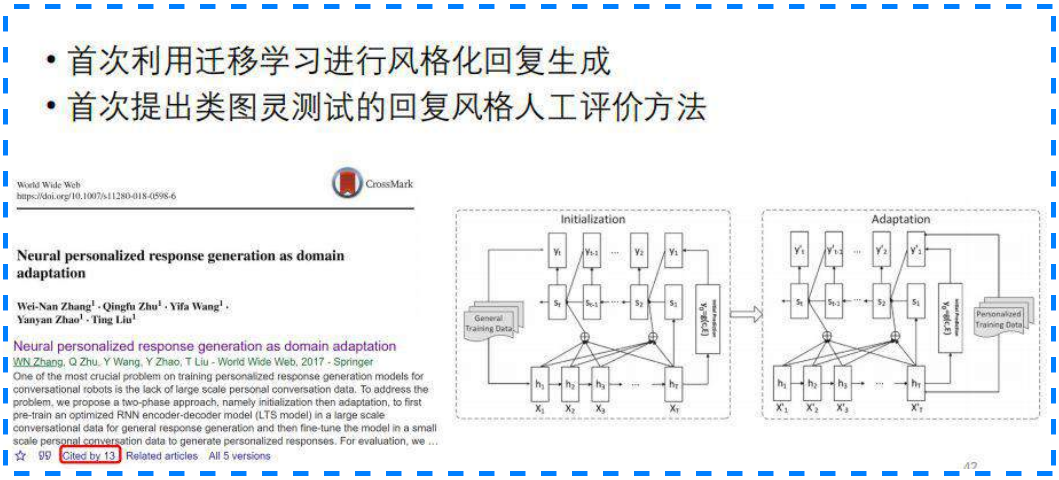
Style transfer 不仅限于文本之间，很可能利用图片信息，通过图片的语义信息生成不同风格的 image caption 这样一个过程。

⑦ 风格化生成技术



从 18 年开始有对话上面生成风格化的技术，分别发表在 NAACL，AAAI 和 ACL 上，整个过程是从规则学习到对抗学习。

⑧ 风格化回复：一个初步的尝试

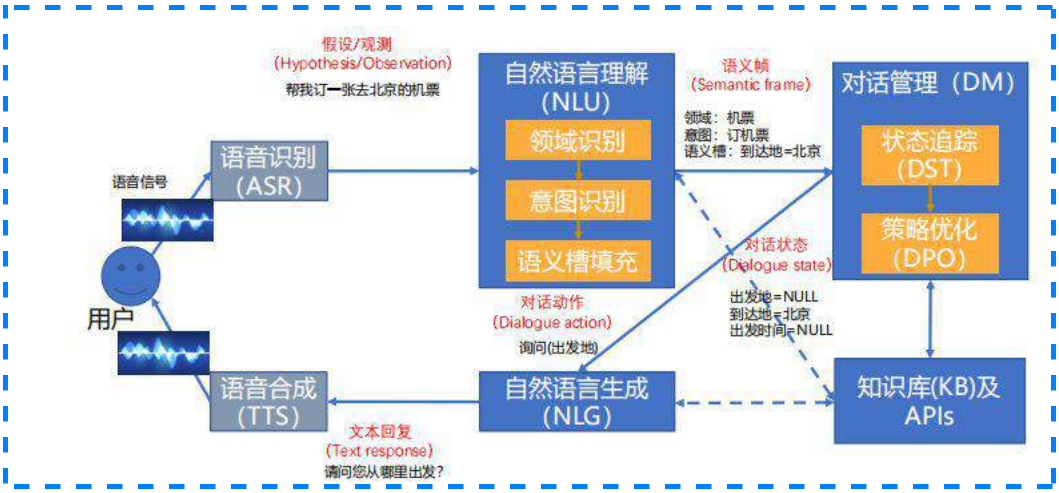


17 年的时候，我们做了一个工作，利用迁移学习进行风格化回复生成，最开始的想法是能不能用一个 bot 去个性化模拟一个人，但是模拟一个特征点可能不太合适，所以我们就模拟一个 CS 背景和一个人文背景不同类别学生之

间的差异。我们仍然面临这样一个问题，具有个性化和风格化的数据是很少量的，我们怎样在很少量的数据上学习出它的风格，实际上采用了一个二阶段的迁移学习，一个 initialization 和一个 adaptation 的过程。Initialization 是用大量的百万级的无风格数据来训练，然后在少量有风格化的数据上做一个 adaption 的过程。

二．任务型对话系统

任务型对话系统的架构：



主要过程就是自然语言理解到对话管理到自然语言生成到语音合成。这其中有对知识库 look up 的一个过程，当然知识库不仅在对话管理进行交互，它也影响到自然语言理解和自然语言生成的过程。

前沿研究点：

这里我介绍一下我们中心车万翔教授团队的两个工作，是任务型对话自然语言理解方面，主要针对两个比较大的问题。

1. 意图识别与槽填充能不能联合建模

2. 在工业型部署系统中，经常会遇到领域的一个迁移，比如今天做母婴领域，明天会做其它领域，或者说我今天卖鞋，明天卖冰箱。这种领域上的迁移不可能每次都找很多人来进行标注，这种情况下就面临着标注数据不足条件下的 NLU 问题。

1. 意图识别与槽填充联合建模

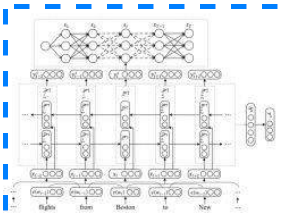
- 意图识别与槽填充不是相互独立而是紧密联系的
- 传统独立的建模意图识别和槽填充，既会引出错误级联，也无法利用共有的知识。
  - 例如：如果这句话意图是 WatchMovie，那么这句话包含的Slot槽值应该是电影相关而不是音乐相关

Sentence	*watch*	*action*	*movie*
Gold Slots	O	B-movie_name	I-movie_name
Gold Intent	WatchMovie		

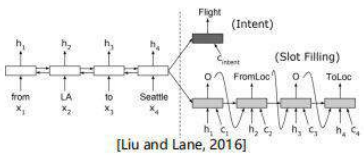
首先是意图识别与槽填充联合建模的问题，这个事件的动机比较明确，如果当作级联建模会存在一个错误结果累积的过程，显然我们可以把它联合建模，domain，intent，slot 都可以联合建模，这里只讨论 intent 和 slot 如何联合建模，这方面有很多之前的工作，包括 16 年开始的一个联合建模的工作，是把 intent 的识别和序列标注的过程当作多任务学习的过程，它们共享的是

编码的双层 LSTM 过程。16 年还有一个工作是把序列标注当作一个 sequence2sequence 的过程，这里 intent 和 slot filing 也是共享编码这样一个结构，它们两个模型上虽然有差别，但是其实创新上没有特别大的差别，从上到下，从左到右分别对应 1 到 4 这 4 个工作，上面两个是隐式建模过程，下面两个是显式建模过程。

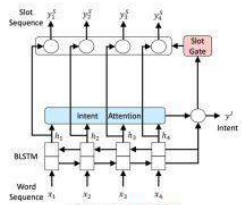
已有工作：



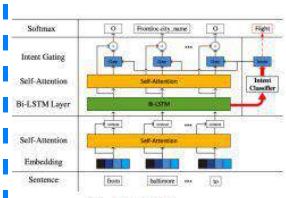
[Zhang and Wang, 2016]



[Liu and Lane, 2016]



[Goo et al, 2018]



[Li et al, 2018]

Model	Intent-Slot Relationship
Joint GRU [1]	Implicit
Attentional Encoder-Decoder [2]	Implicit
Slot Gate Joint Model [3]	**Explicit**
Self-Attentive Model with Gate Mechanism [4]	**Explicit**

这是现在我们正在做的一个工作，我们现在已有的工作都是输入一句话，我们给这句话打一个唯一的 intent 的标签，但在实际中，比如我们在用户建模 encode 一句话的时候，我想买一个漂亮的文具盒还是手机，这里就有两个不同的 intent 甚至 domain 都不一样，实际上 intent 应该打到词级别上。我们在建模一句话的时候，不同的词有不同的 intent，对最终 intent 的贡献也不一样，那么基于这样一个假设，我们把 intent 打到每个 token 上。

实验结果：

Model	SNIPS			ATIS		
	Slot (F1)	Intent (Acc)	Overall (Acc)	Slot (F1)	Intent (Acc)	Overall (Acc)
Joint Seq (Hakkani-Tür et al., 2016)	87.3	96.9	73.2	94.3	92.6	80.7
Attention BiRNN (Liu and Lane, 2016)	87.8	96.7	74.1	94.2	91.1	78.9
Slot-Gated Full Atten (Goo et al., 2018)	88.8	97.0	75.5	94.8	93.6	82.2
Slot-Gated Intent Atten (Goo et al., 2018)	88.3	96.8	74.6	95.2	94.1	82.6
Self-Attentive Model (Li et al., 2018)	90.0	97.5	81.0	95.1	96.8	82.2
Bi-Model (Wang et al., 2018b)	93.5	97.2	83.8	95.5	96.4	85.7
DAT (Wang et al., 2018a)	92.4	-	-	95.6	-	-
Our model	**94.9***	**98.3***	**89.1***	**95.9***	**97.4***	**87.5***
Oracle (Intent)	96.1	-	-	96.0	-	-

Model	SNIPS			ATIS		
	Slot (F1)	Intent (Acc)	Overall (Acc)	Slot (F1)	Intent (Acc)	Overall (Acc)
Our model	94.9	98.3	89.1	95.9	97.4	87.5
Intent detection (BERT)	-	97.8	-	-	96.5	-
Slot filling (BERT)	95.8	-	-	95.6	-	-
Our model + BERT	97.0	99.1	93.0	96.2	98.2	88.6

实验结果显示我们取得了不错的结果，尤其在加了 Bert 之后效果更好。

2. 标注数据不足条件下的 NLU

下一个工作就是我们说到的新领域的问题，面临标注数据不足条件下的 NLU 的问题。

标注数据的自动扩充：

一种方式就是我们造一些伪数据，给定少量用户标注数据，通过已有数据来造一些模板，利用模板将 slot 的值填进去，这样就能生成和当前语义上差不多但形式上不同的数据。

当然我们最主要的一个过程就是相似性排序，最主要的过程除了抽模板，要有一个多样性，是一个 Diversity ranking 的过程，这个 Diversity ranking 的动机也比较简单，就是你给我一句话，我们这两个语义是相似的，但是差一个词，对我这个模型其实是不好的，但是如果表面 surface 差的非常多，但是还表示一样的语义，这样的模板对我们后续的扩展是非常有用处的，所以基于这样一个方式我们提出一个 Diversity ranking 的过程，整个生成过程是一个 seq2seq 的过程。但是我们生成的不是完整的带 slot 值和 value 值，而是带槽的模板，最终我们会把槽和值填回去。

实验结果：

最终我们在 ATIS 数据集和斯坦福对话数据集上都取得了一定的效果提升。

三. 总结

最后对分享的内容进行一下总结，分为两方面：一方面是开放域人机对话，另一方面是人机对话。除了讲的内容之外，列出的主要还是一些挑战。比如

开放域人机对话中怎样建模情感，用户画像，怎么保证回复的一致性、多样化、个性化、风格化，另外就是上下文建模、隐式反馈我们怎样进一步挖掘。在任务型对话上也有几个比较大的关键点，领域、意图、语义槽三者能不能联合建模，多轮语境下的自然语言怎么去理解，最后是数据自动扩充 v.s. 小样本学习方向抉择问题。

四. Demo

我们实验室的名称是哈尔滨工业大学社会计算与信息检索研究中心，中心的主任是刘挺教授。

我们团队的主要研究方向是：1. 对话理解与对话生成；2. 对话式推荐。

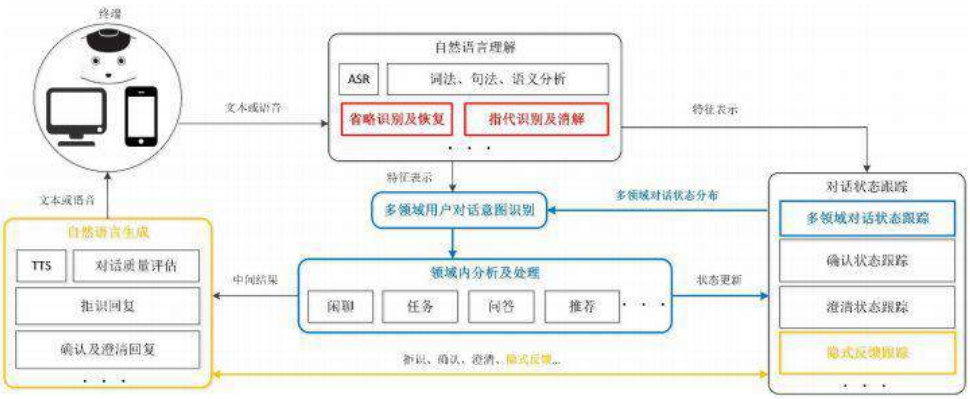
我们实验室将技术做成了一个应用级的 demo，在上面进行持续优化，把所有的技术都应用在上面。16 年 6 月 6 日，人机对话系统“笨笨”正式上线，主要搭载形式是微信公众号以及实体的机器人。主要包括四个功能：闲聊、问答、任务型对话和推荐。

智能人机对话系统“笨笨”

- HIT-SCIR-CI组主导研发
- 平台
  - 微信公众号：2016年6月6日上线
  - 实体机器人
- 主要功能
  - 闲聊
  - 问答
  - 任务型对话
  - 推荐

笨笨的系统结构如下，分为三大部分：自然语言理解、对话管理和自然语言生成。

笨笨系统结构



Zhang W N, Liu T, et al. Benben: A Chinese Intelligent Conversational Robot[C]// ACL 2017, System Demonstrations. 2017:13-18.

下面给出了一些聊天示例，包括情绪抚慰性的聊天，情绪引导性的聊天，知识性问答，任务执行示例和推荐示例。其中任务执行主要包括机票、火车票、公交车、酒店信息查询等。

“聊天”示例

我学习压力很大
放松自己，别给自己太大压力哦！
怎么才能放松自己？
把心放宽，一觉睡到大天亮！
可是我睡不着
不要想那么多啦，不能有太大的压力，一切都会好起来的！
好吧，谢谢你
举手之劳罢了，不用谢的！
我失恋了。。。
不要因为结束而哭泣，微笑吧，为你的曾经拥有……
他为什么不爱我了
这个问题我也没办法啊，我还是一个菜鸟。
有什么烦心事可以跟我聊聊吗。
我现在只想哭
我给你个大笑脸，你来跟着学习一下吧。

“知识问答”示例

今天天气怎么样？
哪个城市的天气？
哈尔滨的
哈尔滨，现在的天气：晴，温度：22°C。数据由OpenWeatherMap在2017-05-17 21:00提供。
北京天气怎么样？
北京，现在的天气：晴，温度：27°C。数据由OpenWeatherMap在2017-05-17 21:00提供。
刘德华的生日
1961年9月27日(农历八月十八)
中国的首都
北京
鲁迅的故乡
浙江省绍兴府会稽县(今浙江省绍兴市绍兴县)
新加坡国土面积
719.1平方公里
2008奥运吉祥物
迎迎有五个，欢欢,分别是贝贝、晶晶，合起来就是北京欢迎你。呢呢。宁波有出售的

“任务执行”示例

要去北京参加CCF青年精英大会
请问您需要我查询出行方案吗?
好
我能查飞机票、火车票和公交车，您要看哪个呢?
看看火车吧
您要从哪里出发呢?
哈尔滨
您想要哪一天出发呢?
明天就走
火车票信息
5月18日
到北京得订个酒店啊
您想要哪天入住呢?
明晚呗
酒店信息
5月18日
以下是我找到的5月19日北京地区的酒店信息
幽谷神谭春立农家院 怀柔区怀北镇蚂蚁树岭村黄土梁自然村28号
标准间（公共卫生间） [双床] 含早餐
家庭间 [双床] 含早餐
五人间 [其他] 含早餐
8人间 [其他] 含早餐
套房 [其他] 含早餐
水长城西水湾欢乐度假村 怀柔区九渡河镇石湖峪民俗村西水湾欢乐度假村
标准间A [双床] 含早餐
标准间B [双床] 含早餐
三人间A [三人间] 含早餐
三人间B [三人间] 含早餐
水长城游玩+特色套餐+床位房 [其他] 含早餐
水长城游玩+特色套餐+真人cs+床位房 [其他] 含早餐
水长城游玩+篝火+烧烤+特色套餐+床位房 [其他] 含早餐
挂甲峪老付农家院 平谷区挂甲峪新村6排6号
大床房 [大床] 含早餐

基于机器阅读理解（MRC）的信息抽取方法

嘉宾：周明昕　编辑：蔡英杰

导读：本次分享的主题为基于机器学习阅读理解（MRC）的信息抽取方法。由香侬科技发表在 ACL2019，论文提出用阅读理解的方法对信息抽取这个传统问题进行建模，取得了比较好的效果。

Part1 香侬科技简介

香侬科技的主要研究方向为自然语言处理和深度学习，秉承“让机器读懂纷繁的大千世界”为使命，致力于用人工智能技术消除信息不对称的壁垒，让所有人都能平等获取信息。目前主要业务包括信息抽取、智能文档解析、智能数据聚类分析、文本对比审核等。

Part2 自然语言处理（NLP）

1. 自然语言处理（NLP）是什么？

自然语言处理是研究能实现人与计算机之间，用自然语言进行有效通信的各种理论和方法。（通俗讲就是计算机要理解人说的话，计算机生成的话，人也能够理解）

DataFunTalk

2. 为什么自然语言处理很难？

自然语言处理的难点在于**歧义性**。如下面的例子：

同样的表达方式，有不同的含义：

“喜欢一个人”：like someone 或者 like to be alone。

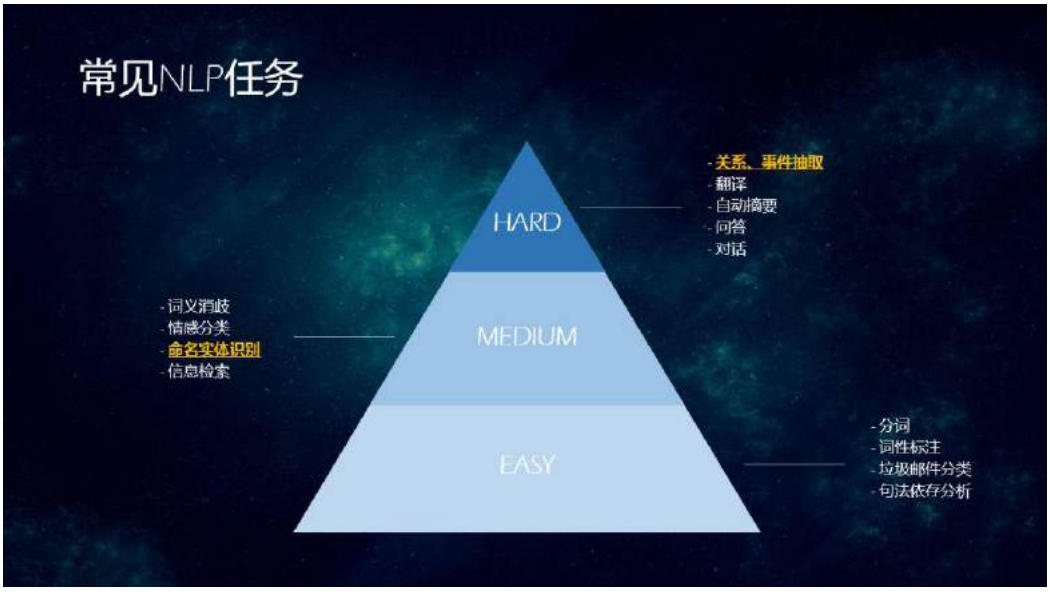
虽然表达方式不同，但表达的含义相同：

“我刚差点摔着”、“我刚差点没摔着”

“苹果多少钱一斤”、“苹果一斤多少钱”、“一斤苹果多少钱”

3. 常见的 NLP 任务

DataFunTalk



常见的 NLP 任务可以分为简单、中等、困难，分别包含的具体任务如下：

简单任务：分词、词性标注、垃圾邮件分类、句法依存分析

中等任务：词义消歧、情感分类、命名实体识别、信息检索

困难任务：关系&事件抽取、翻译、自动摘要、问答、对话

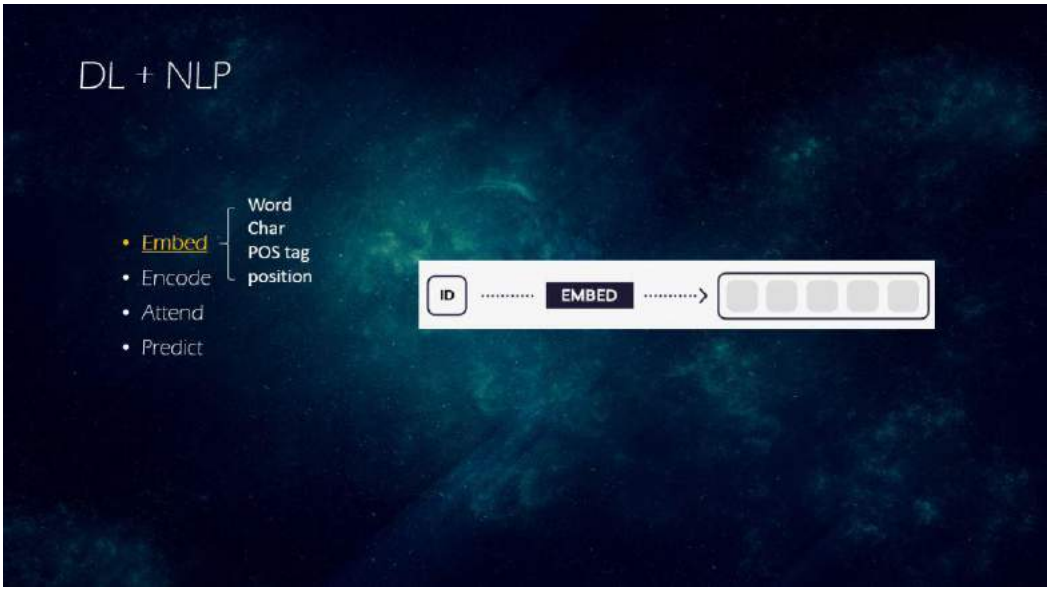
今天主要介绍命名实体识别、关系&事件抽取这两类信息抽取任务。

4. 深度学习与 NLP 结合的范式：

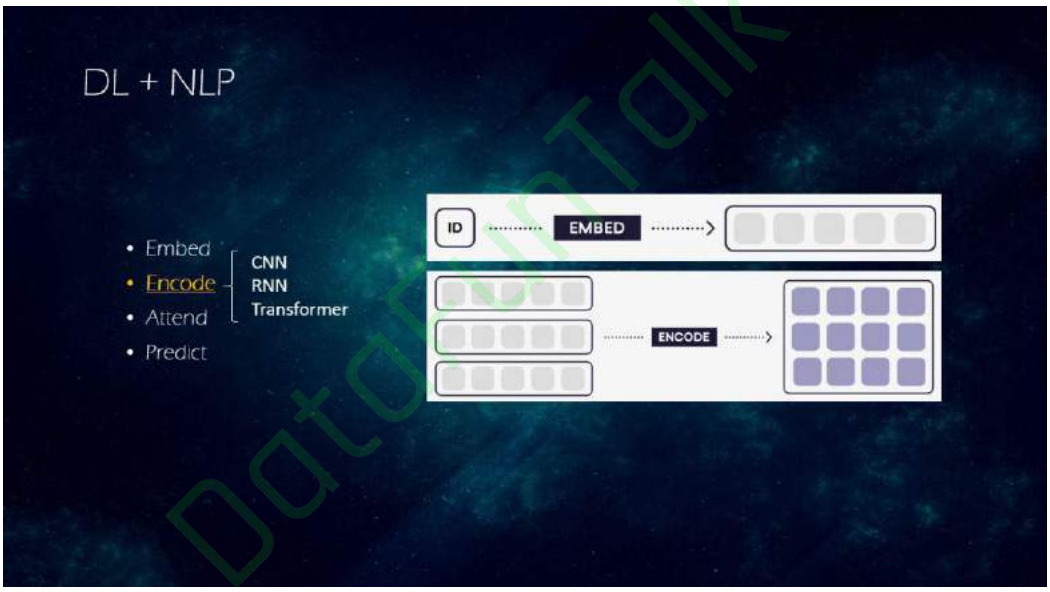
范式 1：

DataFunTalk

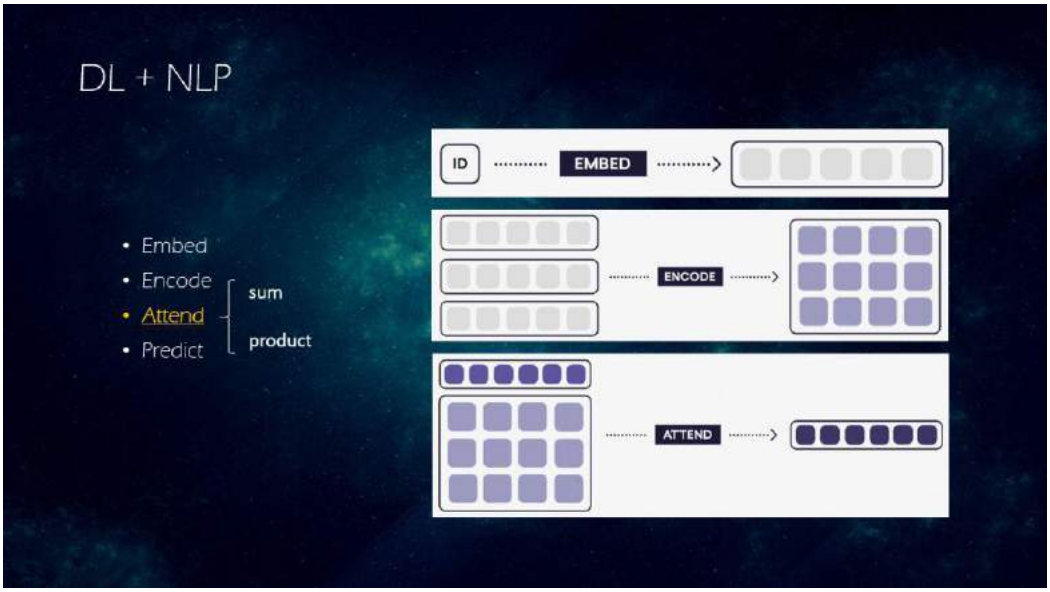
第一种深度学习与 NLP 的结合的范式（Bert 出现之前），一般包含以下几步：



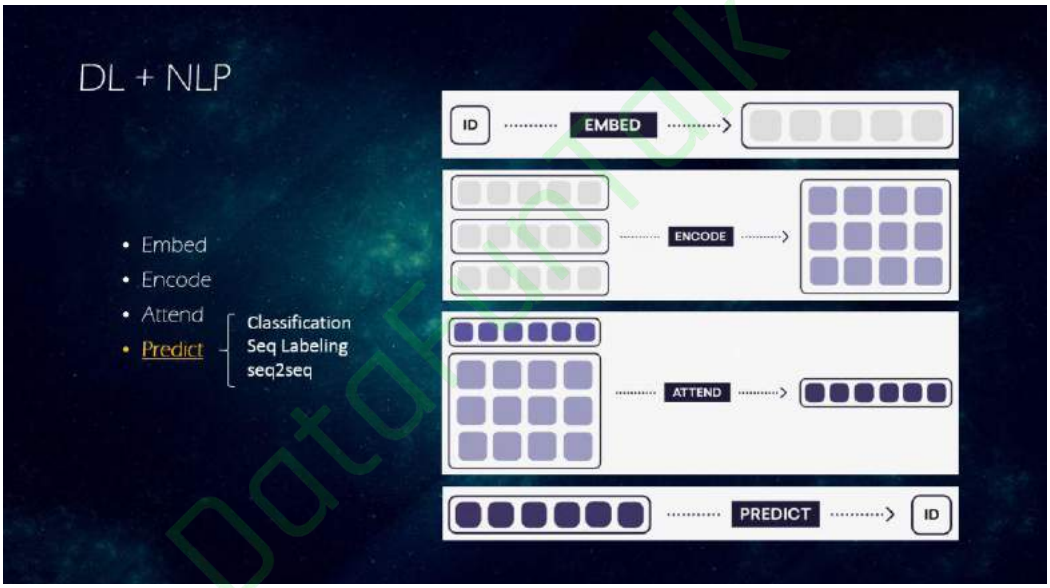
-> Embed：该阶段把离散 id 变成稠密向量，可能对以下几个维度特征进行 embedding ,包括 word(单词)、char(字符)、pos tag(词性)、position(位 置)。



-> Encode：该阶段使得稠密向量获得上下文信息， 一般方案包括CNN/RNN/Transformer。



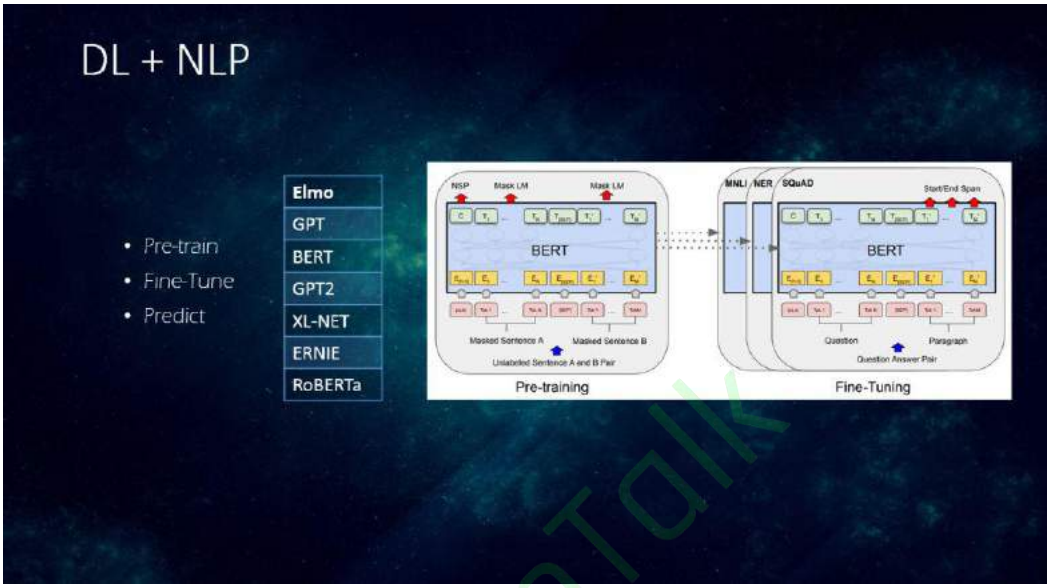
-> Attend：该阶段可以使更远的信息可以互相注意，一般分为加性(sum)、乘性(product)的注意力机制。



-> Predict：NLP 任务一般是预测离散的变量，Classification 分类、Seq Labeling 序列标注、翻译/摘要可以 model 成 seq2seq 的任务。

范式 2：

最近有些 Pre-train 的模型出现，打破的了之前的方案，把 embed 到 Attend，基本上都用的类似的方法，通过 Pre-train 得到一个带着上下文信息的向量。算法包括 elmo、gpt、bert、gpt2、xlnet、ernie、roberta 等，在大数据量上进行预训练。



第二种深度学习与 NLP 的结合的范式，一般包含以下几步：

-> Pre-train：刚开始是用大规模的语料进行预训练。

-> Fine_tune：针对特定任务进行微调。

-> Predict：进行 Predict。

Part3 信息抽取

1. 信息抽取简介

信息抽取是将非结构化的文本信息转换为结构化的信息，如上图所示，左边为非结构化文本，通过信息抽取得到右边结构化的信息。

DataFunTalk

信息抽取可以分为三类：

命名实体识别（NER）：从文本中抽取人物、机构、文件名等实体。

关系抽取（RE）：得到实体后，抽取实体间的关系，一般抽取得到一个（实体，实体，关系）的三元组。

事件抽取：抽取多个实体和事件关系，一般得到一个 trigger 和多个 arguments 的多元组。

今天主要介绍命名实体识别和关系抽取。

2. 实体关系抽取

DataFunTalk

大量的信息抽取任务是实体和关系的抽取，举个例子，乔布斯是苹果公司的CEO。其中"乔布斯"和"苹果公司"是实体，分别是人名、公司名；两个实体之间是雇佣关系。

3. 实体关系抽取的传统做法

实体关系抽取的**传统做法：**

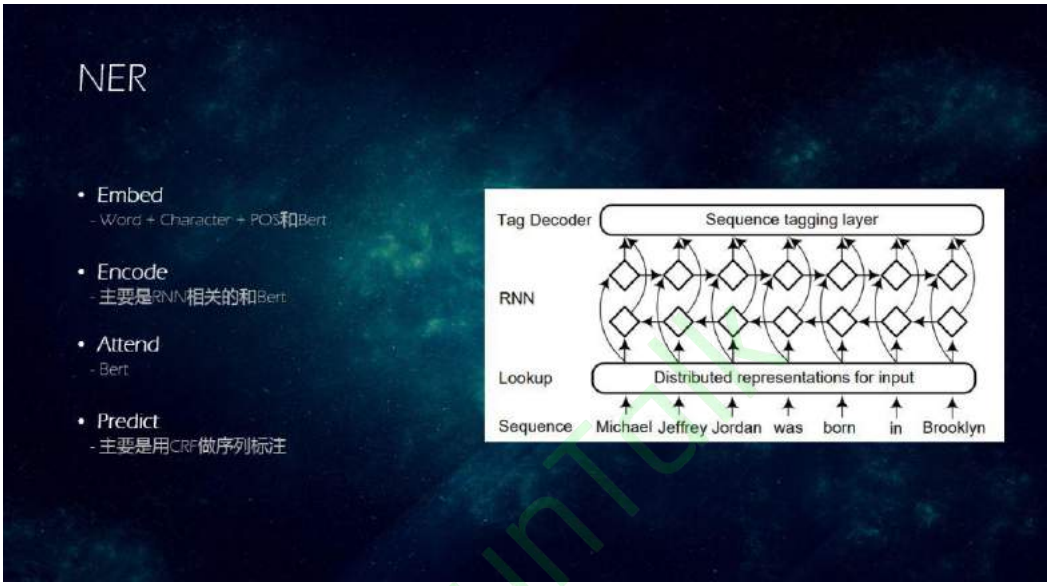
Pipeline 的方法：

先进行命名实体识别，再进行关系识别。

joint 方法：

通过共享参数的模型，把实体识别和关系识别联系到一起。

命名实体识别（NER）：



NER 也可以分为 4 步：

-> Embed：把 Michael Jeffrey Jordan was born in Brooklyn，变成一个稠密向量。

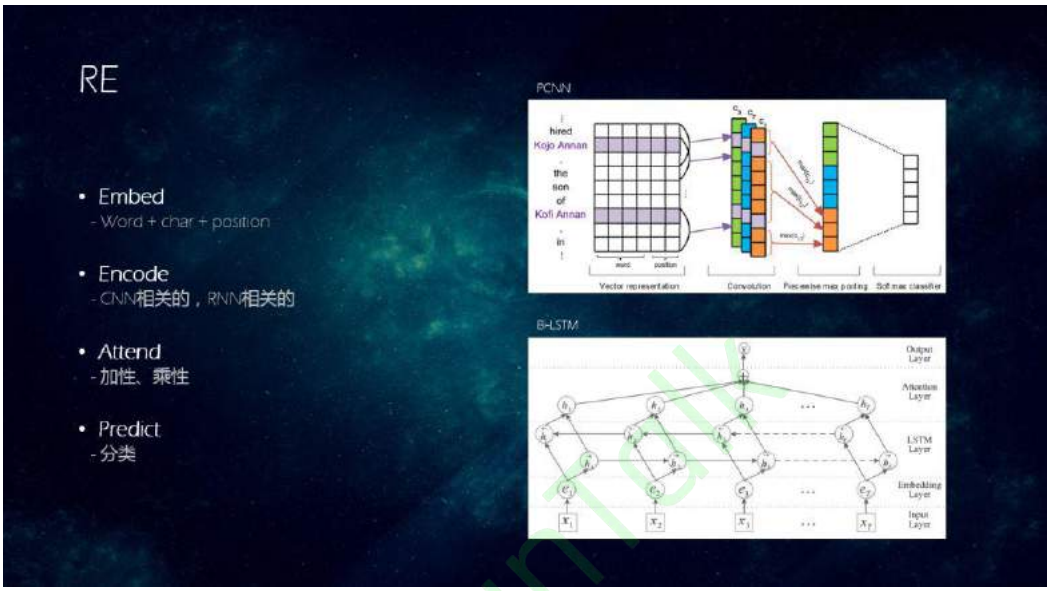
-> Encode：在 NER 中主要是 RNN 相关的；

-> Attend：把注意力集中在“一个人在布鲁克林出生”

-> Predict：主要用 CRF 做序列标注。

也可以用 BERT，把 Embed 到 Attend 之间都用过 BERT 来初始化，然后进行 fine_tune，然后在最上层加一个 Predict 的神经网络。

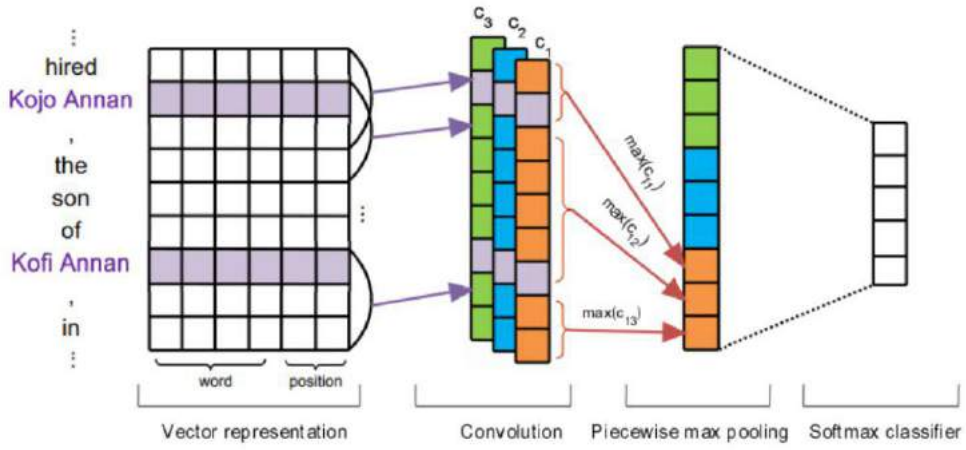
关系抽取（RE）：



RE 一般被 model 成 Classification 任务，同样分为以下 4 步：

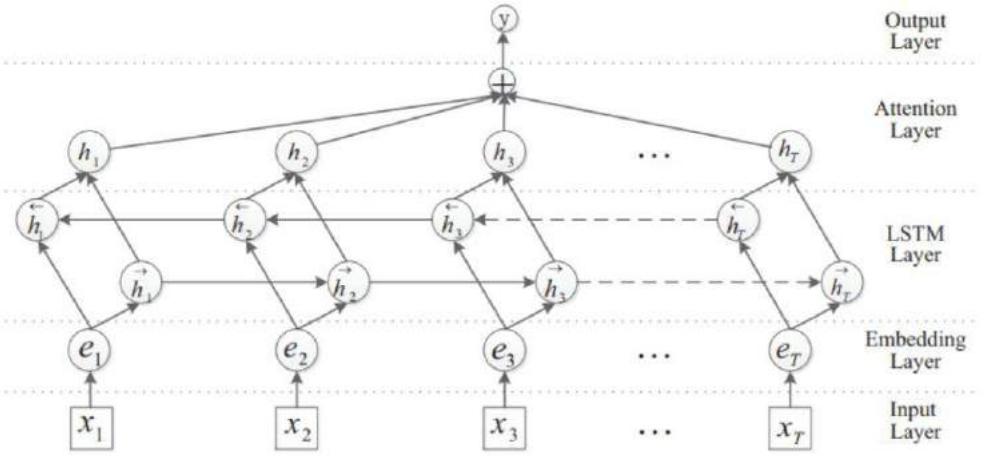
-> Embed：还是将字或者词变成一个稠密的向量，这里还会多一个 position，一般认为待变成稠密向量的词与实体之间的位置是一个相对位置，如下图中

hired 和第一个人它的位置可能是-1，然后逗号和它的位置是 1，这样的一种 Embedding。



-> Encode：对于关系识别来说，即包含 CNN 相关的，也包含 RNN 相关的，把上下文的信息考虑进来。如上图是一个 PCNN，通过两个实体把句子分为三部分，每部分分别做 CNN 的 max pooling（由于这三部分的用处不同，直接做 max pooling 可能会有一些信息的损失），再经过 Softmax classifier 进行分类。

DataFunTalk



上图为 BI-LSTM，同样用了 RNN base 的网络结构，可以学习到更长距离的依赖。

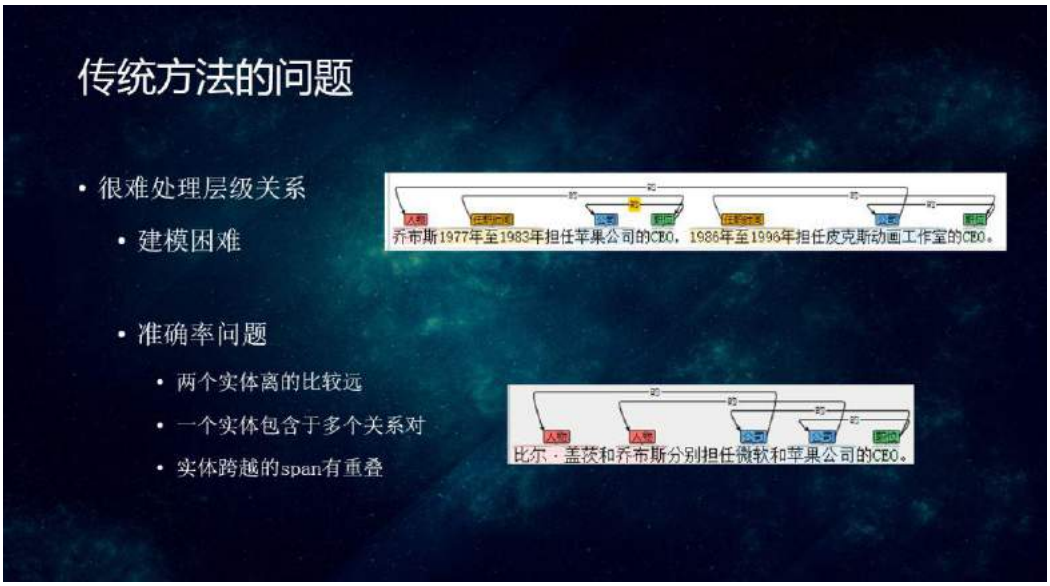
-> Attend：Attention 可以 Attend 到两个实体是由于哪个词分辨出是顾佣的关系，比如“CEO”就可以认为“苹果公司”和“乔布斯”之间是雇佣关系，有比较大的一个权重。

-> Predict：对定长的向量进行 Predict 分类。

刚刚说的都是 Pipeline 的方法，Joint 方法也是类似的，主要是把两部分的一些模型的参数在底层被共享起来。

4. 传统方法的问题

DataFunTalk



Pipeline 和 Joint 两种方案都存在一些问题：

① 很难处理层级关系：建模困难。如右侧第 1 张图所示，“乔布斯在 1977 年至 1983 年担任了苹果公司的 CEO ,在 1986 年至 1996 年担任皮克斯动画工作室的 CEO”。这里存在 7 个实体，相互间都有关系，非常复杂。

② 准确率问题：

两个实体离得比较远，尤其当一个实体与另外一个实体发生关系时，其实是它的缩写与另外一个实体比较近，而本体与另外一个实体比较远的时候，这样两个实体之间的关系往往很难预测得到。

一个实体包含多个关系对，比如这里的“苹果公司”，既和“CEO”产生关系，又和“乔布斯”产生关系， 包含多个关系，是关系的分类比较复杂。

实体跨越的 span 有重叠。如右侧第 2 张图所示，“比尔盖茨和乔布斯分别担任微软和苹果公司的 CEO”，“比尔盖茨”是和“微软”产生关系，“乔布斯”是和“苹果公司”产生关系对于刚刚说的 PCNN 或者 RNN 来说都会产生混乱的情况，导致识别准确率下降。

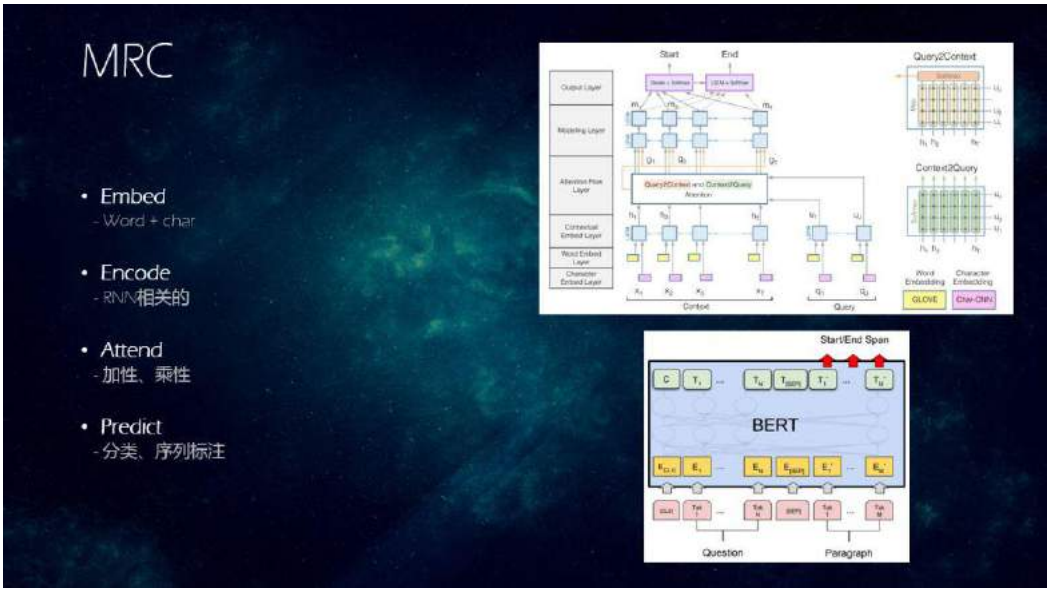
Part4 机器阅读理解（MPC）

1. 机器阅读理解简介

机器阅读理解是给定上下文 c，和问题 q，得到答案 a，a 是原文中的一个或多个 span<Question,Context> -> <Answer>。

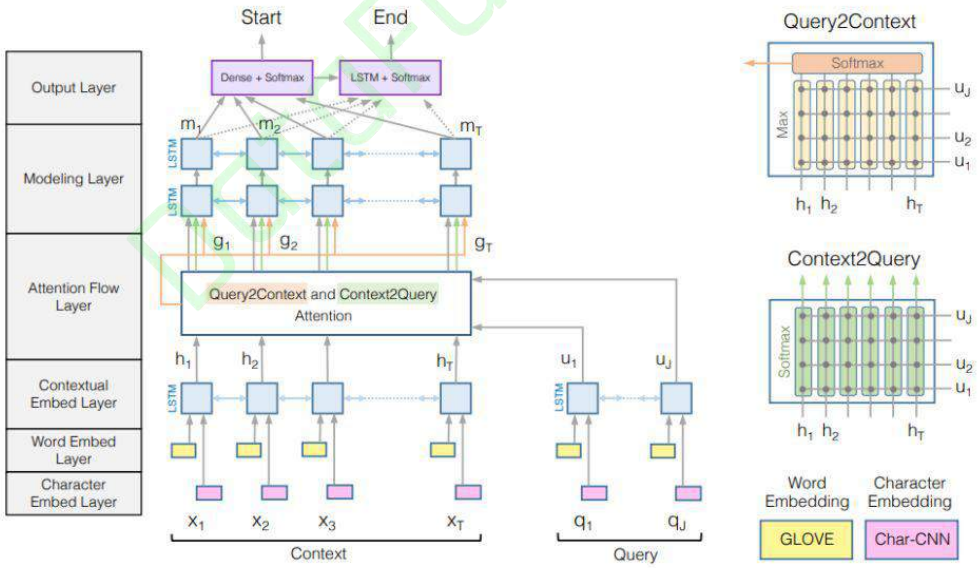
如图所示，通过提问“郑强什么时候就职于越秀会计事务所？”，我们可以发现，是在 1998 年到 2000 年时就职于越秀会计事务所。这样就得到郑强、越秀会计事务所、时间等多个实体间关系。

2. MRC 做法



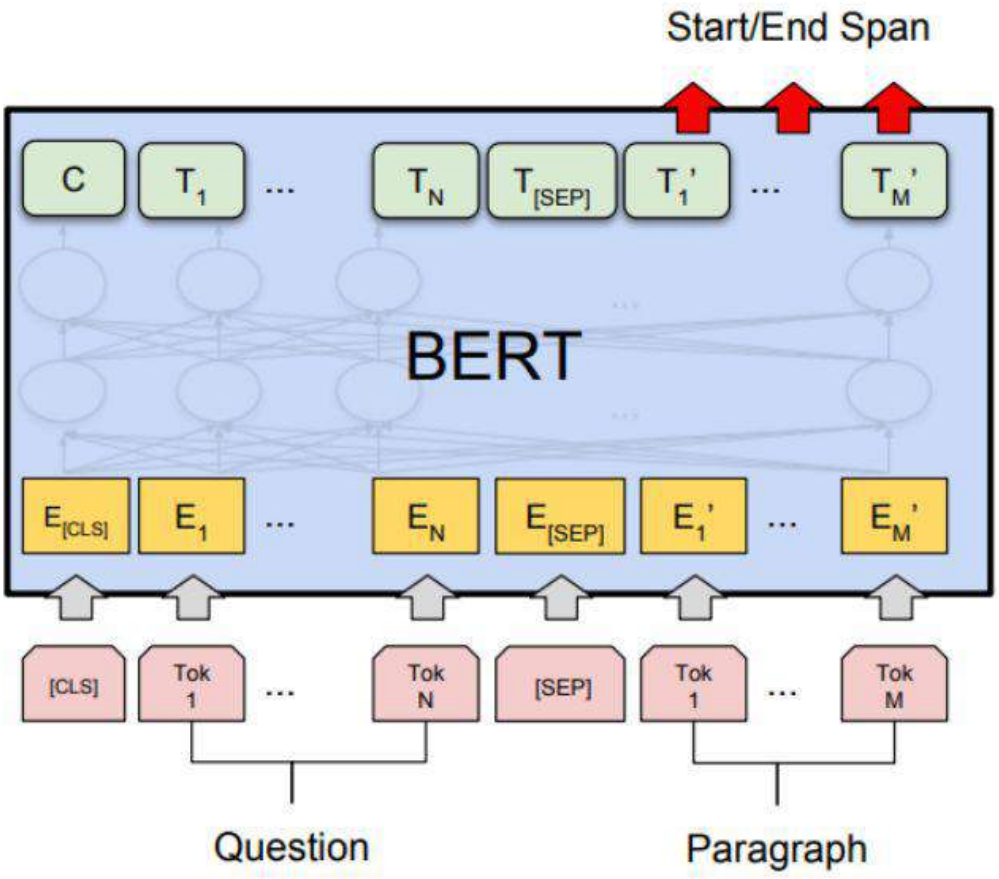
MRC 的做法也是分为 4 步：

范式 1，传统的方案：



先 Embed，把字和词变成稠密向量，通过 Contextual 进行 Encode，这里通过的是 LSTM 或者 GRU 之类的方案，通过上下文一起考虑进来，然后进行 Attend，question 和文章互相注意，最后是 Predict，进行分类和序列标注。

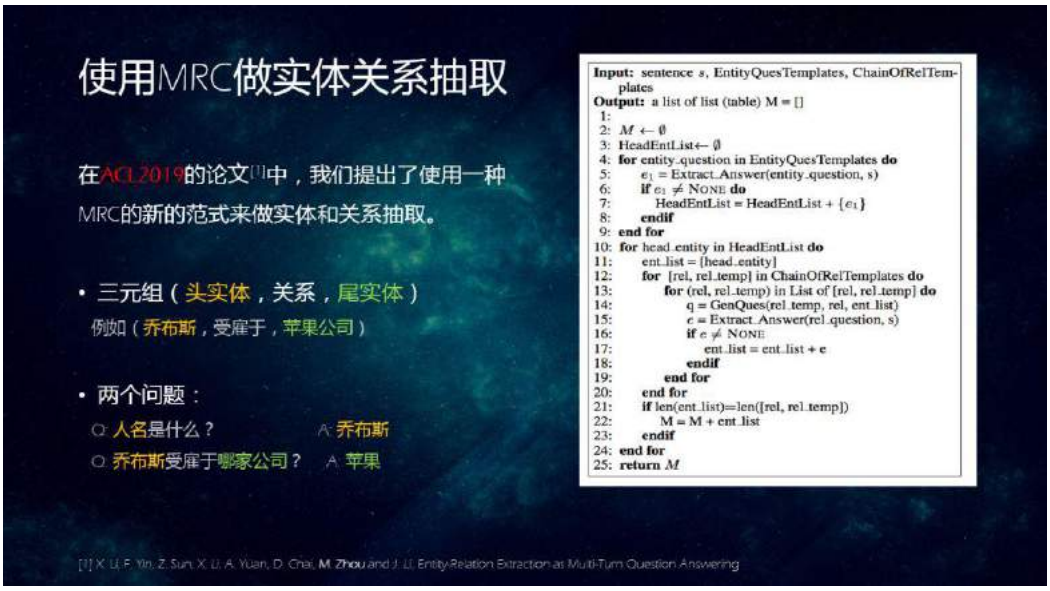
范式 2：



Bert 出现之后，可以通过 Bert 方案，把前 3 步换成 Bert，最后预测一个 start 和一个 end，或者通过序列标注的方案来做。

如上图，Bert 时，会在 Question 的 Tok 前加一个 CLS，后面加一个 SEP，然后把 Paragraph 的 Tok 放在后面。然后预测 Paragraph 中哪位置的 Tok 是 start，哪个位置的 Tok 是 end，来预测阅读理解问题。

3. 使用 MRC 做实体关系抽取



在 ACL2019 的论文中，我们提出了使用一种 MRC 的新的范式来做实体和关系抽取。

三元组：（头实体，关系，尾实体）例如（乔布斯，受雇于，苹果公司）

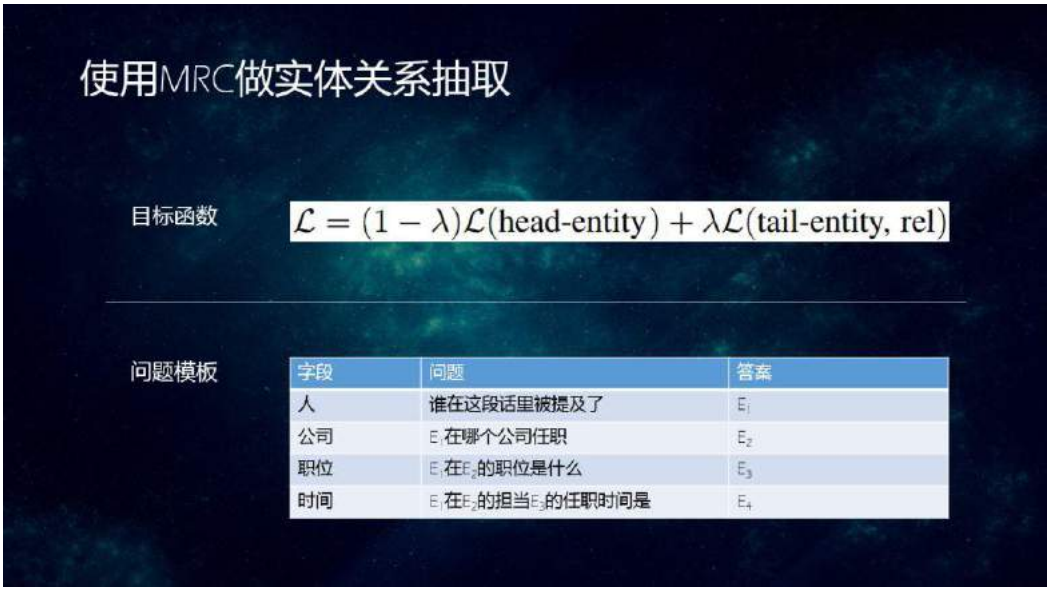
对于“乔布斯是苹果公司的 CEO”可以提两个问题：

① 首先提的是头问题，Q：人名是什么？A：乔布斯

② 基于上一步的答案提出第二个问题，Q：乔布斯受雇于哪家公司？A：苹果

```
Input: sentence s, EntityQuesTemplates, ChainOfRelTem-
    plates
Output: a list of list (table) M = []
 1:
 2: M ← ∅
 3: HeadEntList← ∅
 4: for entity_question in EntityQuesTemplates do
 5:     e1 = Extract_Answer(entity_question, s)
 6:     if e1 ≠ NONE do
 7:         HeadEntList = HeadEntList + {e1}
 8:     endif
 9: end for
10: for head_entity in HeadEntList do
11:     ent_list = [head_entity]
12:     for [rel, rel_temp] in ChainOfRelTemplates do
13:         for (rel, rel_temp) in List of [rel, rel_temp] do
14:             q = GenQues(rel_temp, rel, ent_list)
15:             e = Extract_Answer(rel_question, s)
16:             if e ≠ NONE
17:                 ent_list = ent_list + e
18:             endif
19:         end for
20:     end for
21:     if len(ent_list)=len([rel, rel_temp])
22:         M = M + ent_list
23:     endif
24: end for
25: return M
```

上图为，我们算法的详细流程图，首先得到 head Entity 头实体，对于头实体提问完之后，可以得到头实体的答案，把头实体的答案填入问题模板中，再获得后面的实体。



目标函数就是 head-entity 的 Loss+tail-entity 的 Loss ,然后进行一个加权。

问题模板如上图所示，对于郑强或者马斯克的例子，我们可以问四个问题：

第一个问题是谁在这段话里被提及了，也就是马斯克或者郑强。

第二个问题是郑强在哪个公司任职，得到很多公司。

第三个问题是郑强在这个公司的职务是什么，得到职务。

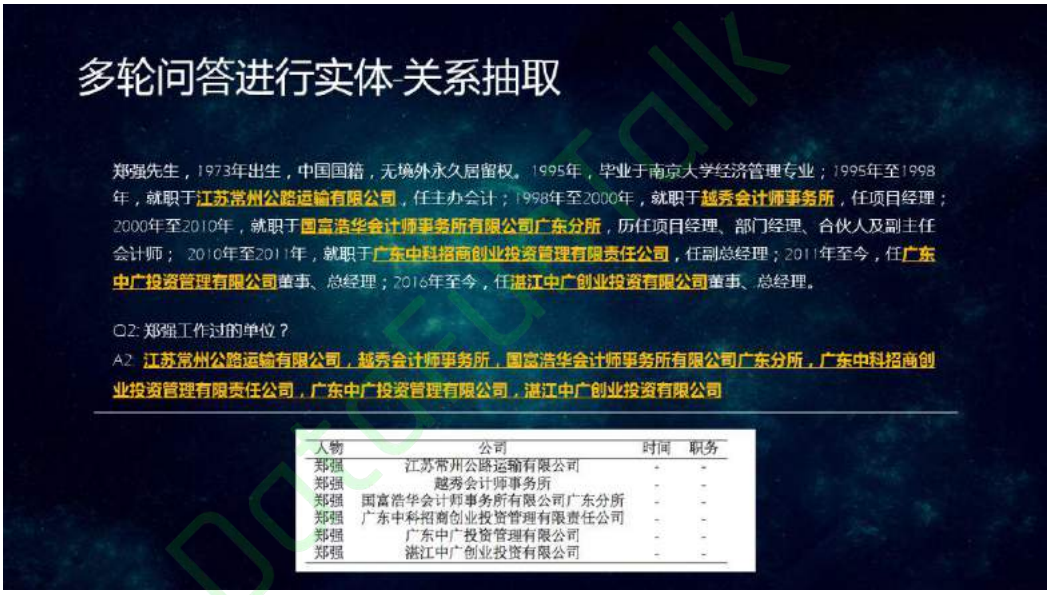
最后，把上面三步的答案拼在一起得到最后一个问题，E1 在 E2 担当 E3 的任职时间是什么，得到 E4。

4. 多轮问答进行实体-关系抽取

接下来就是刚刚举得例子，首先是第一个问题：

文中提到的人物有哪些？

然后是他的公司是什么？



再分别问在每个公司的工作时间？

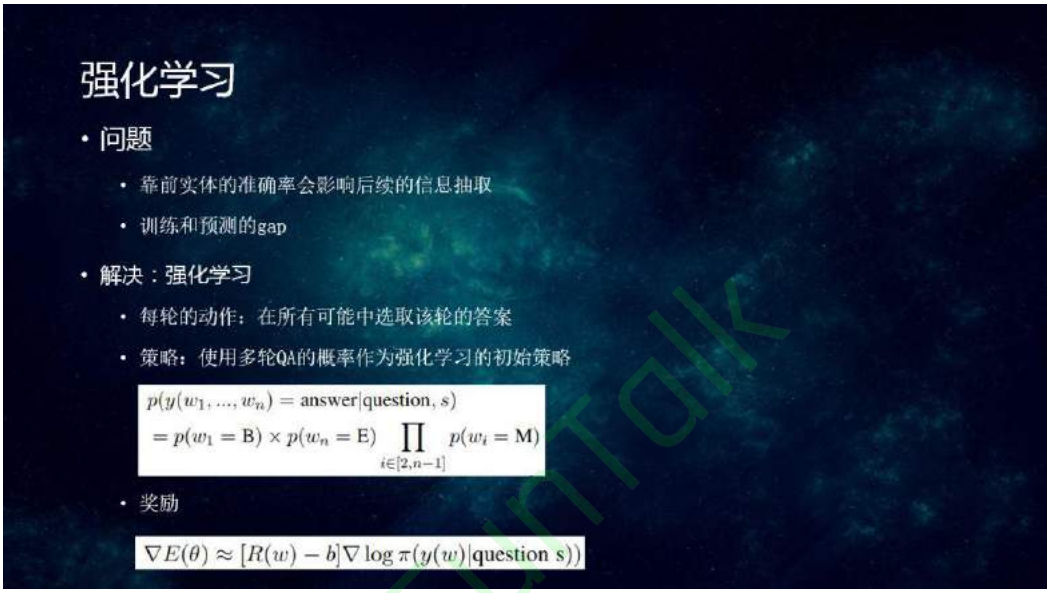
担任什么职务？

在越秀会计事务所工作的时间？

在其他公司担任的职务，等等问题。

这样通过阅读理解的方案就得到了这样一张表，也就是完成了从非结构化文本到结构化文本的转换。

5. 强化学习



这里在做阅读理解的时候有一个问题：靠前实体的准确率会影响后续的信息抽取，这样的问题在翻译中也会遇到；训练的时候我们一般会使用前面实体的 ground truth 填入后面实体的模板；在预测的时候，我们会使用实际预测

出的答案，这样就产生了一个训练和预测的 gap，第三阶段错误的实体，原因很有可能是第一阶段的实体就错了，这样的 Loss 并没有回传给第一阶段的实体。

所以，我们采用了强化学习的方法：

每轮的动作：在所有可能中选取该轮的答案。

策略：预先用 ground truth 训练多轮 QA 的概率作为强化学习的初始策略，避免策略不稳定的情况。

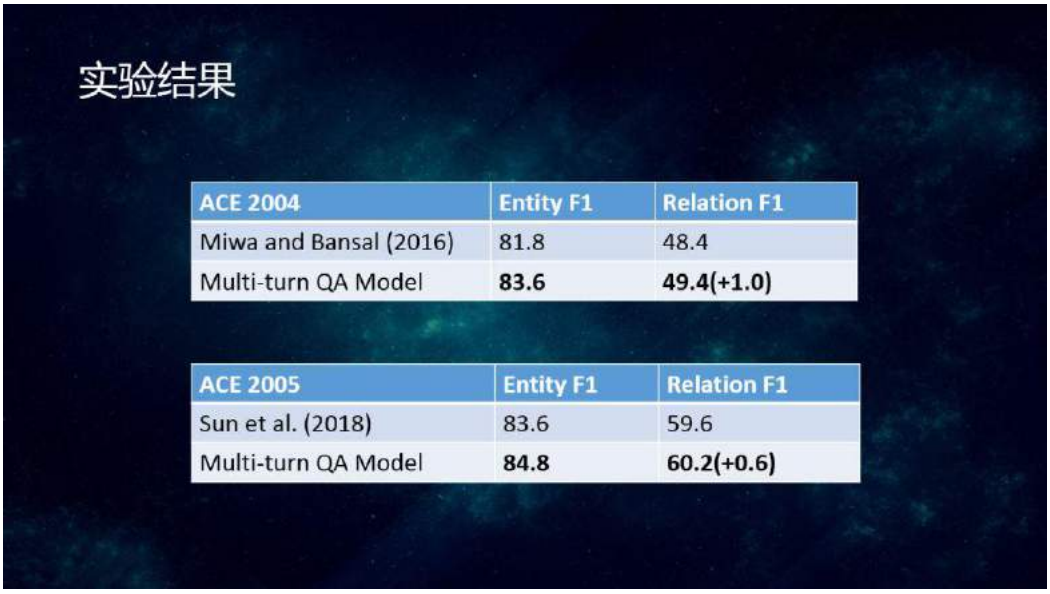
奖励：

$$\nabla E(\theta) \approx [R(w) - b] \nabla \log \pi(y(w)|\text{question s}))$$

得到的奖励就是这个公式，得到这个公式之后就可以将 Loss 回传，这样后面轮数的 Loss 可以反馈给前面的轮数，使前面的轮数更加准确。

6. 实验结果

DataFunTalk



我们发现在 ACE2004 和 2005 数据集上，Multi-turn 的方式相比于传统的方式分别有 1 个点和 0.6 个点的提升。

Part5 性能优化

刚刚讲了使用阅读理解的方式来做信息抽取，使用的阅读理解是基于 BERT 模型，由于 BERT 参数非常庞大，会遇到一些性能的问题：

1. 深度学习模型性能问题

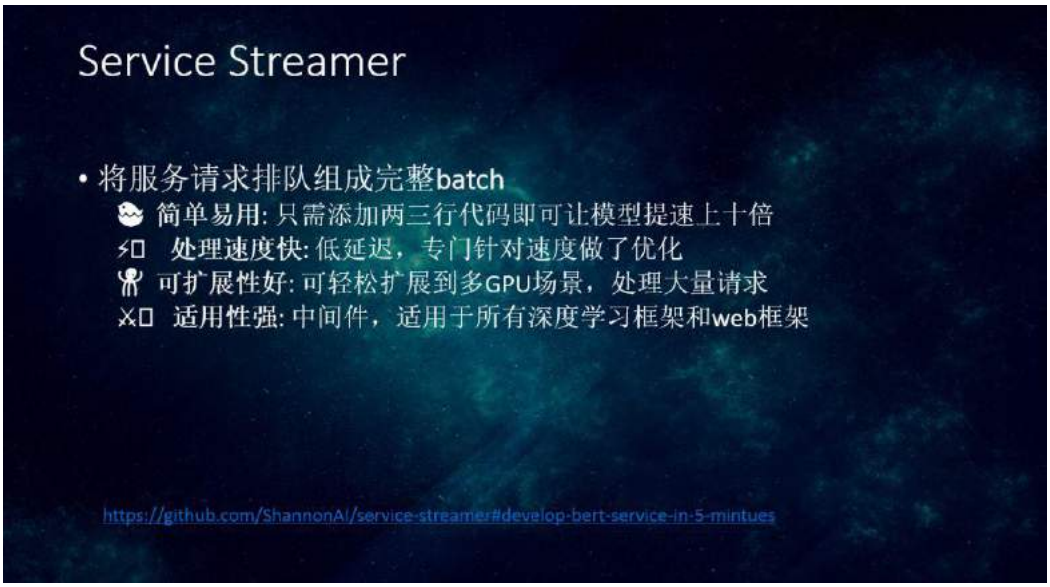
DataFunTalk

在做 web service 时，由于用户的请求是离散的，且大小不定，不易 batch 运算。

以 MASK LM 任务为例 ,一个 64 个字的句子做 MASK-LM 任务 ,封装为 web service，我们发现 QPS 只有 12。因此，我们开发了 Service Streamer。

2. Service Streamer

DataFunTalk



Service Streamer 将服务请求排队组成完整 batch，其好处有：

简单易用：只需要添加两三行代码即可让模型提速上十倍；

处理速度快：低延迟，专门针对速度做了优化；

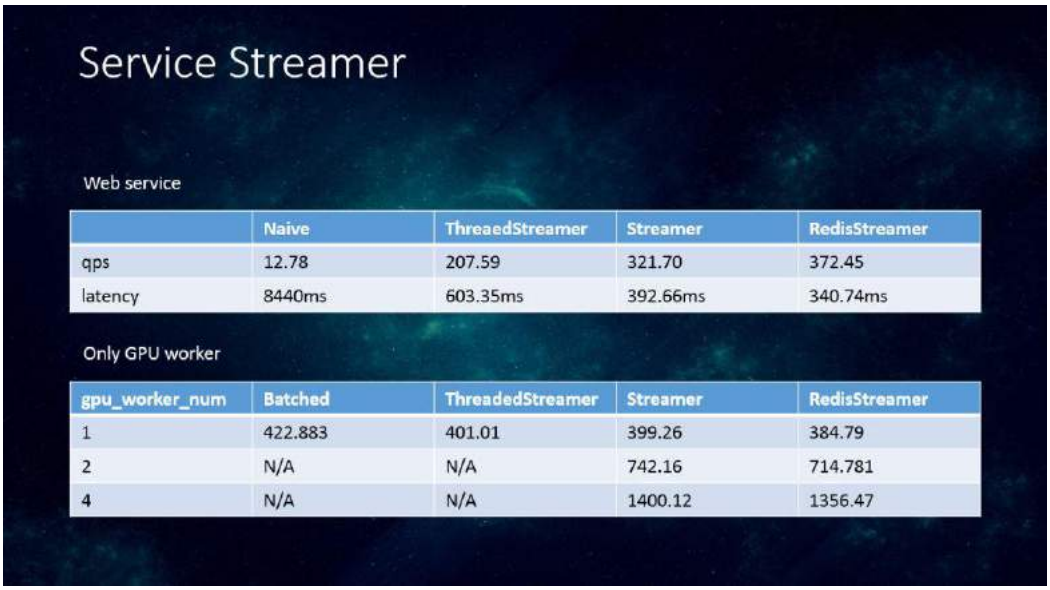
可扩展性好：可轻松扩展到多 GPU 场景，处理大量请求；

适用性强：它是一个中间件，适用于所有深度学习框架和 web 框架。

Github 链接如下，感兴趣的小伙伴可以仔细研究下：

https://github.com/ShannonAI/service-streamer#develop-bert-service-in-5-mintues

DataFunTalk



我们做了一个 benchmark，在一个普通的 Web Service 情况下，QPS 是 12，latency 是 8s，使用 ThreaedStreamer 可以增加到 208，继续使用多进程的 Streamer 可以做到 321，使用 RedisStreamer 会在 350 左右，根据每次测量会有些区别。

在多 GPU 场景下，可以继续提升性能，且非常容易扩展，当使用 2 张或者 4 张卡的时候，QPS 可以成比例的增加。

Part6 总结

最后小结下此次分享的主要内容：

自然语言处理。重点介绍了 NLP 与深度学习结合的两个范式。

DataFunTalk

信息抽取。重点介绍了实体关系抽取的传统做法及传统做法的问题，传统做法很难处理层级关系及准确率问题。

机器阅读理解。基于机器阅读理解的信息抽取方法在结果上优于传统做法。

性能优化，利用 service streamer 中间件可以提升深度学习模型性能。

DataFunTalk

基于知识图谱的问答在美团智能交互场景中的应用和演进

嘉宾：潘路　编辑：凌铭

导读：目前为止 IT 产业经历了六次浪潮，分别为：大型机时代，小型机时代，个人电脑时代，桌面互联网时代，移动互联网时代和 AIOT 时代。在这些时代背后可以发现是人机交互方式的变化：从鼠键交互，到触控交互，再到语音智能交互，可以看到人机交互的方式在向更自然更直接化的方式演进。今天会和大家分享基于知识图谱的问答在美团智能交互场景中的应用和演进。

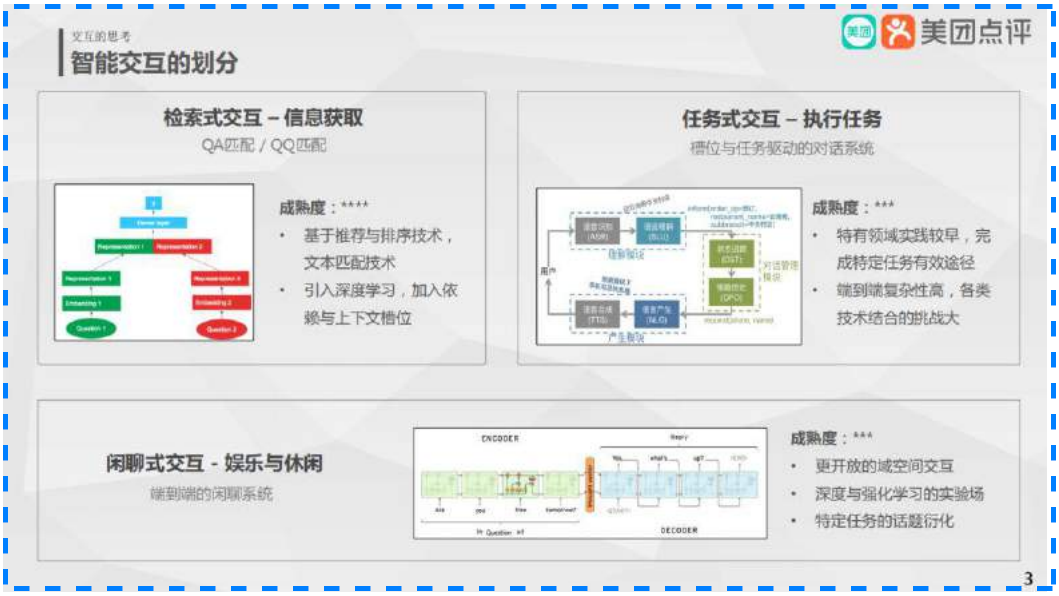
今天的介绍会围绕下面三点展开：

- 智能交互背景介绍
- 受限场景问答应用和演进
- 复杂场景问答应用和演进

——智能交互背景介绍——

1. 智能交互的划分

DataFunTalk



智能交互的划分基本上是根据人类需求拆分：

- 检索式交互—信息获取，比较经典的方法 FAQ：QA 匹配，QQ 匹配；
- 任务式交互—执行任务，比如订机票（ 酒店 ）的特定任务；
- 闲聊式交互—娱乐与休闲，基于深度学习的端到端的学习系统。

2. 美团生活服务交互

DataFunTalk

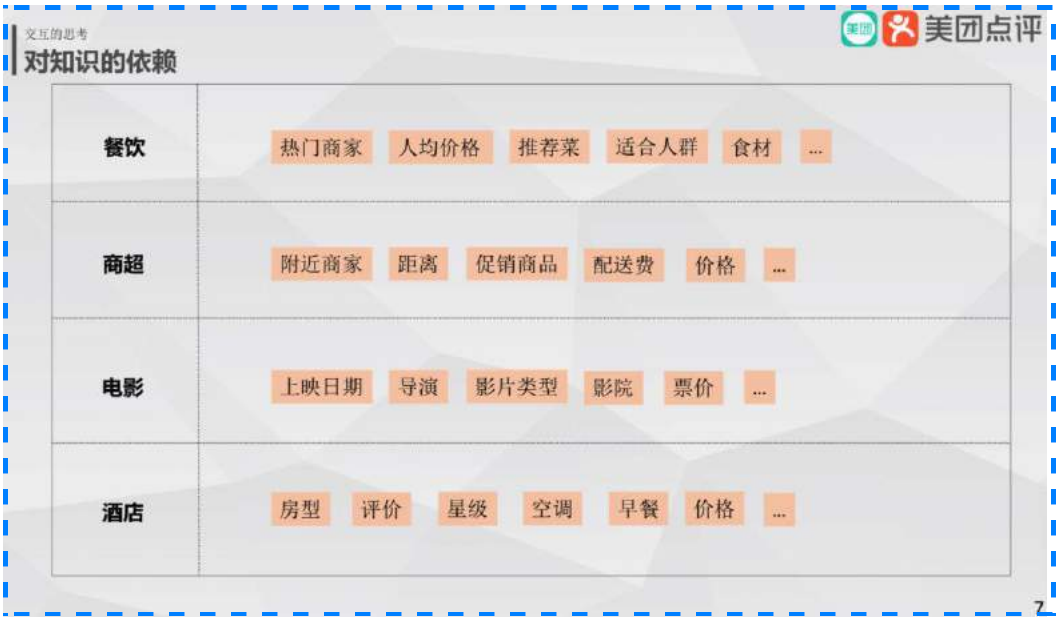
美团是做生活服务的公司，覆盖了餐饮、娱乐、酒店和旅游等各个生活领域，以上这么多种领域更适合哪种智能交互方式，以及智能交互方式是如何在场景中落地的，下文中都会进行说明。

3. 美团 APP 中的交互

下面先举个例子，首先在商超场景下我们分析使用路径发现，接近 4 成的用户在商品 A 加入购物车后，会去查看其他商家与商品 A 相似的商品。这个行为很明显是一个商品比较的诉求，而该诉求通过目前的功能交互方式很难完成。

再举个例子，在打车场景下我们打车需要在 app 中进行 12-14 次页面交互，app 交互比较琐碎。而语言智能交互只需要简单的一句话就能完成，比如"帮我叫个快车，今天下午三点从天安门到北京西站"。

DataFunTalk



以上交互的场景对知识有重要的依赖，需要使用知识帮助人们完成选择：

餐饮：热门商家，人均价格，推荐菜，适合人群，食材等；

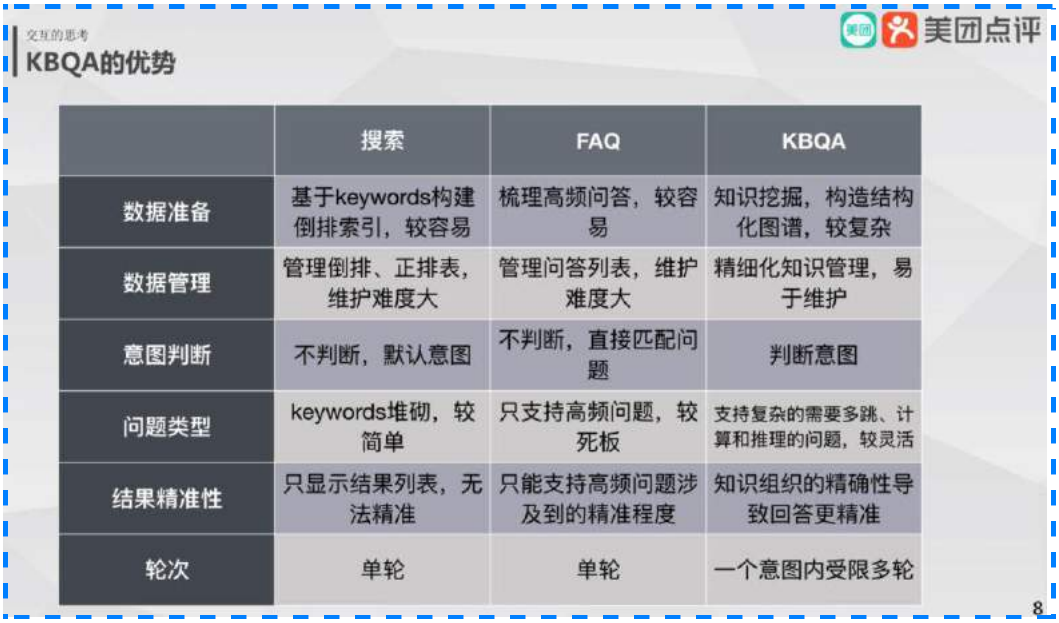
商超：附近商家，距离，促销商品，配送费，价格等；

电影：上映日期，导演，影片类型，影院，票价等；

酒店：房型，评价，星级，空调，早餐，价格等。

由于交互对知识有重要依赖，于是我们引入了基于知识图谱的问答KBQA。

4. KBQA 特点



由上表可以看到，KBQA 相对于其它技术的特点：

数据准备：KBQA 处于劣势，需要建构知识图谱，专业领域人参与并且较为复杂。

数据管理：KBQA 为知识结构易于管理和维护。

意图判断：KBQA 需要判断意图，定位到意图下面的子图，从而对子图进行检索，这样精度比较高。

问题类型：可支持多跳，带有约束的计算和推理，较为灵活。

结果精准性：由于知识结构，回答的精确性会比较高。

轮次：对于多轮问答能够很好完成。

——受限场景问答——

1. 什么是受限场景

受限场景有以下两个特征：

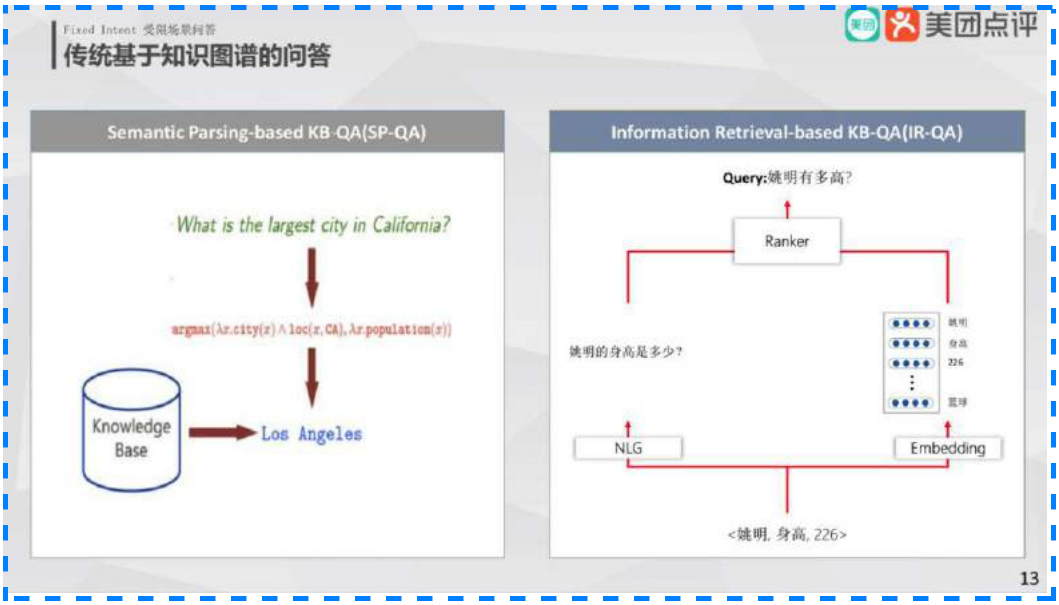
- 交互意图和需求在确定性范围
- 知识与资源处于封闭、收敛空间

例如到店点餐，点餐的意图确定，并且资源也是有限的，商家和菜品都是确定的；同样，机票预订起落地点、仓位、价格、人数也是有限的；打车意图是明确的，而地点可以任意选择不是受限的，但是可通过高频的地标建立受限的地点。

DataFunTalk

下面，会介绍一些受限场景具体实现方案。

2. 传统基于知识图谱的问答



首先介绍的是传统基于知识图谱的问答，该类问答主要分成两大流派：

Semantic Parsing-based KB-QA

把用户的问题转换为机器的查询语句，直接查询知识图谱获取答案。

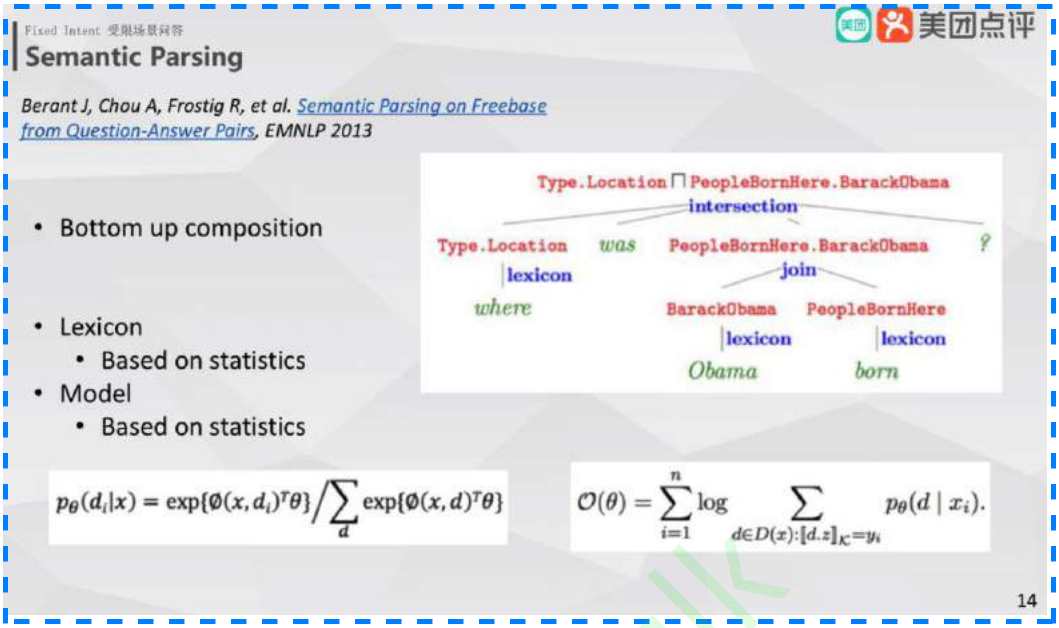
Information Retrieval-based KB-QA

使用端到端的方式解决问题。先抽取用户问题中核心信息，然后根据核心信息定位到图谱中确定子图（ 子集 ），最后生成答案。生成答案有两种方式：

① 子图的候选答案形成三元组逆向推导生成自然语言的问题和用户原始问题做匹配

② 子图候选答案和周围的信息做 embedding 与原始问题 embedding 做匹配。

3. Semantic Parsing



通过《Semantic Parsing on Freebase from Question-Answer Pairs, EMNLP 2013》这篇论文介绍 Semantic Parsing 的几个步骤：

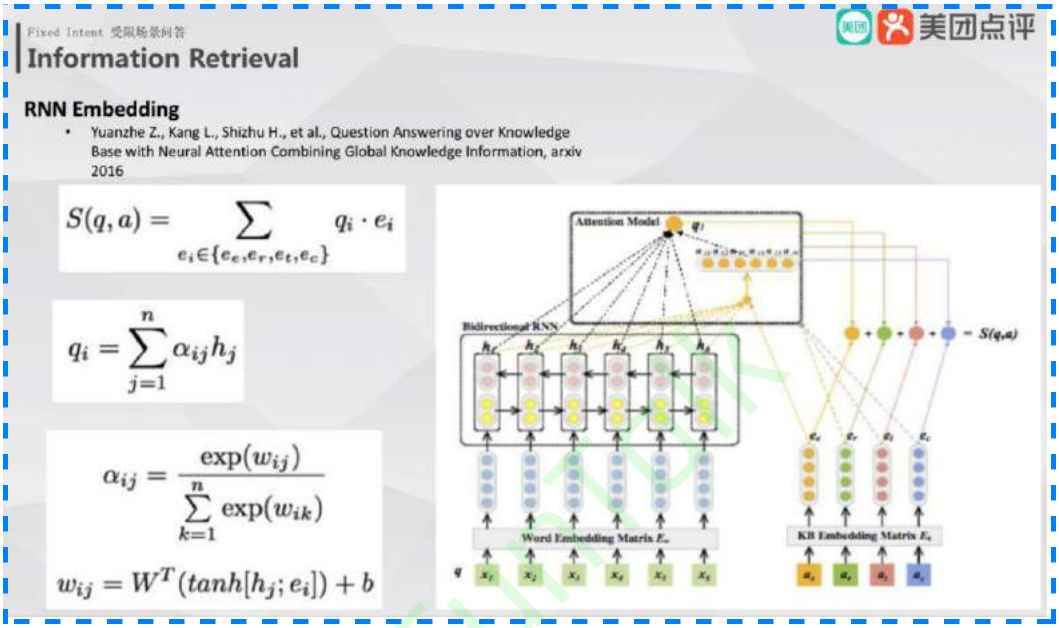
- 把问句的词语或者单词与知识库的实体或者关系进行映射，映射构成叶子节点。

- 直接对叶子节点使用链接、求交和聚合三种操作自下向上构建语法树。

- 这三种操作中会存在多颗语法树，需要构建一个分类器把正确的语法树区分出来。最终语法树的根节点则为输出的查询语句。

Semantic Parsing 会有一些局限，需要大量的自然语言逻辑表达式的标注，并且只能满足有限的场景和逻辑表达。而 Information Retrieval 可以改进 Semantic Parsing 的限制。

4. Information Retrieval



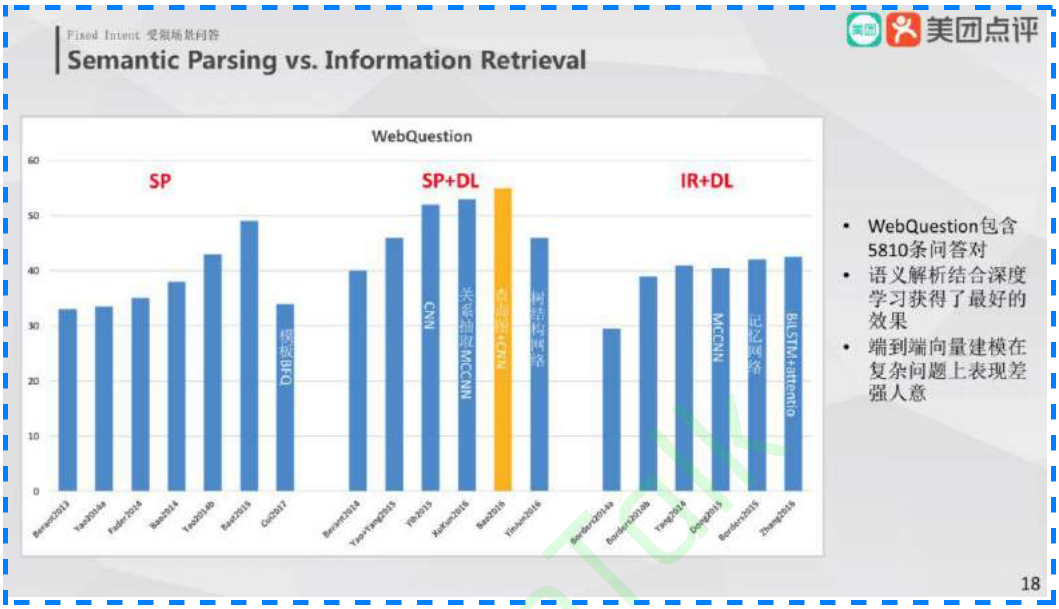
同样，通过《Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information, arxiv 2016》阐述 Information Retrieval 的几个步骤：

问题编码，问题使用双向 lstm 并加入答案的 attention 编码；

答案编码，把所有答案分成四块进行编码，分别为答案本身编码，答案到实体边的关系编码，答案类型编码和答案上下文编码；

匹配，使用问题编码和答案编码进行匹配，最终输出排名最高的作为匹配答案。

5. 效果比较



在 WebQuestion 数据集可以看到，Semantic Parsing+DeepLeanring 效果表现最好 f1 值在 45-55 之间。Information Retrieval 的端到端在复杂问题处理上表现不太好。

6. 面向美团场景

针对美团的场景技术如何选型呢？首先美团领域比较多，但是每个领域之间的关联不是很强，其次模型的训练样本也比较少，希望能快速实现领域之间迁移。Semantic Parsing 需要大量的自然语言标注样本，而 Information Retrieval 的可解释性在目前工业界稍微差些。

我们做法将这两者技术流派优点结合起来，通过 Information Retrieval 精确定位到有限空间的子图中，Semantic Parsing 生成可解释性的查询语句。那么技术步骤为首先做实体识别和链接，其次确定子图，之后做关系识别，最后 SparQL 查询输出结果。这样的优点为可通过规则和无监督冷启动快速进行迁移。

7. 小结

在受限场景的问答，其特征为资源内容、交互轮次、知识外延和意图都是有限的；它可以实现基础属性问答、带约束的资源查询、动态属性值计算和资源信息比较。

——复杂场景问答——

1. 复杂场景定义

DataFunTalk

在美团，绝大多数都是复杂场景的问答。与简单场景相比，其交互意图与需求从确定的空间到模糊的空间，资源从封闭的数量到开放的数量。

这里举三个例子：

"**百威啤酒**有没有**便宜**的？"，在商家点餐的受限场景下，**百威啤酒**指代就是商家所卖的几款**百威啤酒**；但是在智能音箱的复杂场景，**百威啤酒**指代的是家旁边的**超市**所卖的百威啤酒还是旁边**酒吧**做活动的**百威啤酒**呢？

"**下午四点钟**的**复联四**还有没有 **IMAX 票**了？"，假设已经识别**复联四**为一个电影，**下午四点**的时间如何与电影关联起来，另外 **IMAX 票**为影厅的一个属性如何与电影关联起来呢？

DataFunTalk

"**第一个**销量多少？"，很明显用户在多轮问题所问的，那么**第一个**指代的是什么呢？

2. 方案概述

以上三个例子，我们可以归纳成三大问题：实体爆炸、非连通查询和融合上下文。

3. 知识建设面临的挑战

DataFunTalk

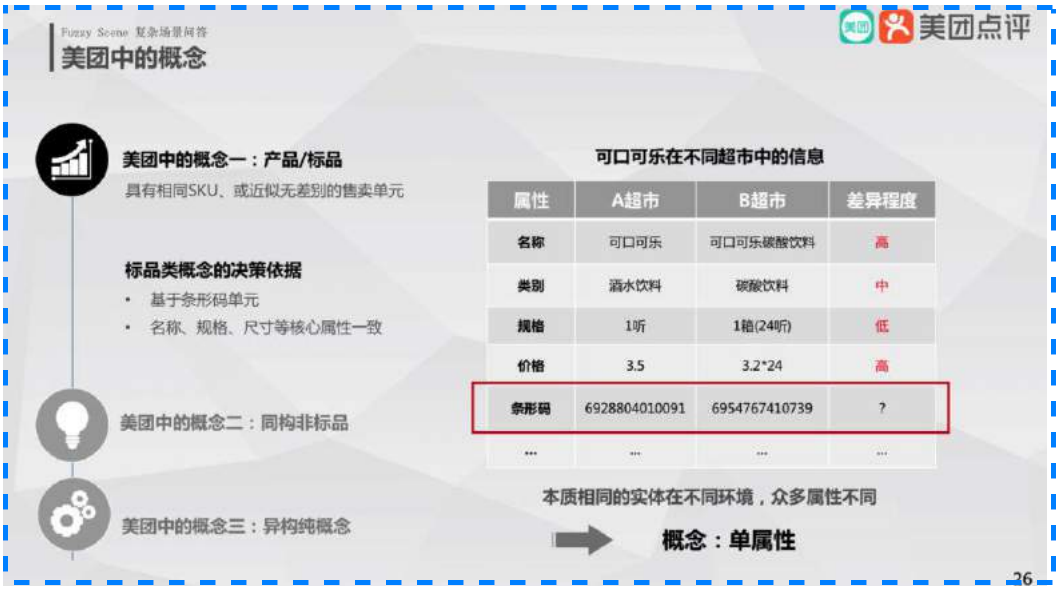
我们先介绍实体爆炸的问题。传统的开放域知识图谱是根据实体上层的概念节点（ 抽象物理空间 ）去建设的。比如：宫保鸡丁、青岛啤酒、复联四，它们不是商家卖的商品，就是一个抽象的概念。但是美团场景下，面临了挑战，比如：青岛啤酒原浆 5L 桶装（xxx 超市，条形码:yy... ），青岛啤酒 10 度 500ml（xxx 超市，条形码:yy... ）等这些都是实实在在商家卖的商品。这些实体如何与用户资源关联起来，并且把这些实体汇聚起来？

4. 三层概念节点

我们提出三层概念节点：产品/标品、同构非标品和异构纯概念

产品/标品

DataFunTalk



能区别不同商品类别的属性称为标品类概念 ,比如条形码可对商品做明确的区分，除此之外还有名称、尺寸和规格等等属性，这些属性一致就称为同类商品。

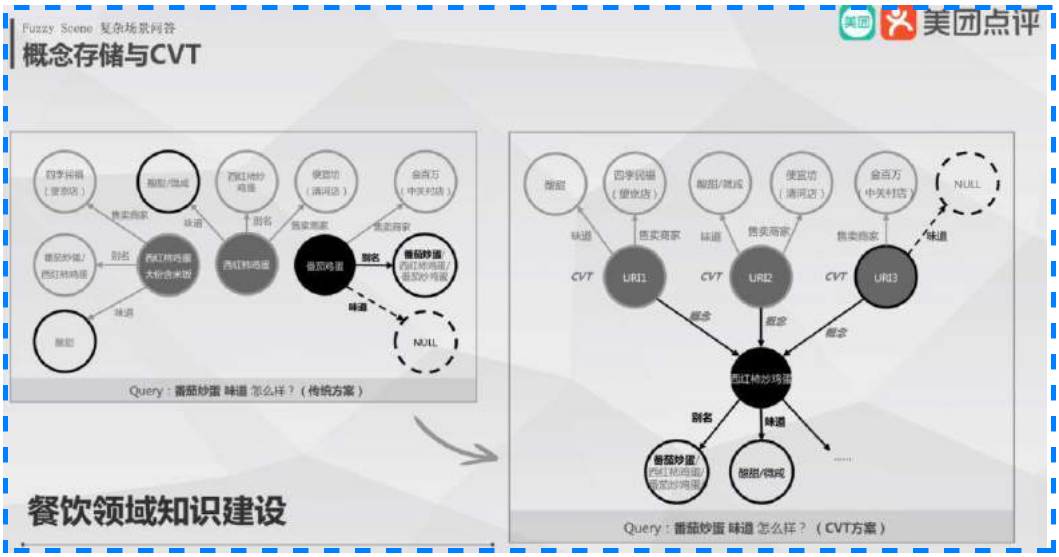
同构非标品

其定义为相同本体下，要求某些特殊属性相同。没有明确的属性做商品类别的区分，但是特殊的属性可以做区分，比如青岛啤酒、车厘子 JJ 级等，他们的品牌相同，等级相同、类型相同。这些概念可以作为商品检索词。

异构纯概念

异构纯概念脱离了本体的概念，是由人类自行组织构建，形成的认知。比如消费的人群、消费的商圈，称之为异构纯概念。其给予场景化问答提供非常大的帮助。

5. 概念存储

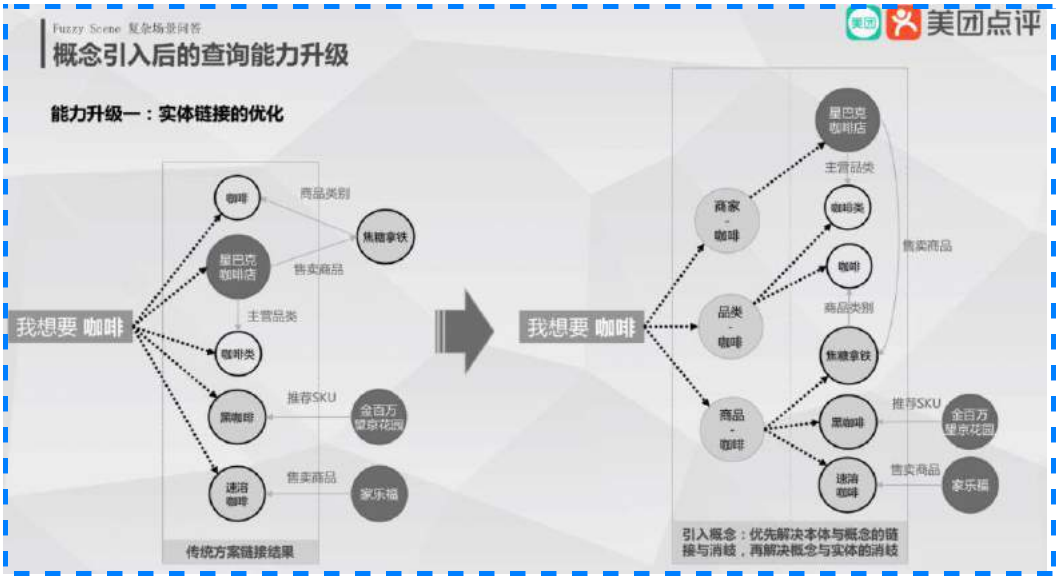


我们借鉴了 Freebase CVT 的存储，将概念节点当作节点，真实售卖的商品为概念节点的 CVT 子节点 ,把 CVT 相似的属性比如名称和品牌等抽取出来作为概念的属性。

6. 概念引入效果

① 实体链接的优化

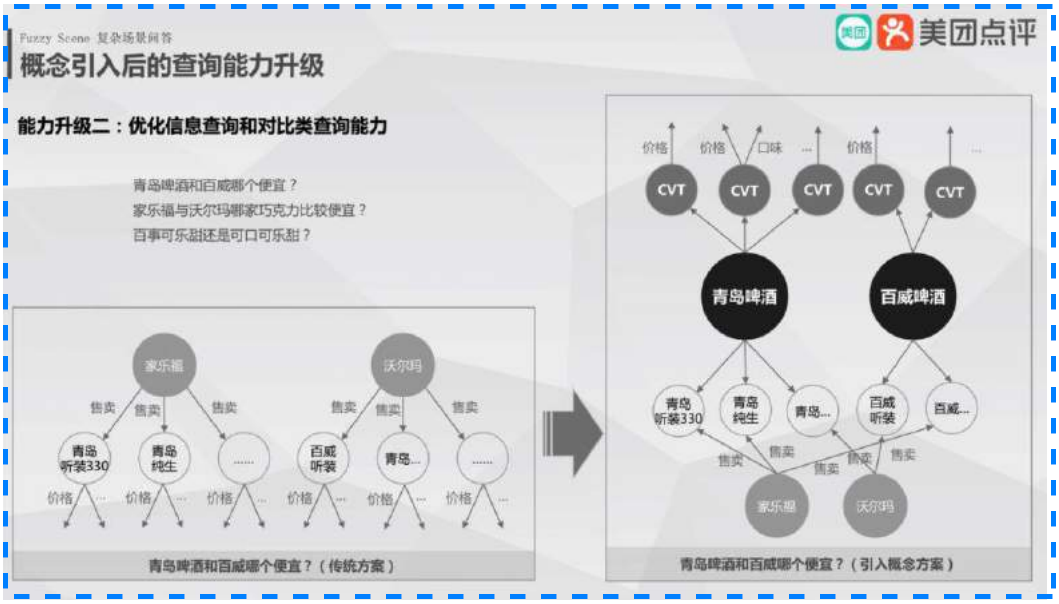
DataFunTalk



举个例子 ,咖啡在没有概念的情况下 ,我想要咖啡会链接到很多资源 （ 咖啡类、咖啡店和超市卖的速溶咖啡等 ）；当概念引入后，会链接到概念节点上，再基于概念做链接和消歧，那么就能直接链接到商家、品类和商品上 ,并且若用户没有召回资源的真实诉求 ,不会进入下一个实体层去查询。

② 优化信息查询和对比类查询能力

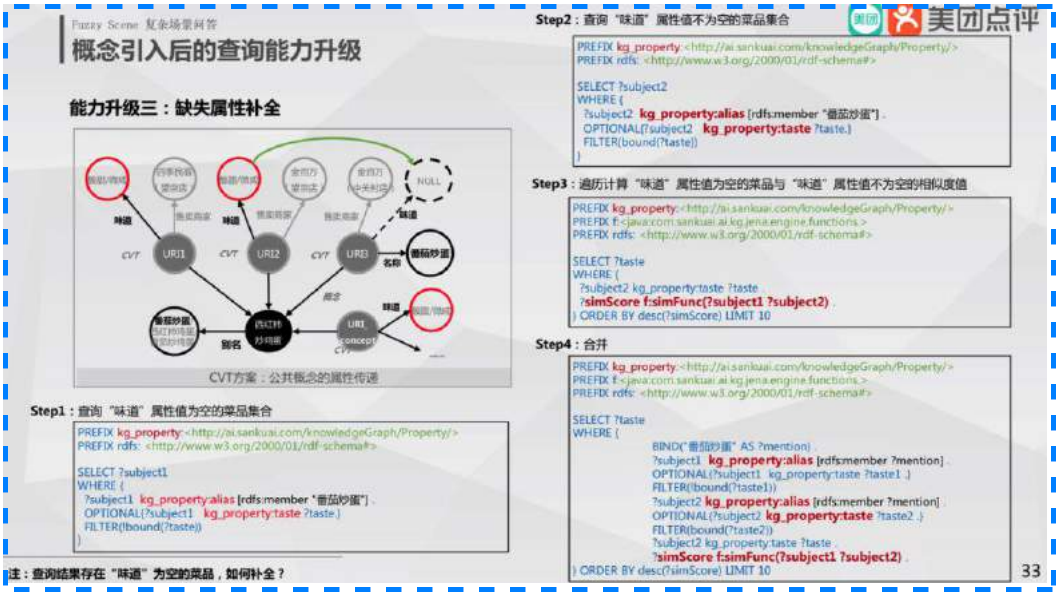
DataFunTalk



假设用户查询"青岛啤酒和百威啤酒哪个便宜"在没有概念的情况下，要查询成千上万个商品去做比较，但是概念的引入，只需要在概念层进行比较。

③ 缺失属性补全

用户在金百万（ 中关村店 ），问番茄炒蛋味道如何？假设知识图谱不存在番茄炒蛋在这家店的味道，该如何回答呢？

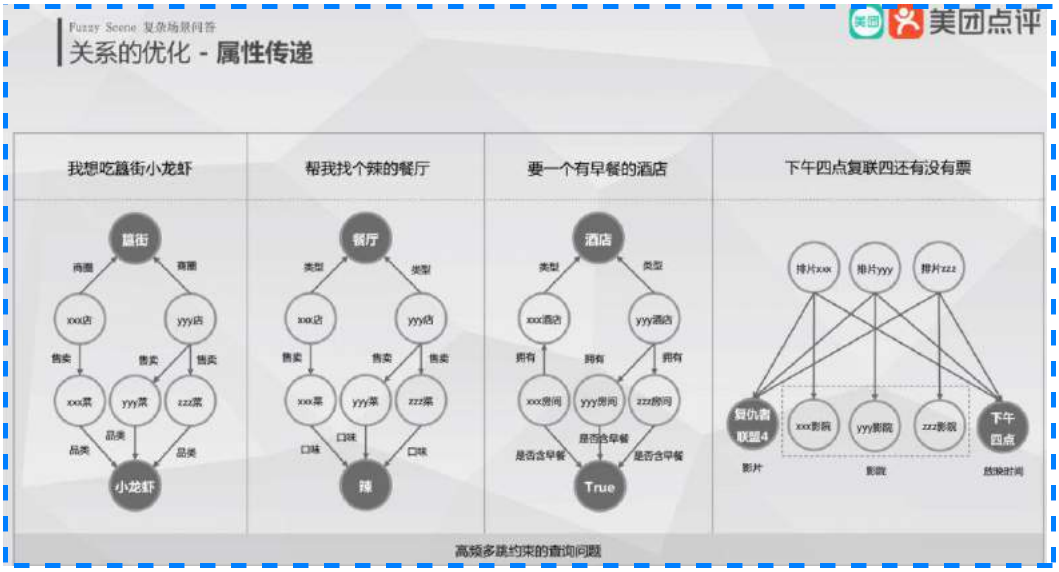


有两种方式：

第一，若存在番茄炒蛋概念且这个概念有味道属性，可使用这个概念的味道属性填充该问题；

第二，更好的方式：找到用户问题最相似的子节点，即金百万（ 中关村店 ）的子节点，发现其定义了番茄炒蛋和其味道属性，可拿该味道进行填充。

7. 非连通查询



在非连通查询中会存在属性传递（ 传递约束 ）的问题，举几个例子：

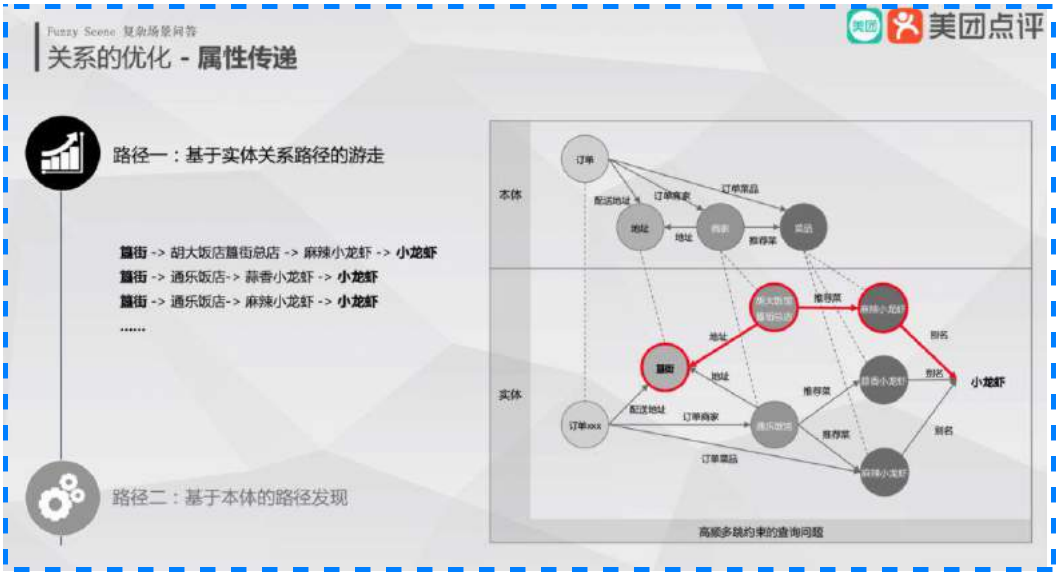
"我想吃篮街的小龙虾"，篮街为一个商圈其不能与小龙虾直接相连接；

"帮我找个辣的餐厅"，辣为菜的口味不能与餐厅直接相连接；

"帮我看一下下午 4 点复联四还有没有票",复联四与排片时间也不能直接关联。

① 属性传递

DataFunTalk



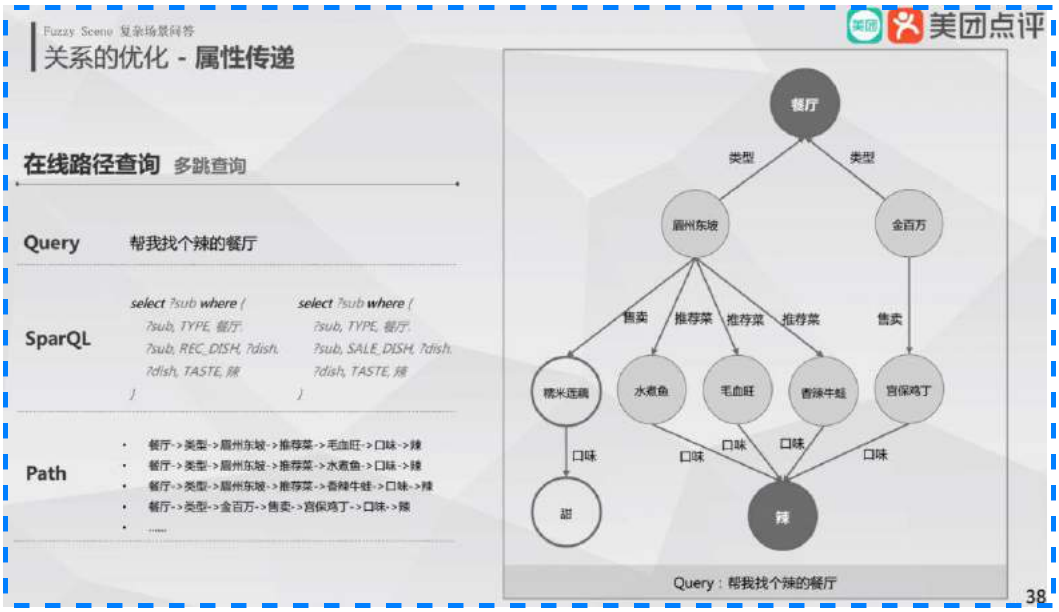
路径游走有两种方式：

一是实体关系路径游走，例如"篮街 -> 胡大饭店篮街总店 -> 麻辣小龙虾 -> 小龙虾"。

二是本体的路径发现，游走方式为"篮街 -> 地址(本体)-> 订单(本体)-> 菜品(本体)-> 小龙虾"。

② 在线路径查询

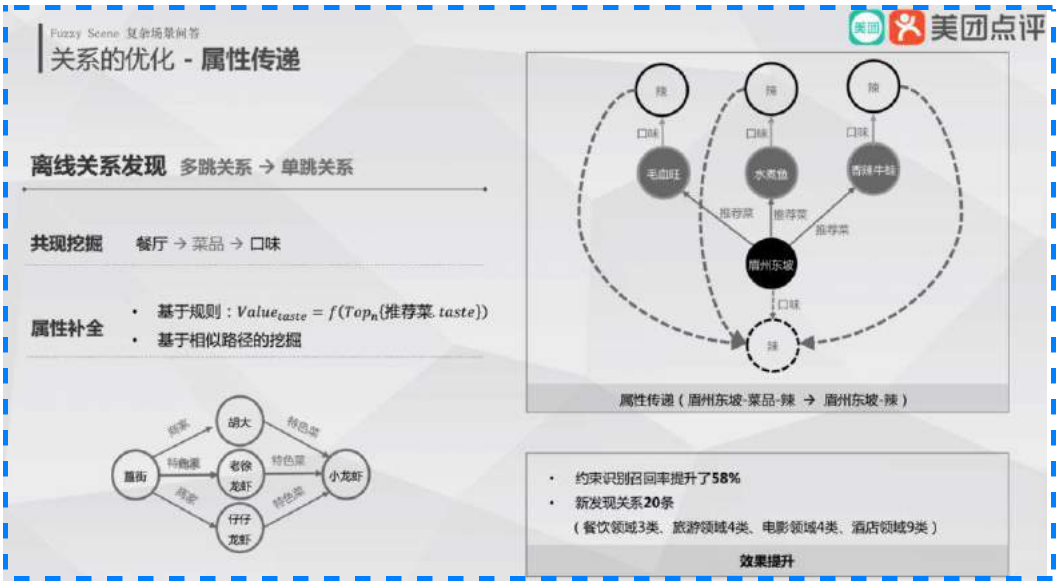
DataFunTalk



基于上面路径发现之后，融入到现有的流程中，当用户输入"帮我找个辣的餐厅"，首先做路径的发现，"餐厅->类型->眉州东坡->推荐菜->毛血旺->口味->辣"，再形成 SparQL 查询语句，得到结果就能回答用户的问题。

③ 离线关系发现

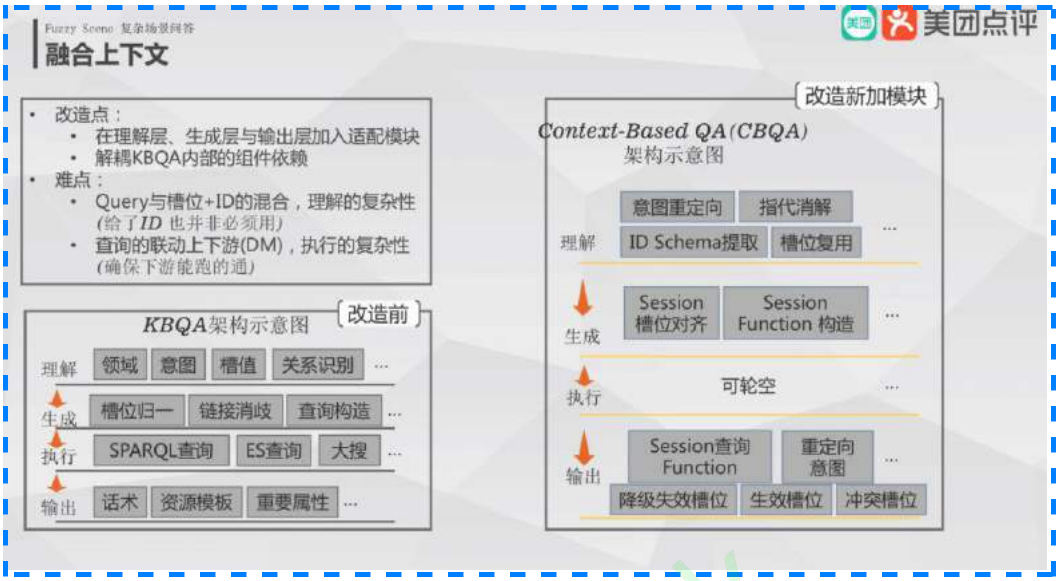
DataFunTalk



既然眉州东坡大多数推荐菜的口味是辣的，那么可以构建眉州东坡口味的边为辣。同样发现簋街多跳之后特色菜为小龙虾，那么也可以构建簋街的特色菜为小龙虾.....由此可见，在离线把边补充足够成分，那么在线路径查询的压力要小很多。

8. 融合上下文

用户在外卖点餐的真实场景中，首先通过任务型交互给到用户的商家列表，用户可选择商家以及对应的菜谱，然后从购物车里筛选内容，最后下单。这个场景为状态迁移的多轮交互任务。我们想在该场景中融合问答，那么用户在商家状态下问该商家的推荐菜品，在菜品状态问菜品口味，在购物车状态问优惠券，也可以在下单后问订单的状态。那么可以将任务型多元交互系统和问答系统融合起来。



我们提出了两点改造：

- 理解层、生成层和输出层适配多轮交互模块。
- 解耦 KBQA 内部组件，可独立被外部模块调用。

在改造过程中遇到两个难点：

- 理解复杂性提升，需要理解上下文的传入槽位、id 与意图。

- 执行复杂性提升，融合了问答之后任务状态可随时打断或者变更。

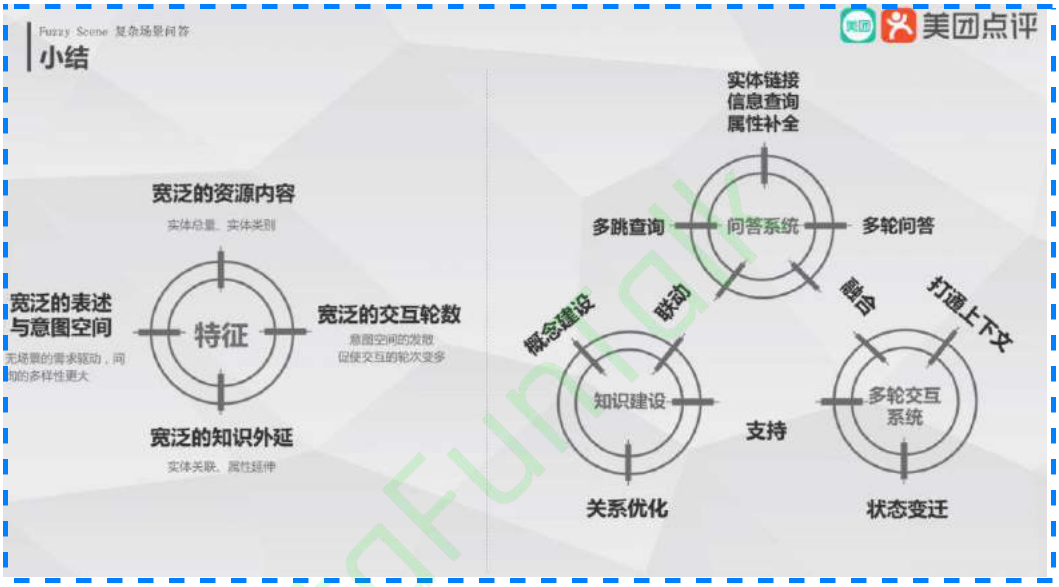
具体的架构改造为：

理解层，加入了意图重定向、指代消解、ID Schema 提取和槽位复用等。

生成层，Session 槽位对齐、Session Function 构造等。

输出层，不会输出答案本身，而是输出查询 Function 和意图。

7. 小结



在复杂场景问答中，表述与意图空间、资源内容、交互轮数、知识外延都是宽泛的。在该场景下不仅仅只有优化算法本身，还需要知识建设和多轮交互系统做一个联动的优化。

DataFunTalk

人机语音对话技术在 58 同城的应用实践

嘉宾：李忠　编辑：赵旺

导读：本文将介绍 58 同城人机语音对话机器人的完整解决方案，重点分享对话策略管理、自动电话拨打、意图识别、防骚扰控制等核心模块的设计实现，并阐述如何将语音机器人应用于 58 各业务场景，以助力销售、运营和客服提高人效。

——背景——

58 同城是中国最大的生活信息服务平台，公司主营业务包括：招聘、汽车、金融、本地服务、二手等。其中，电话沟通是信息连接的重要渠道之一，如招聘业务模块，需要大量的业务人员进行求职信息确认、面试预约、面试回访等重复耗时的沟通工作。为了更好的服务 B 端商家和 C 端用户，实现“让生活简单美好”的公司使命，我们开发了语音机器人来进行电话语音沟通，减少业务人员的工作量并提升服务质量。

传统的沟通工作是完全依赖于人工拨打电话，而人工的方式相比于语音机器人存在很多问题。下图是语音机器人与人工的对比情况：

DataFunTalk

比较项	语音机器人	人工
年工作天数	365天（无休息）	250天（节假日休息）
日电话拨打量	由程序决定上限	200个/人（8小时工作）
工作态度	稳定，情绪饱满	情绪不可控
数据统计和记录	全面、客观	缺失、主观
费用	电话费 + 语音技术费用	人力成本 + 电话费

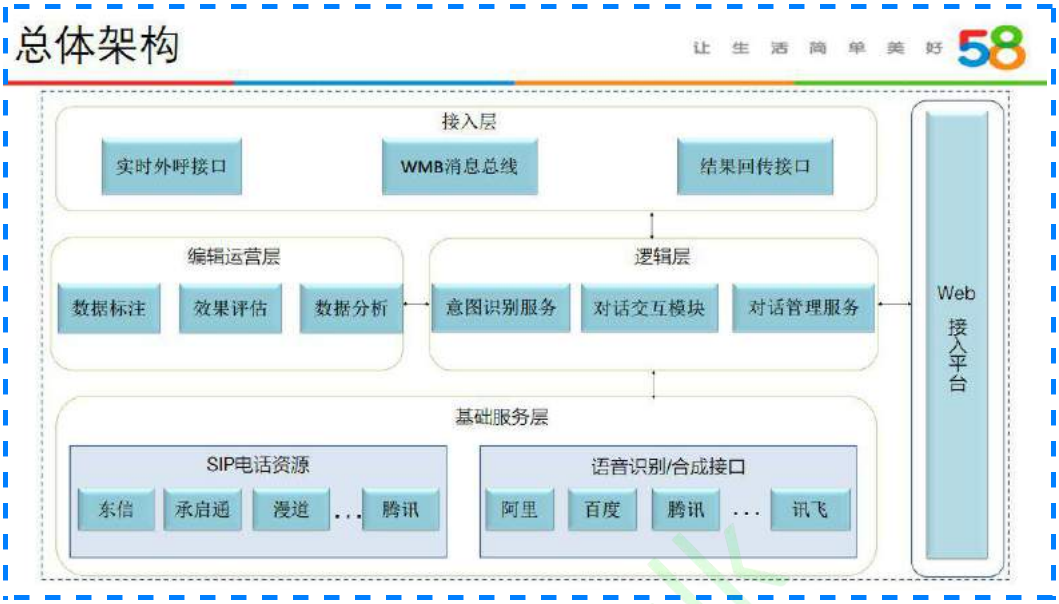
其中，关于工作态度，人工难以保持自身情绪的长久稳定，而机器人则可以始终如一的保持稳定、情绪饱满的工作态度；另外，在成本费用上，长期来看 ,语音机器人的成本远低于人力成本。对比了市面的语音机器人后，为了能快速的根据需求场景的变化与解决个性化的业务问题 ,公司采取了自研语音机器人的战略。

下面我们看一下 58 同城的人机语音对话示例：

从图中可以看出，语音机器人还是保持了相对流畅的对话。机器人怎样实现根据用户的不同回应做出不同的响应的呢？后面给大家做进一步的详细的介绍。

——语音机器人的总体架构——

1. 总体架构



语音机器人的总体架构 ：

① 接入层：主要以 API 接口的形式，方便业务方调用语音机器人。当一通电话结束时，机器人可以把消息以 WMB 方式传递给业务方，相当于异步调用。同时销售等业务方，可以通过结果回传接口将后期跟进、成单信息反馈给后台，进行进一步的算法优化。

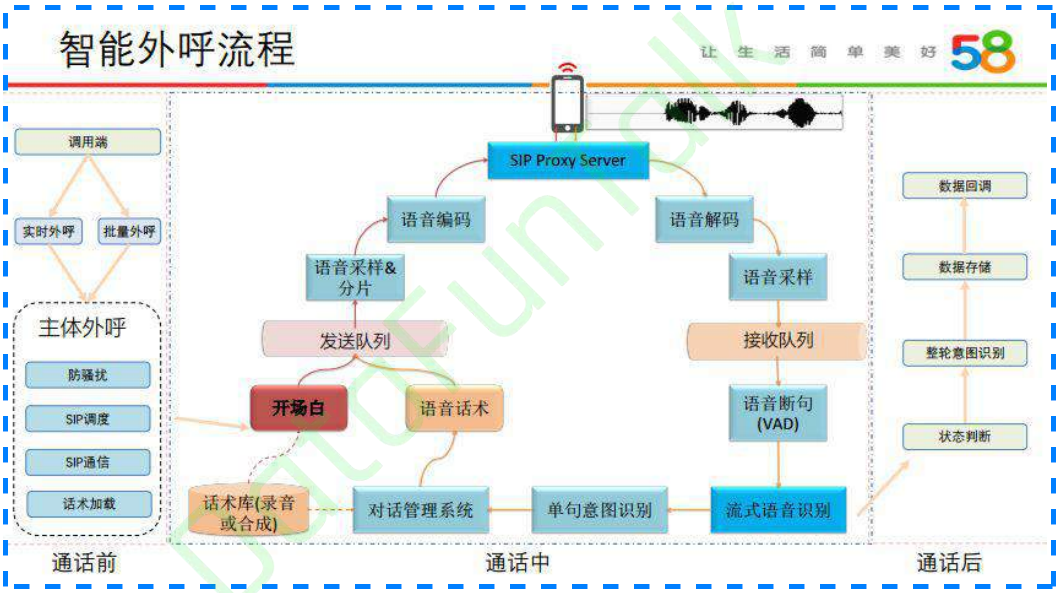
② web 管理层：主要负责话术的配置、权限的控制、批量拨打、防骚扰策略设置、数据可视化查询等。

③ 逻辑层：整个机器人的核心控制层，相当于人的大脑，保证整个对话流程完整进行。

④ 编辑运营层：目前主要用于数据标注，标注数据用于模型迭代与线上效果评估。

⑤ 基础服务层：主要包括 SIP 电话资源和语音识别/合成接口两部分。SIP 电话资源即实现拨打电话的资源，像东信、漫道等。语音识别采用的是阿里、腾讯等第三方的接口。

2. 智能外呼流程



语音机器人智能外呼流程主要分为通话前、通话中、通话后三个部分。通话前主要是调用端，调用端会把被叫号码、业务场景等使用的信息传给主体外呼，主体外呼拿到信息后会进行一些策略的设置，如防骚扰逻辑的设置、选择合理的 SIP 方、建立 SIP 通信、根据业务场景加载话术等。当触发开场白后，进入“通话中”状态，通过发送语音和利用第三方语音合成接口编码语音，经 SIP Proxy Sever 端发送给用户，用户接听后，再对用户的反应进行判别与用户语音的解码，再利用第三方的流式语音识别接口得到文本。通过对文本的分析，结合话术跳转的逻辑，对用户的回复进行合理的响应。通话结束后，语音机器人会对整轮电话进行通话状态判断、整轮通话意图识别、数据存储与回调，并以 WMB 的方式回传给业务方。

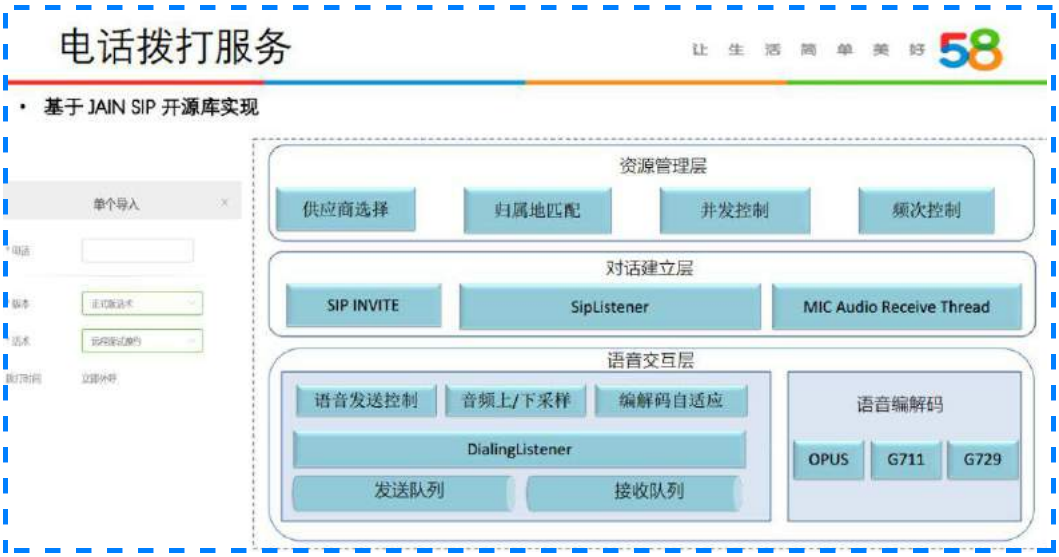
——核心功能——

58 同城语音机器人的核心功能包括四部分：电话拨打服务、通话状态识别、智能对话交互、整轮意图识别。

1. 电话拨打服务

实现电话的拨打服务功能，和客户建立连接。

DataFunTalk



当前这一功能实现主要是基于 JAIN SIP 的开源库，主要包括四层。其中在资源管理层控制上 ,对于呼叫用户会考虑同样的归属地来提升拨打电话的接通率。对话建立层基于 SIP 协议来实现建立通话连接与释放连接的处理。

防骚扰策略：

为了避免过多打扰和消耗用户，设计了防骚扰策略。其策略主要考虑：

① 被呼名单限制：设置了呼叫白名单与黑名单，比如有些特定用户名单是不能呼叫的。

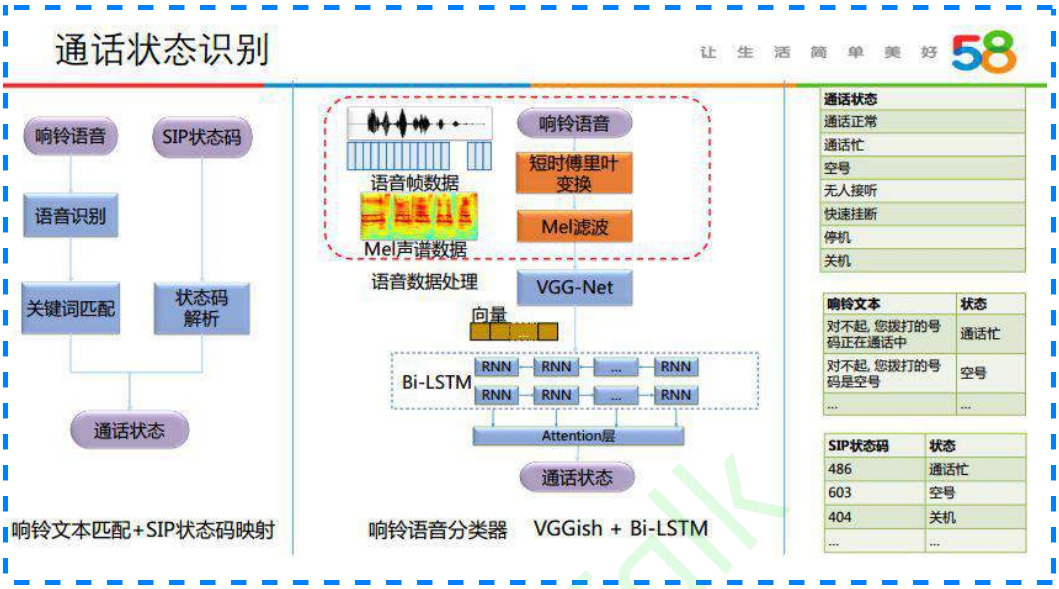
② 呼叫时间控制：哪些时间段是不能呼叫的。

③ 呼叫频次控制：对用户的呼叫频率要控制，防止过渡消耗用户。

④ 用户情绪识别：对呼叫中用户的情绪进行分析，当用户情绪很抵触时，添加到被禁的黑名单中。

2. 通话状态识别

判断拿到的客户电号码话是否真实存在和是否处于正常状态。比如做销售业务时，需要核对业务方拿过来的客户号码是否是空号、异常号码等，这个可用通过电话拨打服务来实现状态判断。

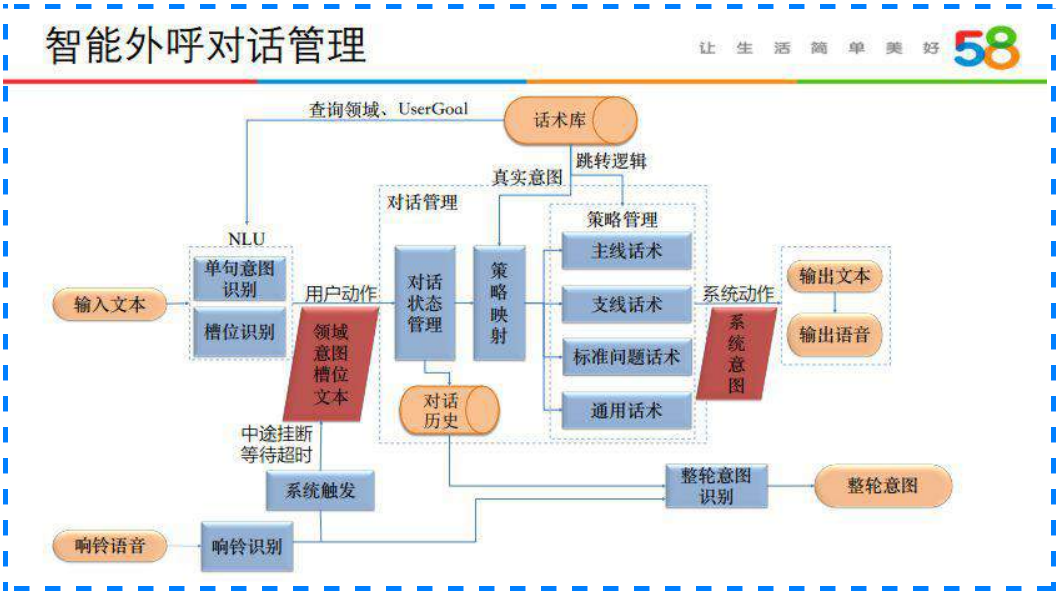


具体可以利用 SIP 协议和响铃语音来实现。SIP 协议 :利用基于 SIP 协议返回的 SIP 状态码来进行判断，比如返回 603 则判断为空号；响铃语音：通过将语音信号转化为文本，进行文本关键词匹配和文本分类，对拨打号码进行判断，比如“您拨打的电话已关机”判断为关机状态，还可以利用语音信息特征设计响铃语音分类器，判断号码状态。

3. 智能对话交互

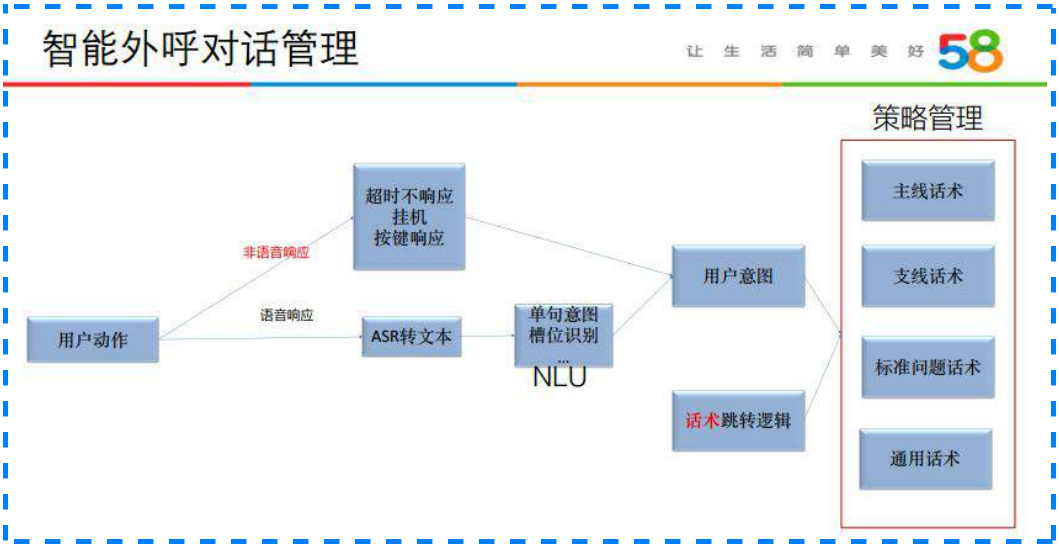
为了实现多轮对话流畅进行，智能对话交互主要包含以下模块：智能对话管理、电话按键捕获、单句意图识别、标准问题匹配、槽位提取。

① 智能对话管理：

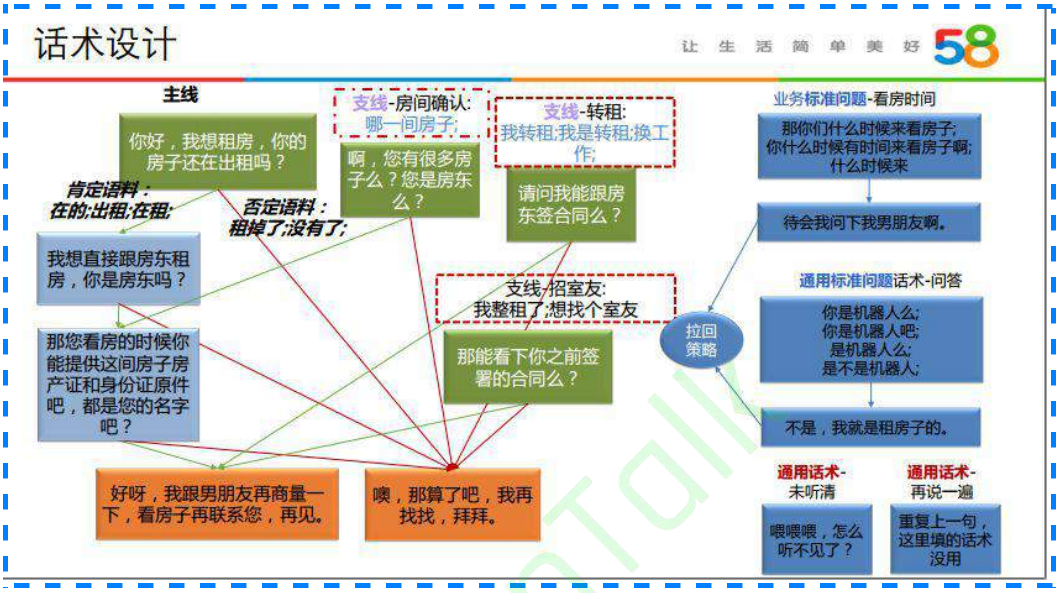


通常情况下：将用户语音转化后的文本作为模型的输入，首先对文本进行包括单句意图识别与槽位识别的 NLU 工作，然后将信息转化为用户动作并交给对话管理器。对话管理器需要判别用户的意图，根据预先设计好的话术库，结合映射策略进行话术选择跳转，生成系统动作。

具体的话术选择与策略，可以根据用户动作的语音响应与非语音响应，做进一步的处理，如下图所示。



对于四类具体话术策略跳转的理解，可以结合下图话术实例来看：



② 电话按键获取

捕获用户的按键信息。电话按键信息的捕获主要是基于 RTP、SIP、SDP 协议对按键的信号解析获得。

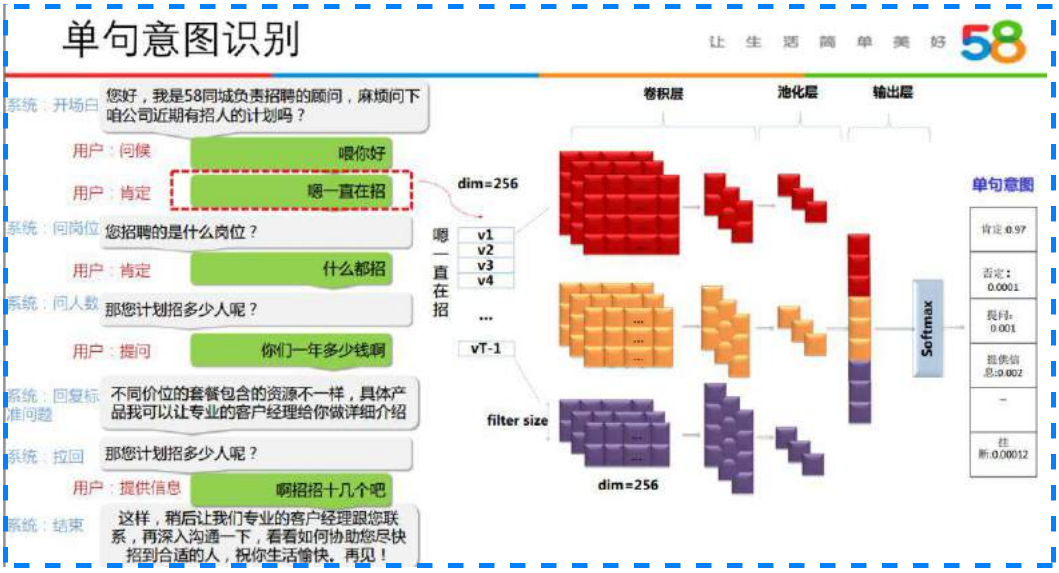
③ 单句意图识别：

用户的意图可能会有很多种，我们将意图整体划分为了十九类意图，并建立了标签、意图及说明的识别表。

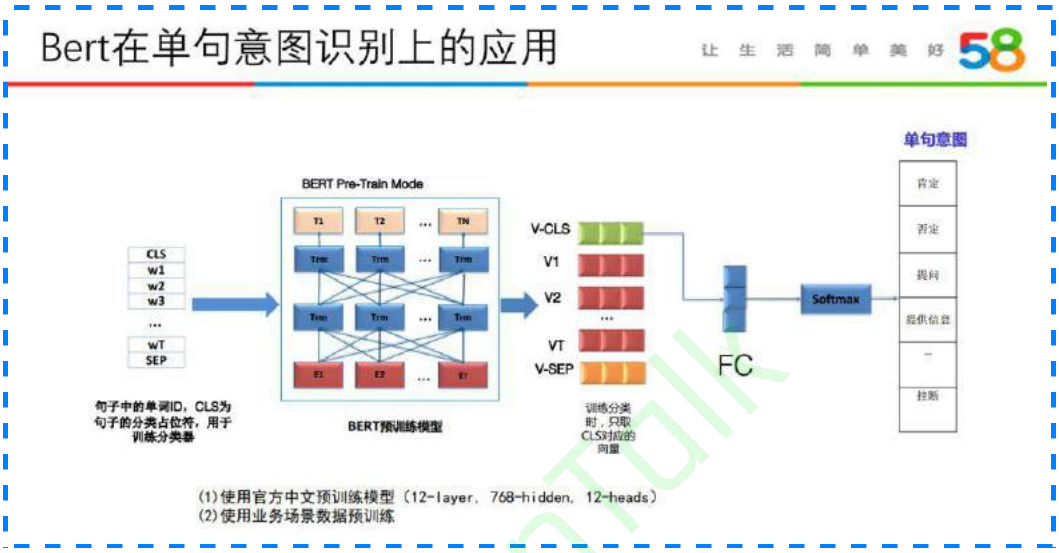
序号	标签	意图	说明
1	affirm	肯定	表达肯定态度，如：“是的。没错。”
2	negate	否定	表达否定态度。如：“不需要、不用了”
3	busy	在忙	对方不方便。如：“我现在没空、我在开车”
4	not_understand	不理解	没听懂说话者的意思。如：“不懂。不明白”
5	keep_listen	继续聆听	表示对方一句话未说完。如：“你”、“我还”
6	inform	提供信息	提供业务相关信息，如：“在北京、在找工作、招司机”
7	request	提问	询问信息。如：“什么是网络推广？价格怎么样？”
8	confirm	确认	确认某信息。如：“是明天吗？不行吗？58同城吗？”
9	request_alts	更换	更换选择。如：“有没有更便宜的？还有其他的么？”
10	request_compare	对比	对比方案，如：“哪个更便宜？去北京还是上海？”
11	offer	提供方案	offer对方或offer自己。如：“已开通**套餐、我注册过了、我已经买过了。”
12	suggest	建议	提供建议。如：“能大点声吗？”
13	hangup	挂断	1. 用户骂人； 2. 用户说“不要再给我打了”； 3. 打错了； 4. 对方也是机器人系统通常会给出结尾语后挂断。
14	goodbye	再见	如：“再见、拜拜”
15	greeting	问候	如：“喂、你好”
16	thankyou	感谢	如：“谢谢”
17	you_are_welcome	客气	如：“没事儿、没关系”
18	sorry	抱歉	如：“不好意思、抱歉、对不起”

单句意图识别的实现：当前公司上线主要用的是 TextCNN 进行多分类，将用户的单句话转换为文本后再进行分类。

DataFunTalk



我们也在尝试使用 Bert 模型，进行单句意图识别：

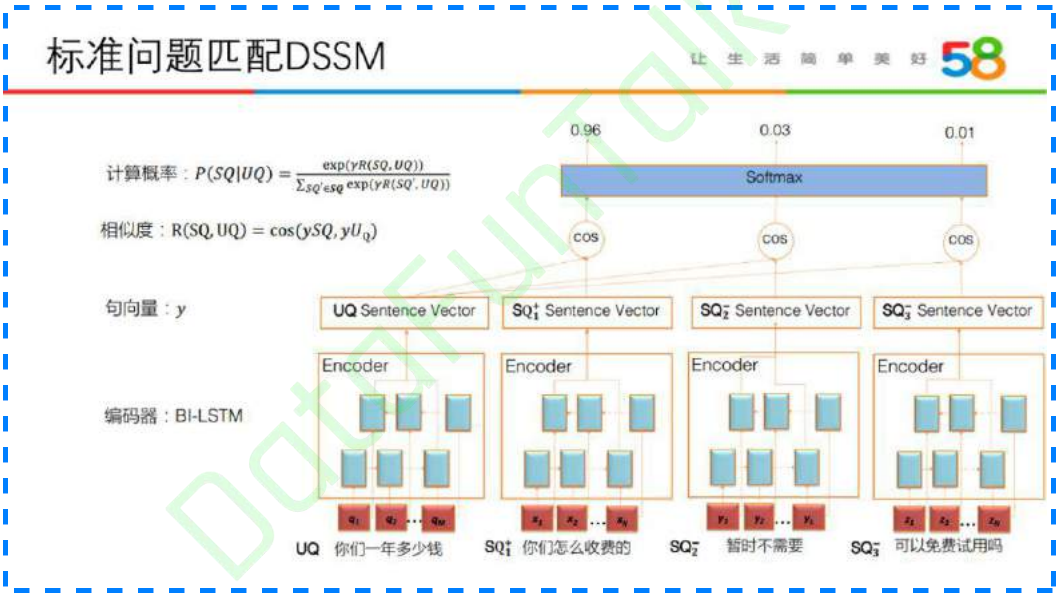


用公司自己的场景语料在谷歌官方中文预训练模型的基础上进行的 Pretrain，拿到 Bert 关于句子的编码向量 CLS 加全连接层后接 softmax 进行多分类。相对于原来 TextCNN 的方式，效果上准确率会有百分之二的提升。

④ 标准问题匹配：

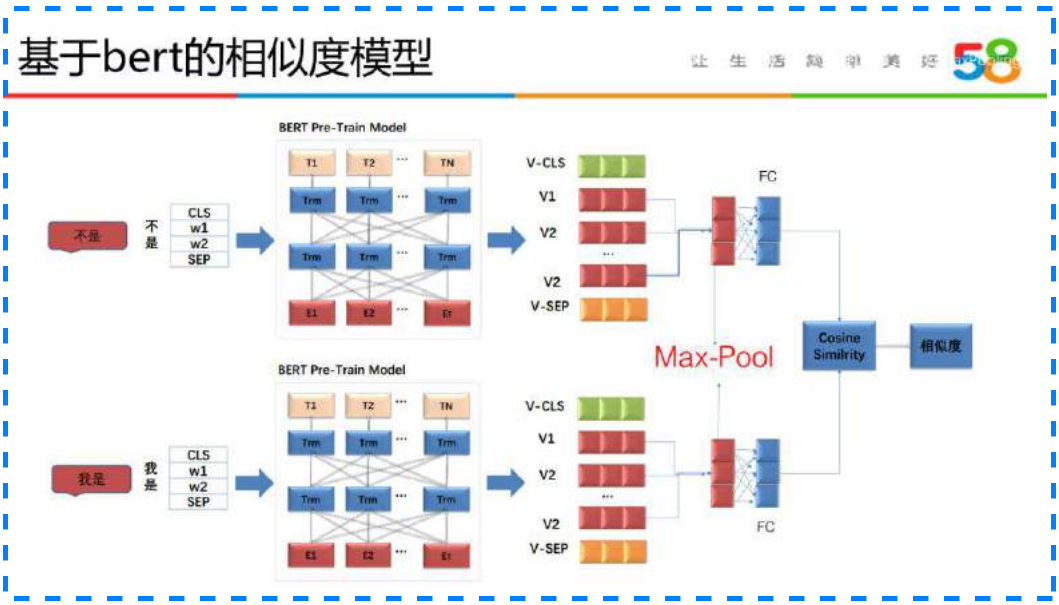
当用户意图为提问时，对于问题类型为标准型问题，可以与标准问题库的问题进行问题匹配，选择对应的答案进行回复。

标准问题匹配当前使用的主要模型是 Bi-LSTM-DSSM：



Bi-LSTM 进行句子的语义编码，DSSM 进行句子匹配。在正负例的选择上，当前主要需要依据具体场景与实验效果进行设定。计算损失时的目标是与正例的相似度越高以及与负例的相似度越低。

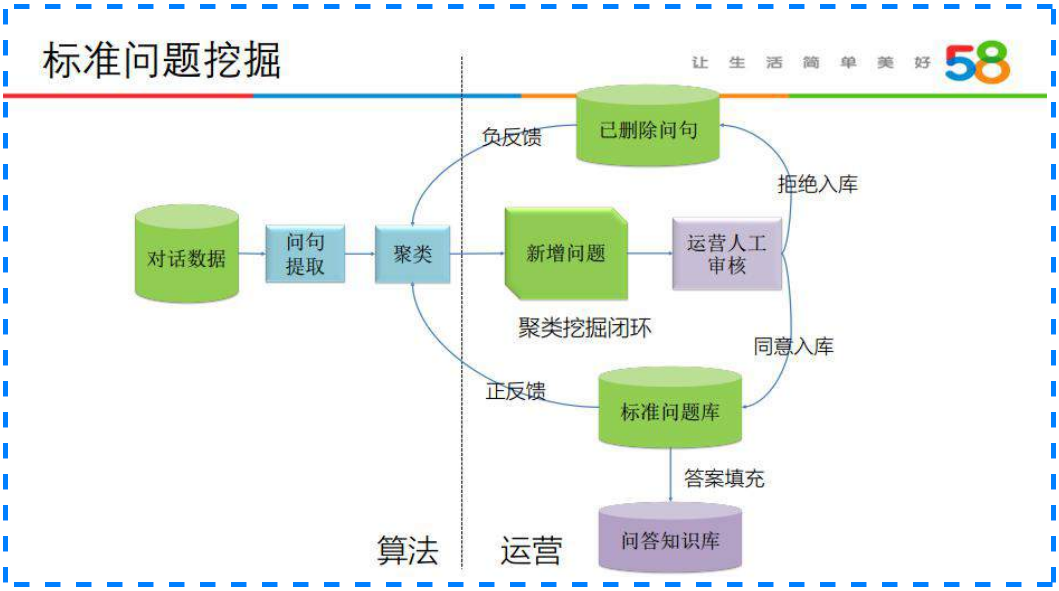
在标准问题匹配上，也有尝试用 Bert 的方式来做：

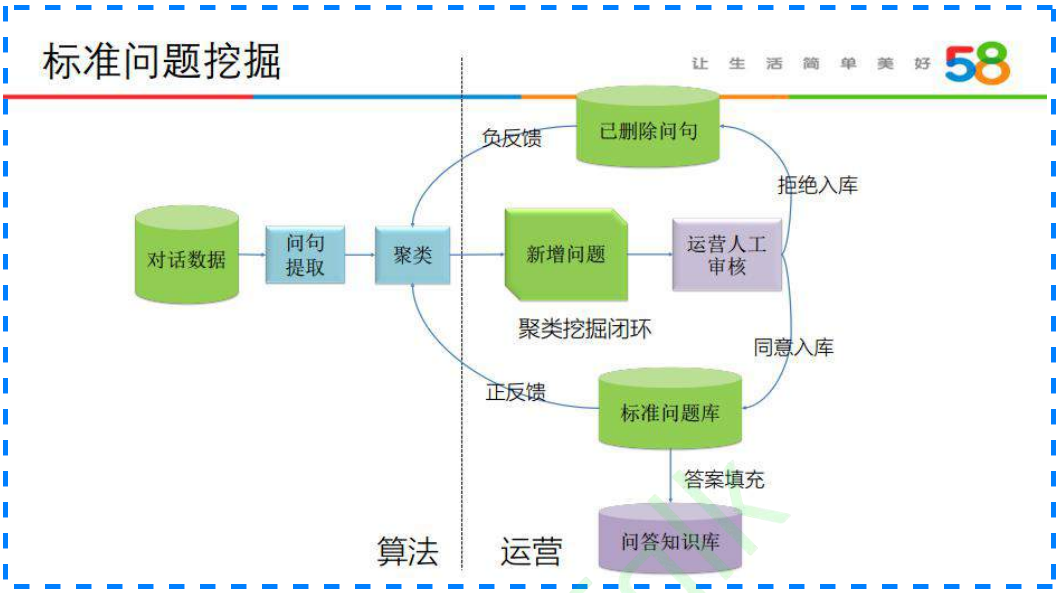


在标准问题的匹配上，Bert 主要是用于句子信息的抽取，在具体实现的时候，对比了 CLS、Mean-Pool、Max-Pool 等常见方式，发现用 Max-Pool 的方式去抽取作为句子的向量表示是最好的。

标准问题挖掘：

完善和更新标准问答数据集。

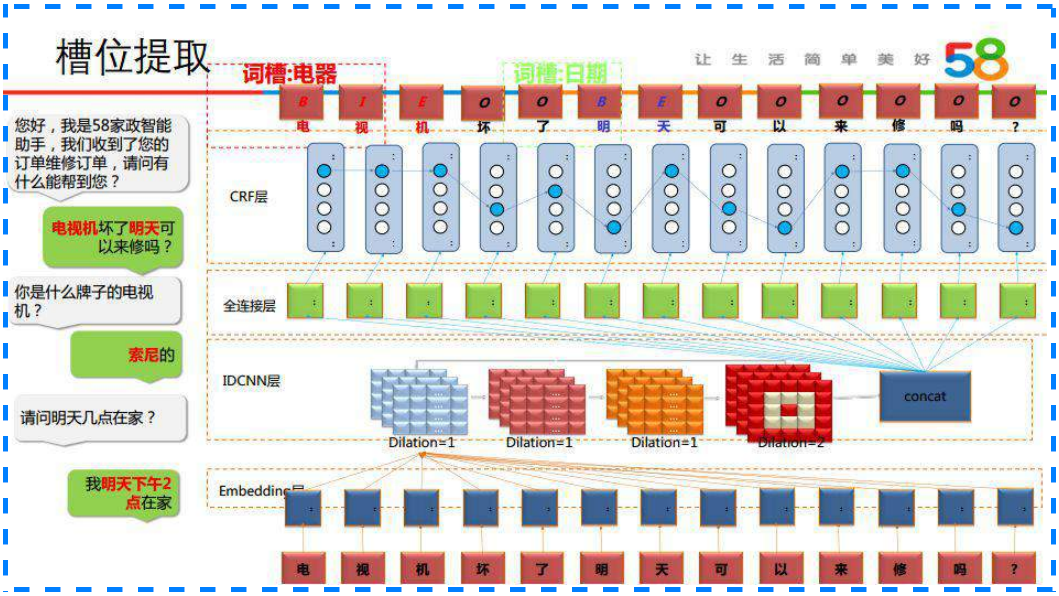




从对话数据中，提取问题与更新问答知识库。这里在新增问题时需要结合人工进行问题入库与否的审核，对于确定后的标准问题，需要对答案填充后才能添加到问答知识库中。

⑤ 槽位提取：

提取对话文本中的词槽，当前主要使用的是 IDCNN+CRF 的方式进行。



当前外呼中提取的词槽实例如下：

词槽实例

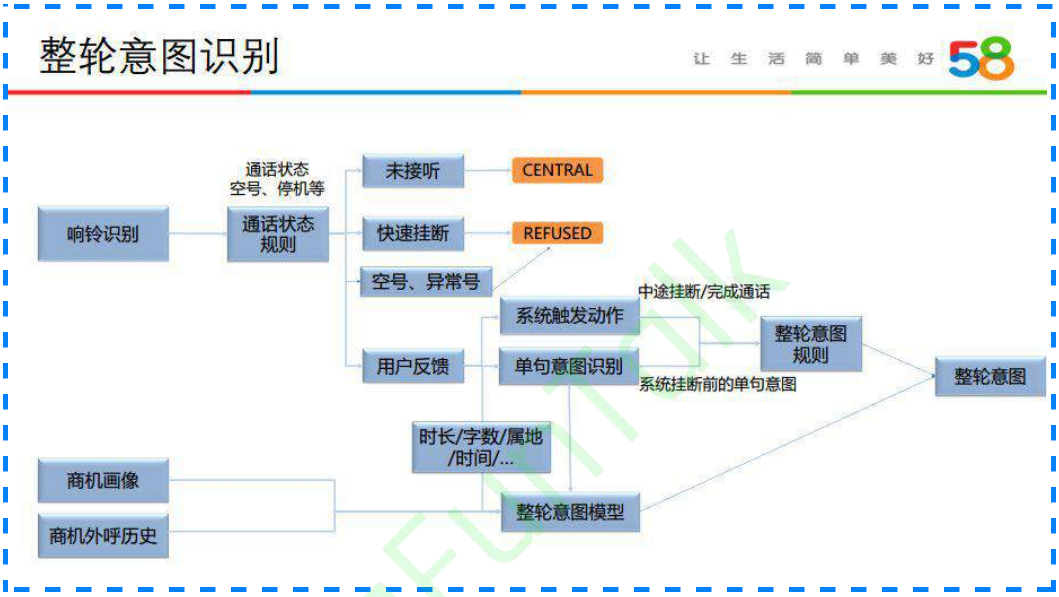
序号	需求内槽位名	说明	实体举例	例句
1	Date	日期	下个月、2019年01月01日、明天等	下个月吧
2	Time	时间	支持大小写，如：10点、下午3点、早上等	明天上午10点
3	Duration	时长	1个小时、两天、一年等	过一个小时再打吧
4	Festival	节日	圣诞节、春节等	春节后再说吧
5	Country	国家	如：中国、德国等	是中国的一个城市
6	Province	省份	山东、辽宁等	我是山东人
7	city	城市	北京、北京市、天津等	我想在北京租房
8	district	区县	包含区划和县镇，如：朝阳区、平遥县等	在朝阳区吧
9	business_circle	商圈	望京、五道口等	我住在望京
10	street	街道	长安街、军民团结路等	我们公司在酒仙桥北路
11	community	小区	小区名，同步了安居客数据	嗯常营镇保利家园
12	facility	周边设施	包含景点、商场等	朝阳公园
13	org	组织机构名	五八同城、安居客、306医院等	我在五八同城
14	loc	通用地点	东北、南方等	我是东北人
15	job	职位	司机、工程师等	我们招司机
16	number	数量（含单位）	一个、两三个人等	招两三个吧
17	tel	电话号	支持阿拉伯数字和中文数据	幺七八二八四五五五八二、15380134531
18	appliance	家用电器	冰箱、洗衣机等	我家冰箱坏了

4. 整轮意图识别

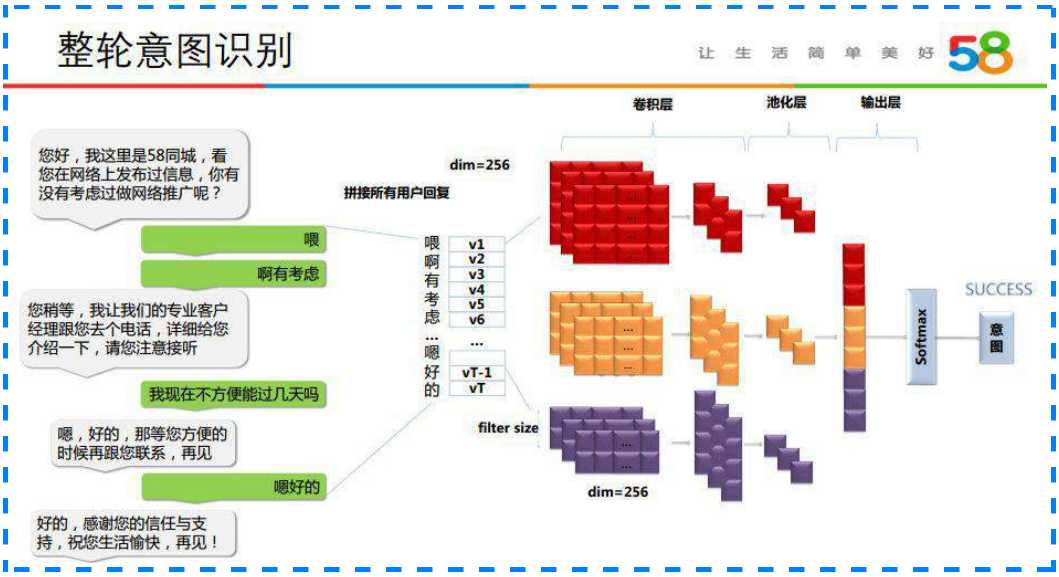
在部分外呼业务场景下，当外呼电话结束时，需要对客户意图进行预测，目前采用整轮意图识别的方式对客户意图进行预测。比如在销售场景下，将客户的意图分成了三类分别为 SUCCESS、CENTRAL、REFUSED，具体意图说明如下表：

意图标签	意图	说明
SUCCESS	成功邀约	用户接受方案，可以进一步联系
CENTRAL	中性态度，可后续跟进	用户在忙或用户还在考虑
REFUSED	明确拒绝	用户确实不需要

整轮意图识别会考虑整个对话从响铃识别到对话结束的相关信息来判定，具体可以见下图：

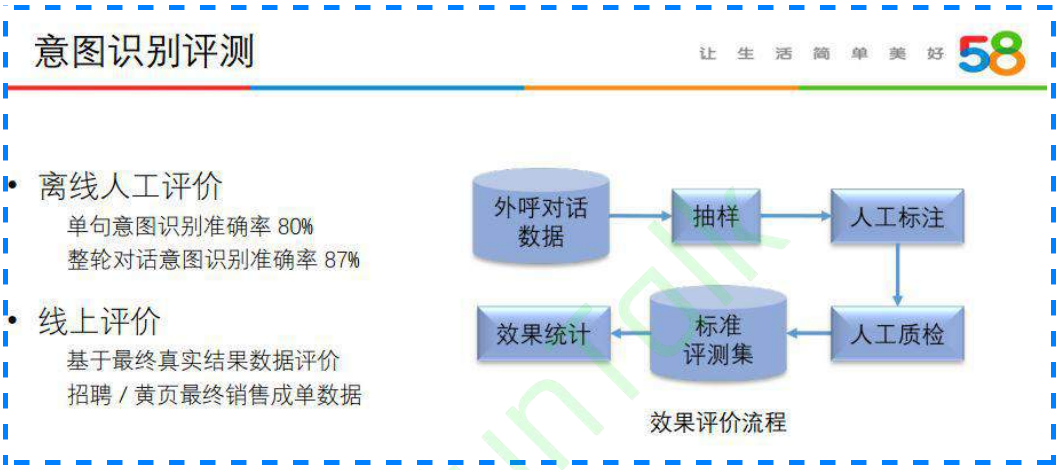


具体的意图识别算法，主要包括 TextCNN 等模型进行分类，输入为整个对话过程中用户的所有回复文本的拼接，当前实际上线的模型是采用多模型融合来做的。此处以 TextCNN 单模型为例，模型结构如下图：



意图识别的评测：

评测会包括离线人工评价与线上评价。



DataFunTalk

目前影响意图识别准确率的主要因素是语音识别错误。

——应用场景——

语音机器人的实用场景不仅在是在销售场景，在 58 同城的应用主要还可以概括六个方面，包括：

① 通知：最基本的场景，如当用户信息发生更改，机器人可以用语音的方式告知用户哪些信息发生了更改，需要注意哪些特别事情等。

② 满意度回访：如使用 58 同城的产品时，可以利用语音机器人对客户进行产品满意度的回访。

③ 信息验真：进行用户信息真实与否的确认。

④ 销售：判断与发现潜在客户。

⑤ 告警：比如内部运维场景下，服务器出现了异常情况，机器人语音电话将报警信息告诉相关负责人。

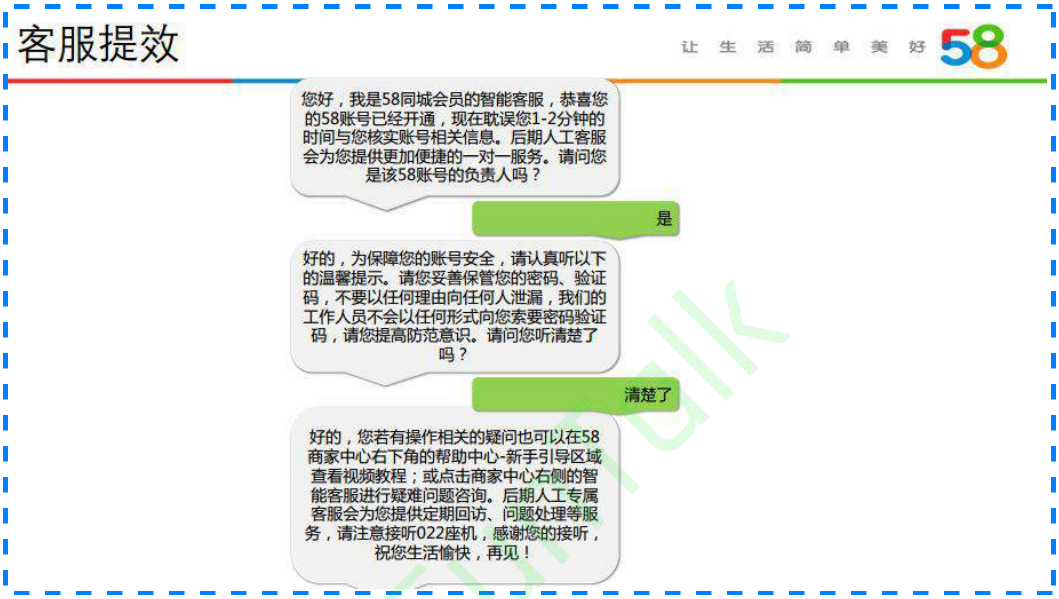
⑥ 销售客服培训：在人力客服上岗前，需要先进行上岗培训，原来主管对于新员工的基础培训与考核部分，可以改为利用机器人进行初步培训。

下面可以看看四个具体的实际应用案例：

DataFunTalk

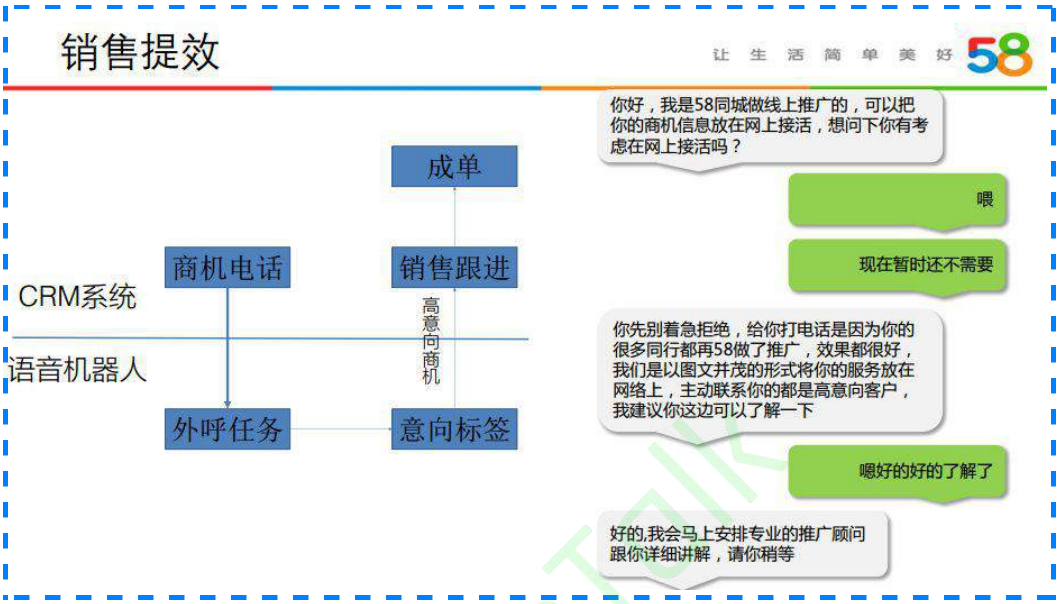
① 校招提效：通知场景，语音机器人在校招面试中成功运用，提升校招工作人员的效率。

② 客服提效



③ 运营提效

④ 销售提效



利用语音机器人识别意向程度高的潜在客户，提升销售人员的销售效率。

小结：语音对话技术不仅在销售业务领域落地，在信息通知、内部的业务告警等多个业务模块都得到了应用。本文围绕人机语音对话技术在 58 同城的落地场景，对语音机器人的总体架构、核心功能、具体实践场景进行了系统的介绍。

DataFunTalk

人机对话技术研究进展与思考

嘉宾：袁彩霞　编辑：Hoh Xil

导读：本次分享的主题为**人机对话技术研究进展与思考。**主要梳理了我们团队近两年的工作，渴望可以通过这样的介绍，能给大家一个关于人机对话（包括它的科学问题和应用技术）方面的启示，帮助我们进行更深入的研究和讨论。主要包括：

1. Spoken dialogue system：a bird view（首先我们来看什么是人机对话，尤其是 Spoken dialogue。其实说 Spoken 的时候，有两层含义：第一个 spoken 就是 speech，第二个我们处理的语言本身具有 spoken 的特性。但是，稍后会讲的 spoken 是指我们已经进行语音识别之后，转换为文本的一个特殊的自然语言，后面讨论的口语对话不过多地讨论它的口语特性，主要是讲人和机器之间的自然语言对话。）

2. X-driven dialogue system：紧接着来讲解我们近些年的研究主线 X-driven dialogue syatem，X 指构建一个对话系统时，所采用的数据是什么，从最早的 dialogue -> FAQ -> KB -> KG -> document 以及我们一直在尝试的图文多模态数据。

3. Concluding remarks（结束语）

DataFunTalk

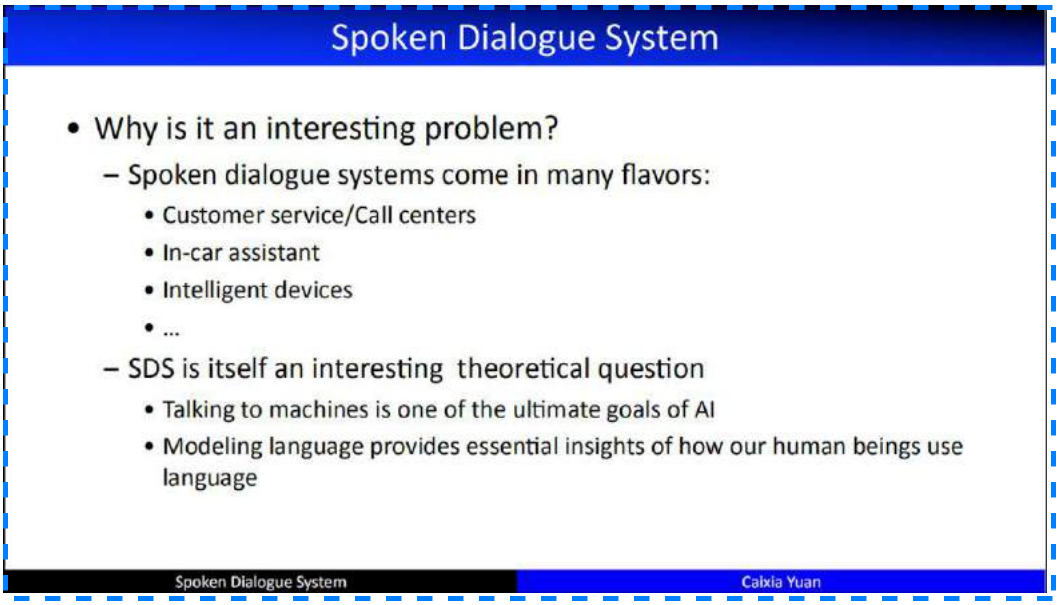
01

Spoken dialogue system : a bird view

学术界关于对话系统有着不同的划分，这种划分目前看来不是非常准确，也不是特别标准的划分了。但是，接下来的内容，主要是围绕着这两个主线：

限定领域，专门指任务型对话（ 围绕某一特定用户对话目标而展开的 ）。对于任务型对话，对话系统的优化目标就是如何以一个特别高的回报、特别少的对话轮次、特别高的成功率来达成用户的对话目标。所以即便是限定领域，我们这里讨论的也是特别限定的、专门有明确的用户对话目标的一种对话。

开放领域，not purely task-oriented， 已经不再是纯粹的对话目标驱动的对话，包括：闲聊、推荐、信息服务等等，后面逐步展开介绍。

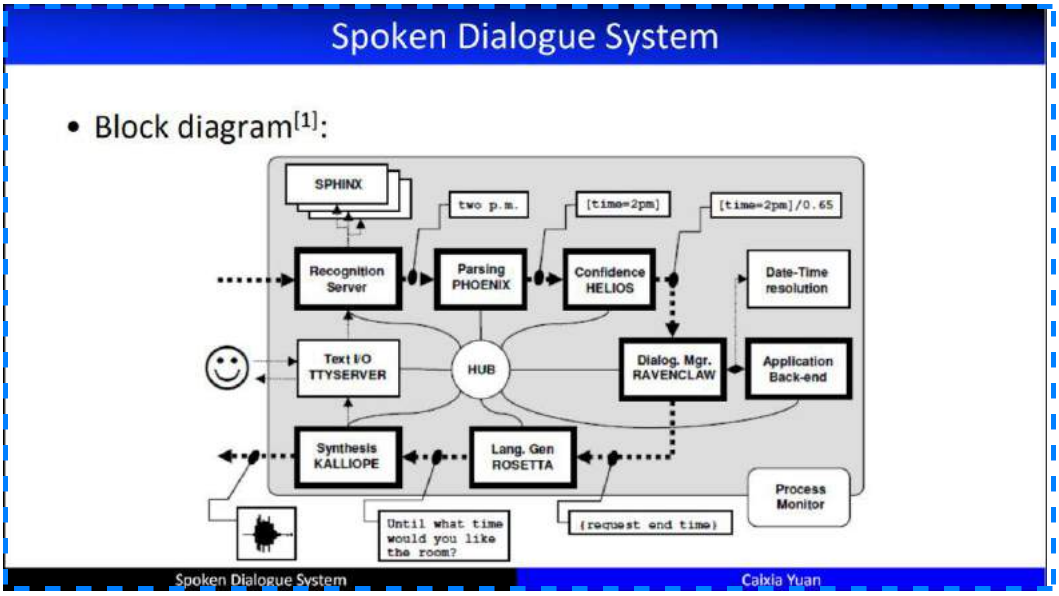


我们在研究一个问题或者做论文答辩和开题报告时，经常讨论研究对象的意义在哪里。图中，前面讲的是应用意义，后面是理论意义。我们实验室在北京邮电大学叫智能科学与技术实验室，其实她的前身叫人工智能实验室。所以从名字来看，我们做了非常多的 AI 基础理论的研究，我们在研究这些理论的时候，也会讲 AI 的终极目的是研制一种能够从事人类思维活动的计算机系统。人类思维活动建立在获取到的信号的基础上。人类获取信号的方式大体有五类，包括视觉、听觉、触觉、味觉、嗅觉等，其中视觉和听觉是两个比较高级的传感器通道，尤其是视觉通道，占据了人类获得信息的 80%以上。所以我们从这两个角度，设立了两个研究对象：第一个是语言，第二个是图像。而我们在研究语言的时候，发现语言有一个重要的属性，叫交互性，

交互性最典型的一个体现就是对话；同时，语言不是一个独立的模态，语言的处理离不开跟它相关的另一个通道，就是视觉通道。所以我们早期更多是为了把交互和多模态这样的属性纳入到语言建模的范围，以其提升其它自然语言处理系统的性能，这就是我们研究的一个动机。

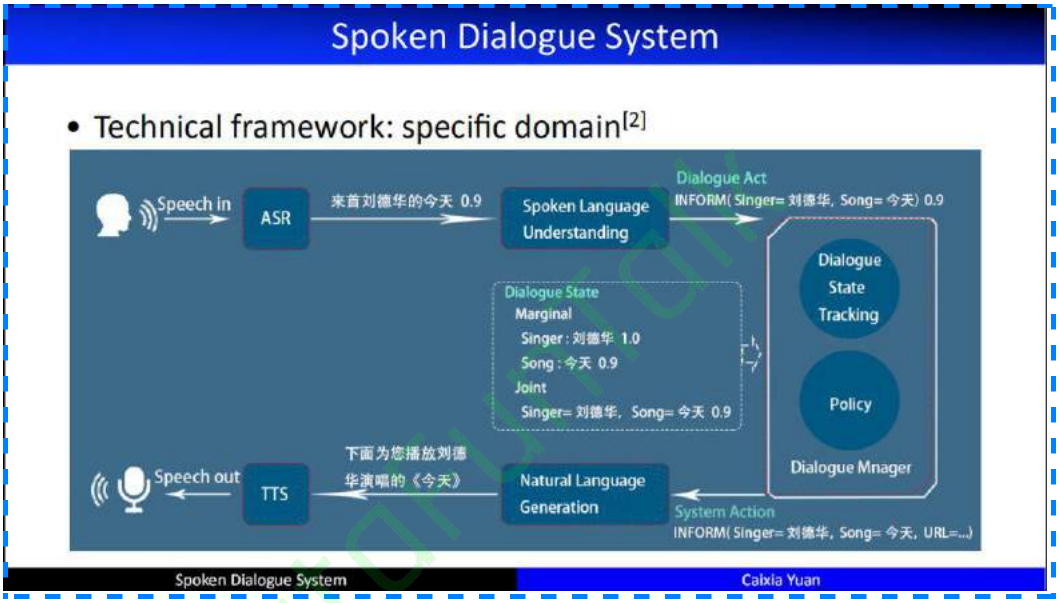
1. Block diagram

上图为 CMU 等在 1997 年提出来的人机对话框架，基于这个框架人们开发出了非常多优秀的应用系统，比如应用天气领域的 "Jupiter"。这个框架从提出到商业化应用，一直到今天，我们都还沿着这样的一个系统架构在进行开发，尤其是任务驱动的对话。



这就是具体的对话系统的技术架构。

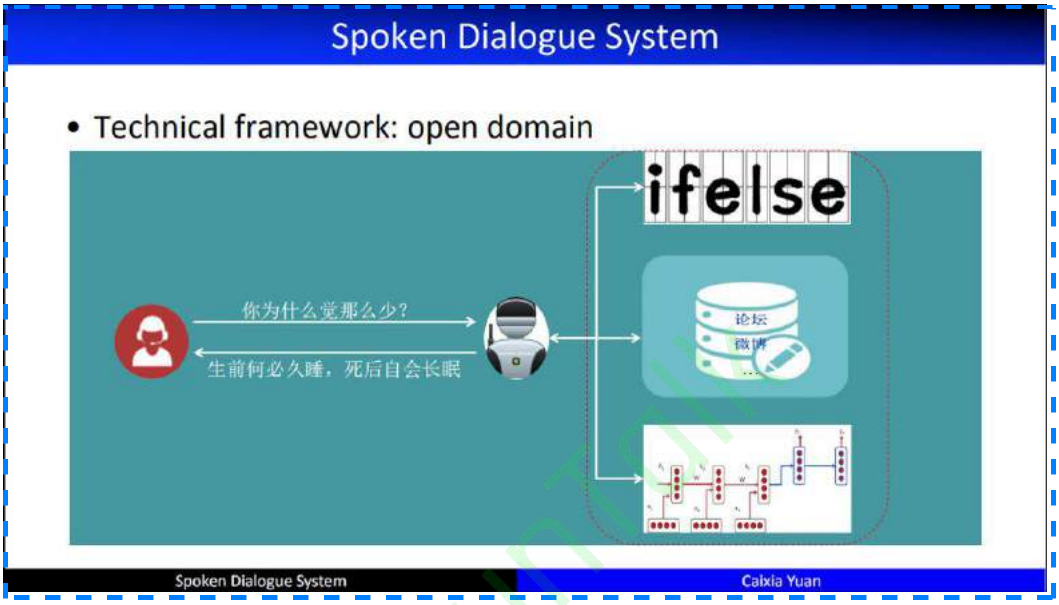
2. Specific domain



这个架构发展到现在，在功能模块上，已经有了一个很清晰的划分：

首先进行语音识别，然后自然语言理解，紧接着做对话管理，将对话管理的输出交给自然语言生成模块，最后形成自然语言应答返回给用户。这就是一个最典型的 specific domain 的架构。早期 task 限定的 dialogue，基本上都是按照这个架构来做的。这个架构虽然是一个 Pipeline，但是从研究的角度来讲，每一个模块和其它模块之间都会存在依赖关系。因此，我们试图从研究的角度把不同的功能模块进行统一建模。在这个建模过程中，又会产生新的学术性问题，我们旨在在这样的问题上可以产生驱动性的技术。

3. Open domain



Open domain，也就是“闲聊”，实现上主要分为途径：

第一个是基于匹配/规则的闲聊系统；第二个是基于检索的闲聊系统；第三个是基于编解码结构的端到端对话系统。当然，实际情境中，这几个途径往往结合在一起使用。

02

X-Driven dialogue system

目前无论是任务型对话还是闲聊式对话，都采用数据驱动的方法，因此依据在构建人机对话系统时所用到的数据不同，建模技术和系统特性也就体现出巨大的不同。我们把使用的数据记为 X，于是就有了不同的 X 驱动的对话。

1. Our roadmap

如果想让机器学会像人一样对话，我们可以提供的最自然的数据就是 dialogue。我们从 2003 年开始做对话驱动的对话；2012 年开始做 FAQ 驱动的对话；2015 年开始做知识库（KB）驱动的对话；2016 年开始做知识图谱（KG）驱动的对话，相比于 KB，KG 中的知识点产生了关联，有了这种关联人们就可以在大规模的图谱上做知识推理；2017 年开始做文档驱动的对话。这就是我们研究的大致脉络。

2. Dialogue-driven dialogue

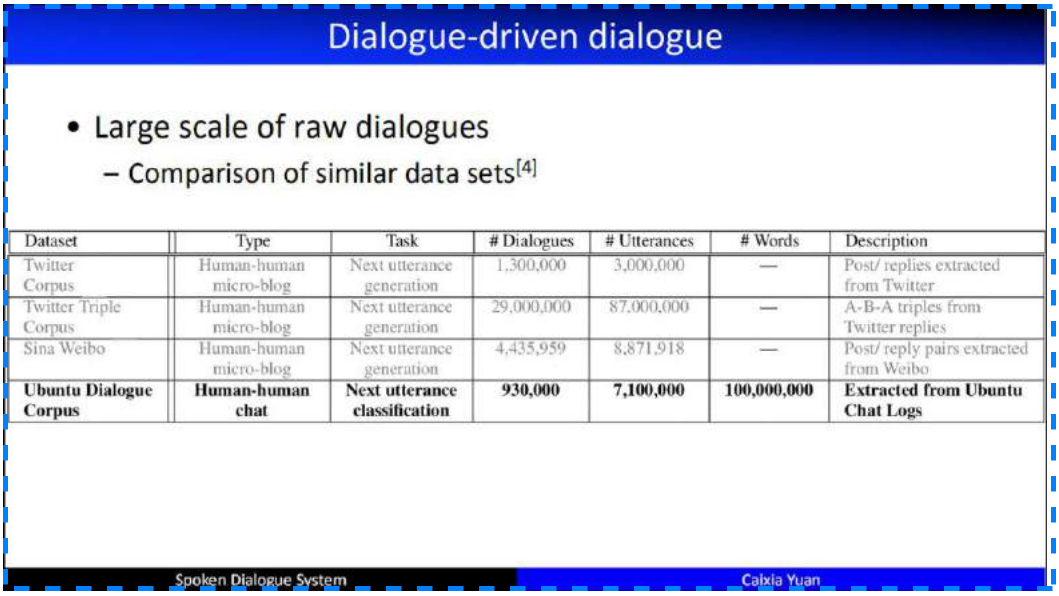
Dialogue-driven dialogue

- Large scale of fully-labeled dialogues
  - Comparison of similar data sets[3]

Metric	DSTC2	SFX	WOZ2.0	FRAMES	KVRET	M2M	MultiWOZ
# Dialogues	1,612	1,006	600	1,369	2,425	1,500	**8,438**
Total # turns	23,354	12,396	4,472	19,986	12,732	14,796	**113, 556**
Total # tokens	199,431	108,975	50,264	251,867	102,077	121,977	**1,490,615**
Avg. turns per dialogue	14.49	12.32	7.45	**14.60**	5.25	9.86	13.46
Avg. tokens per turn	8.54	8.79	11.24	12.60	8.02	8.24	**13.13**
Total unique tokens	986	1,473	2,142	12,043	2,842	1,008	**23689**
# Slots	8	14	4	**61**	13	14	24
# Values	212	1847	99	3871	1363	138	**4510**

Spoken Dialogue System　　Caixia Yuan

早期在做 Dialogue driven 的时候，多依赖人工采集数据，但是，从 2013 年以来，逐步开放了丰富的涵盖多领域多场景的公开数据集。比如最近的 MultiWOZ，从 task specific 角度讲，数据质量足够好、数据规模足够大，同时涵盖的对话情景也非常丰富。但是，目前公开的中文数据集还不是很多。

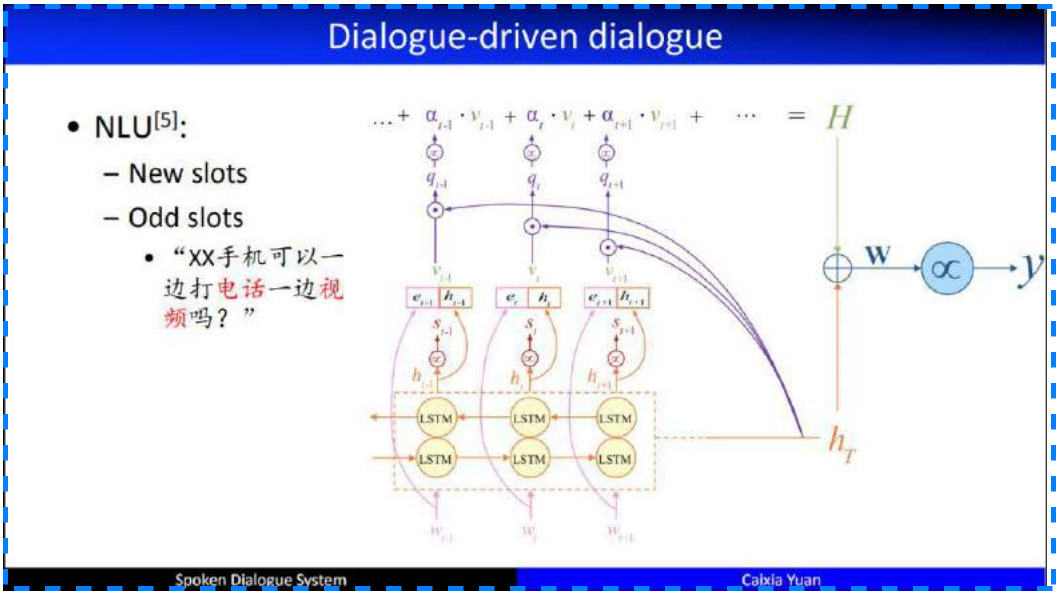


这个是和任务型对话无关的数据集，也就是采集的人与人对话的数据集。尤其以 Ubuntu 为例，从 15 年更新至今，已经积累了非常大规模的数据。

以 Dialogue 为输入，我们开展了任务型和非任务型两个方向的工作。先来看下任务型对话：

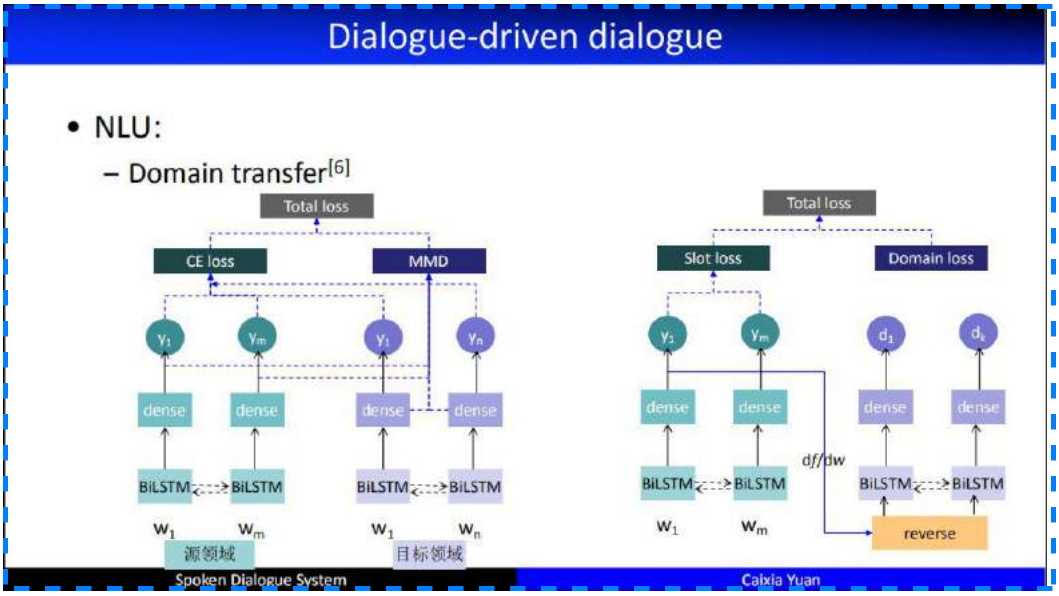
2.1 NLU

DataFunTalk



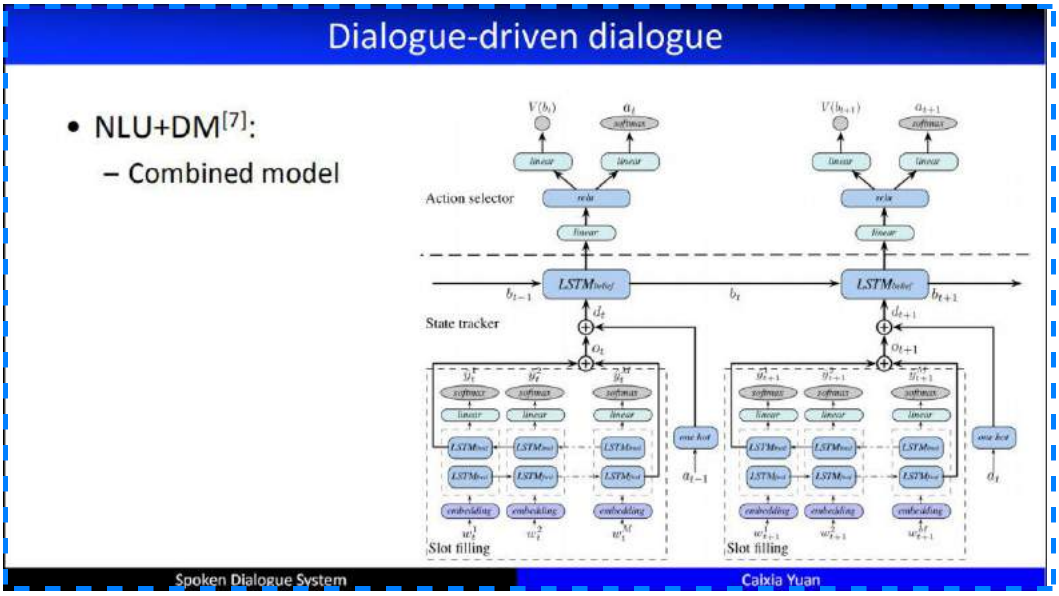
当一个用户输入过来，第一个要做的就是自然语言理解（NLU），NLU 要做的三件事为：Domain 识别；Intent 识别；信息槽识别或叫槽填充。这三个任务可以分别独立地或采用管道式方法做，也可以联合起来进行建模。在联合建模以外，我们还做了一些特别的研究。比如我们在槽识别的时候，总是有新槽，再比如有些槽值非常奇怪，例如 "XX 手机可以一边打电话一边视频吗？"，对应着槽值 "视频电话"，采用序列标注的方式，很难识别它，因为这个槽值非常不规范。用户输入可能像这样语义非常松散，不连续，也可能存在非常多噪音，在进行联合建模时，传统的序列标注或分类为思想，在实际应用中已经不足以解决问题了。

我们针对这个问题做了比较多的探讨，右图为我们 2015 年的一个工作：在这三个任务联合建模的同时，在槽填充这个任务上将序列标注和分类进行同时建模，来更好地完成 NLU。

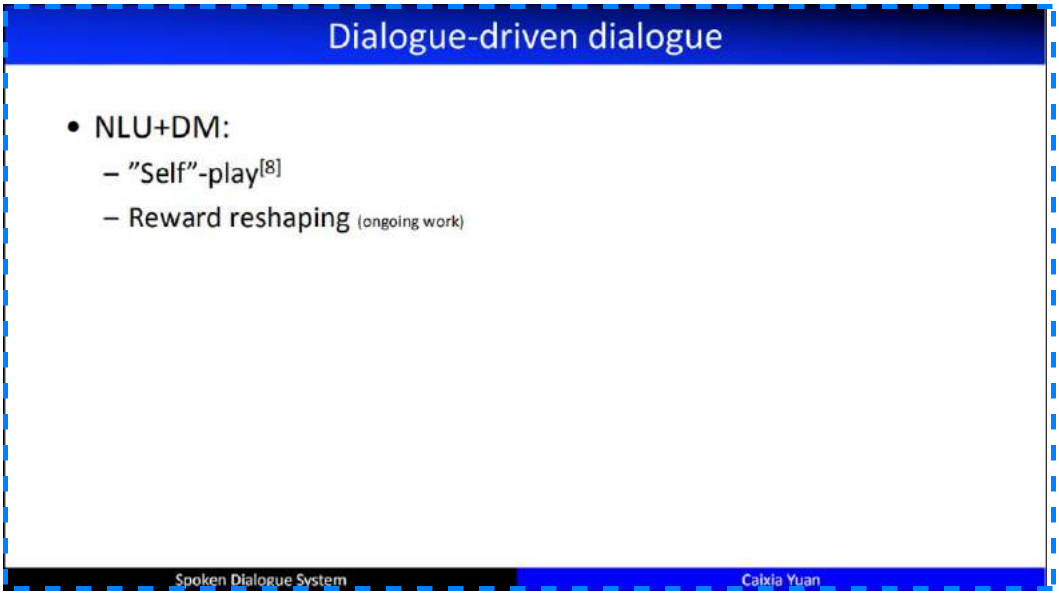


在 NLU 领域还有一个非常重要的问题，随着开发的业务领域越来越多，我们发现多领域对话产生了诸多非常重要的问题，例如在数据层有些 domain 数据可能很多，有些 domain 数据可能很少，甚至没有，于是就遇到冷启动的问题。因此，我们做了非常多的 domain transfer 的工作。上图为我们 2016 年的一个工作，我们会把数据比较多的看成源领域，数据比较少的看成目标领域。于是，尝试了基于多种迁移学习的 NLU，有的是在特征层进行迁移，有的是在数据层迁移，有的是在模型层进行迁移。图中是两个典型的在特征层进行迁移的例子，不仅关注领域一般特征，而且关注领域专门特征，同时采用了对抗网络来生成一个虚拟的特征集的模型。

2.2 NLU+DM



紧接着，我们研究了 NLU 和对话管理（DM）进行联合建模，因为我们发现人人对话的时候，不见得是听完对方说完话，理解了对方的意图，然后才形成对话策略，有可能这两个过程是同时发生的。甚或 DM 还可以反作用于 NLU。早期我们基于的一个假设是， NLU 可能不需要一个显式的过程，甚至不需要一个显式的 NLU 的功能，我们认为 NLU 最终是服务于对话管理（DM），甚至就是对话管理（DM）的一部分。所以，2013 年的时候，我们开始了探索，有两位特别优秀的毕业生在这两个方面做了特别多的工作。比如，如何更好地联合建模语言理解的输出和对话管理的策略优化。这是我们在 NLU 和 DM 联合建模的工作，同时提升了 NLU 和 DM 的性能。



在联合模型中，我们发现，DM 的建模涉及到非常多的 DRL（ 深度强化学习 ）的工作，训练起来非常困难，比如如何设计一个好的用户模拟器，基于规则的，基于统计的，基于语言模型的，基于 RL 的等等我们尝试了非常多的办法，也取得了一系列有趣的发现。2018 年时我们研究一种不依赖于规则的用户模拟器，业界管这个问题叫做 "Self"-play，虽然我们和 "Self"-play 在网络结构上差异挺大的，但是我们还是借鉴了 "Self"-play 训练的特性，把我们自己的系统叫做 "Self"-play。在这样的机制引导下，我们研究了不依赖于规则，不依赖于有标记数据的用户模拟器，使得这个用户模拟器可以像 Agent 一样，和我们所构造的对话的 Agent 进行交互，在交互的过程中完成对用户的模拟。

在训练过程中还有一个重要的问题，就是 reward 怎么来，我们知道在 task oriented 时，reward 通常是人类专家根据业务逻辑/规范制定出来的。事实

上，当我们在和环境交互的时候不知道 reward 有多大，但是环境会隐式地告诉我们 reward 是多大，所以我们做了非常多的临接对和 reward reshaping 的工作。

2.3 小结

Dialogue-driven dialogue 这种形式的对话系统，总结来看：

优点：

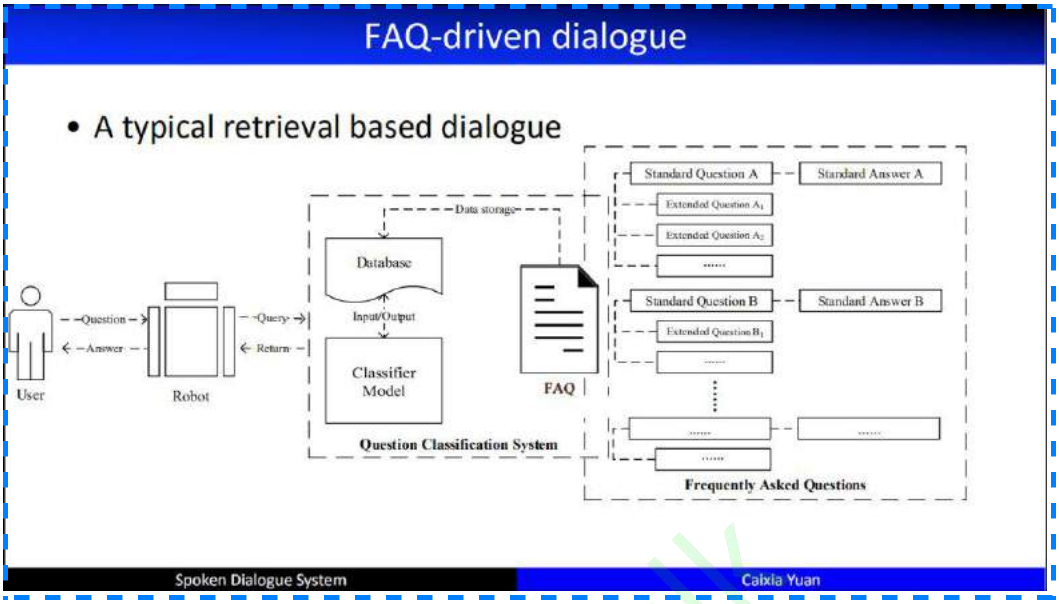
定义非常好，逻辑清晰，每一个模块的输入输出也非常清晰，同时有特别坚实的数学模型可以对它进行建模。

缺点：

由于非常依赖数据，同时，不论是在 NLU 还是 NLG 时，我们都是采用有监督的模型来做的，所以它依赖于大量的、精细的标注数据。

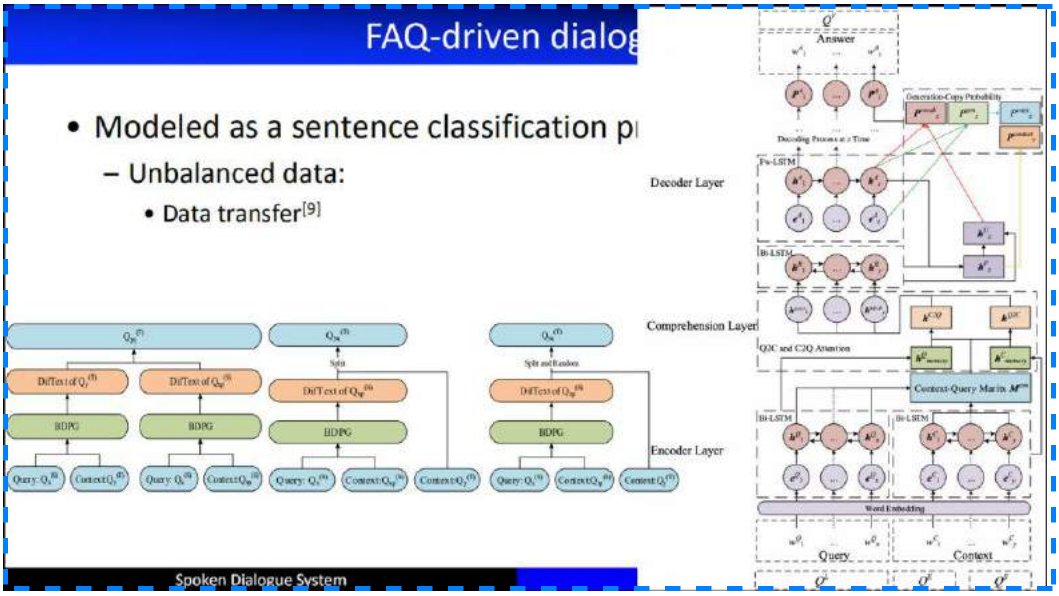
而 DM 往往采用 DRL 来做。NIPS2018 时的一个 talk，直接指出：任何一个 RL 都存在的问题，就是糟糕的重现性、复用性、鲁棒性。

3. FAQ-driven dialogue



FAQ 是工业界非常常见的一种情景：有大量的标准问，以及这个标准问的答案是什么。基于这个标准问，一个用户的问题来了，如何找到和它相似的问题，进而把答案返回给用户，于是这个 Service 就结束了。

实际中，我们如何建 FAQ？更多的时候，我会把这个问题和我们库的标准问题做一个相似度的计算或者做一个分类。

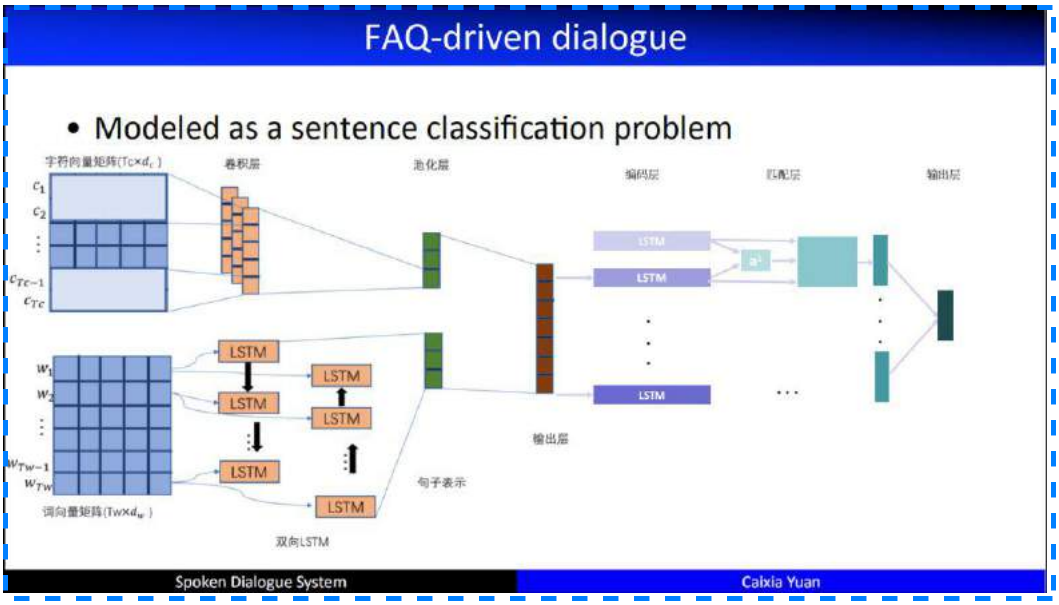


我们在做这个工作的时候发现一个特别大的问题，就是 Unbalanced Data 问题。比如，我们有 5000 个问题，每个问题都有标准答案，有些问题可能对应的用户问题特别多，比如 "屏幕碎了" 可能会有 1000 多种不同的问法，还有些问题，可能在几年的时间里都没有人问到过。所以，面对数据不均衡的问题，我们从 2016 年开始做了 Data transfer 的工作。

大致的思路是：我有一个标准问题，但是很糟糕，这个标准问题没有用户问题，也就是没有训练语料。接下来发现另外一个和这个标准问很相似的其它标准问有很多的训练语料，于是借助这个标准问，来生成虚拟样本，进而削弱了 Unbalance。

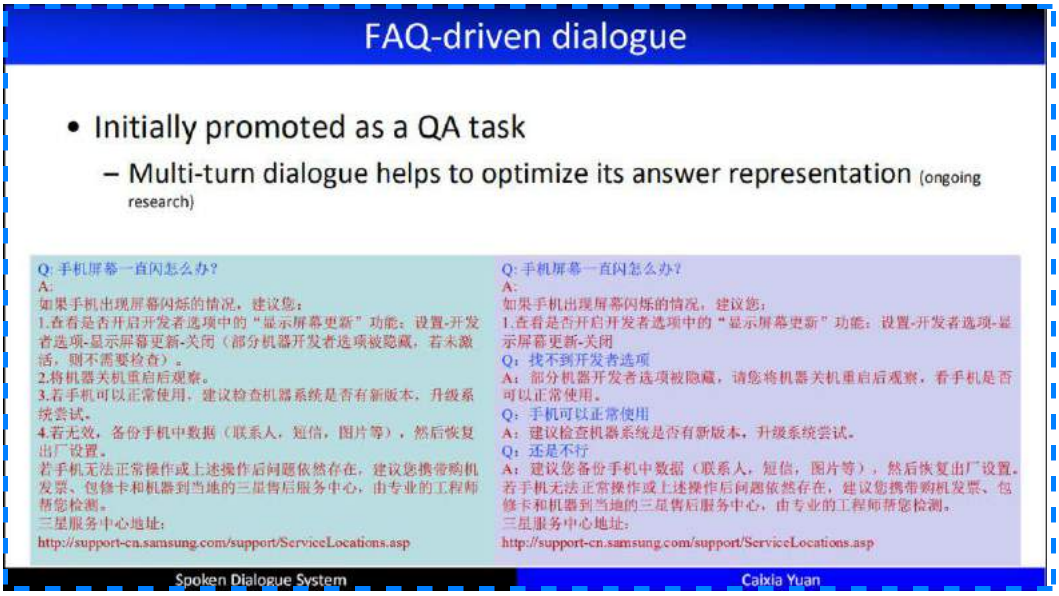
具体的方法：我们把目标领域的标准问看成 Query，把和它相似的标准问题及其对应的用户问题看成 Context，采用了 MRC 机器阅读理解的架构来生

成一个答案，作为目标问题的虚拟的用户问题，取得了非常好的效果，并且尝试了三种不同的生成用户问题的方法。



实际项目中，FAQ 中的 Q 可能有非常多的问题，例如 3000 多个类，需要做极限分类，这就导致性能低下，且非常耗时，不能快速响应用户的问题。于是我们做了一个匹配和分类进行交互的 model，取得了不错的效果。

DataFunTalk



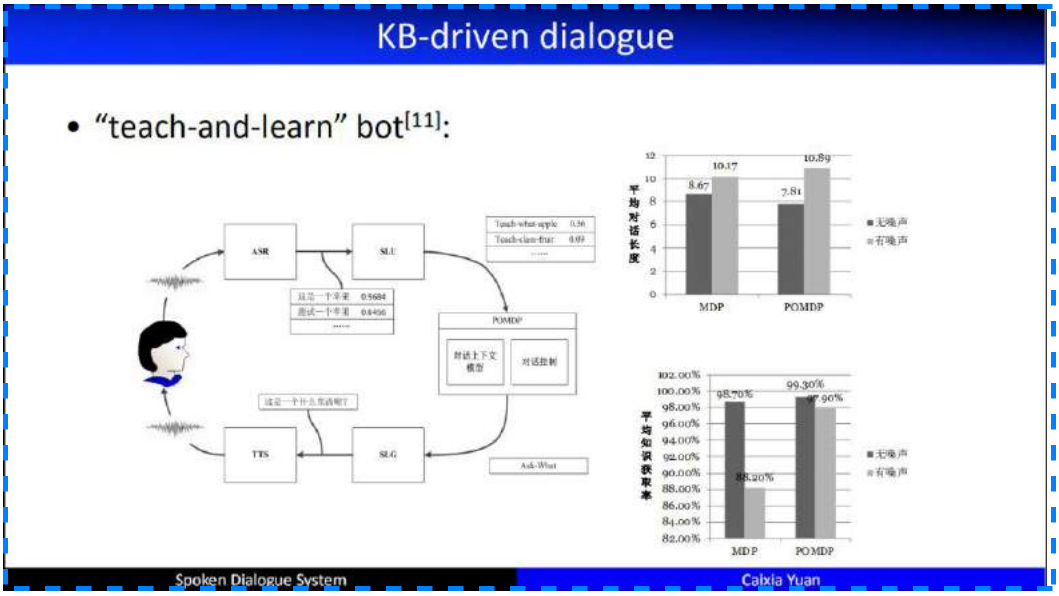
目前，大部分人都认为 FAQ 驱动的 dialogue 不叫 dialogue，因为我们通常说的 dialogue 轮次是大于两轮的。而 FAQ 就是一个 QA 系统，没有交互性。有时候带来的用户体验非常不友好，比如当答案非常长的时候，系统要把长长的答案返回，就会一直讲，导致用户比较差的体验。于是，我们基于 FAQ 发展出了一个多轮对话的数据，如右图，这是我们正在开展的一个工作。

4. KB-driven dialogue

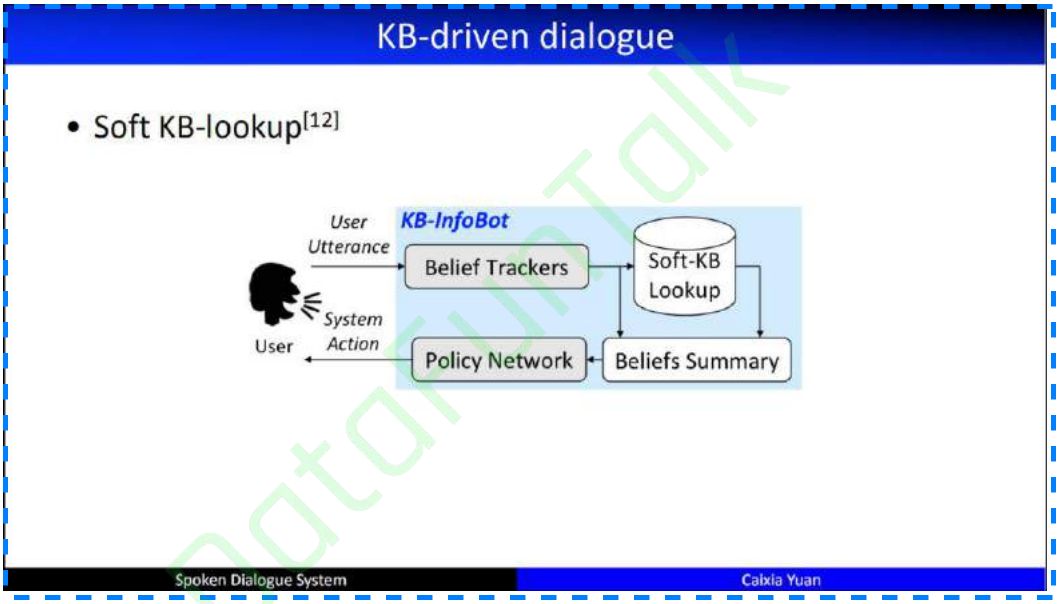
DataFunTalk

KB 最早人们认为它就是一个结构化的数据库，通常存储在关系型数据库中。比如要订一个酒店，这个酒店有各种属性，如位置、名称、户型、价格、面积等等。早期 CMU 的对话系统，所有的模块都要和 Hub 进行交互，最后 Hub 和后端的数据库进行交互。数据库的价值非常大，但是早期人们在建模人机对话的时候，都忽视了数据库。这里就会存在一个问题：机器和用户交互了很久，而在检索数据库时发现没有答案，或者答案非常多，造成用户体验非常糟糕。

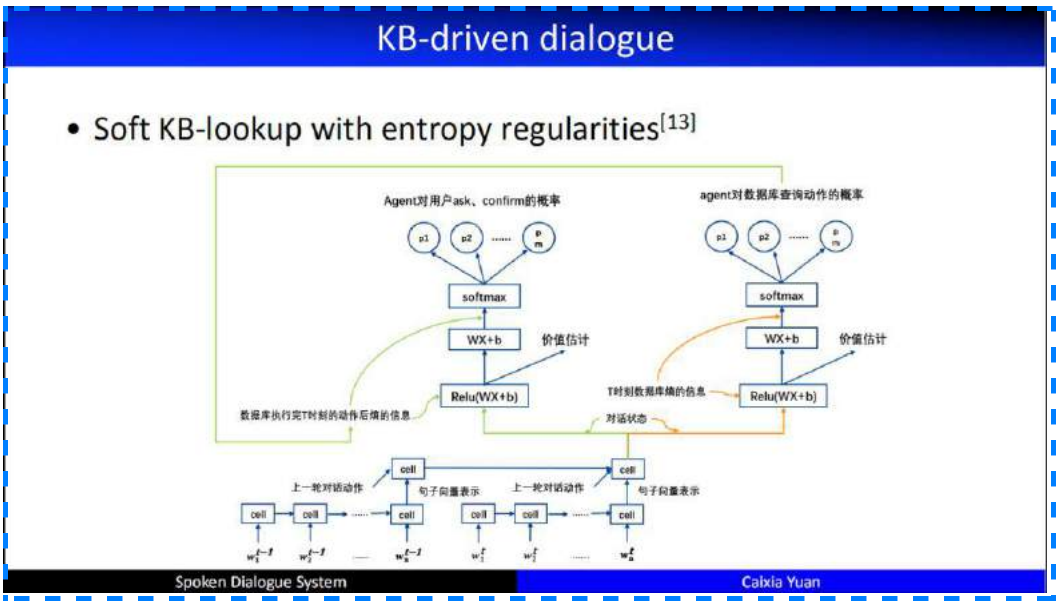
DataFunTalk



从 2012 年开始，我们开始把 KB 引入我们的对话系统。图中的对话系统叫做 "teach-and-learn" bot，这是一个多模态的对话，但是每个涉及到的 object，我们都会把它放到 DB 中。和用户交互过程中，不光看用户的对话状态，还要看数据库状态。这个想法把工作往前推进了一些。

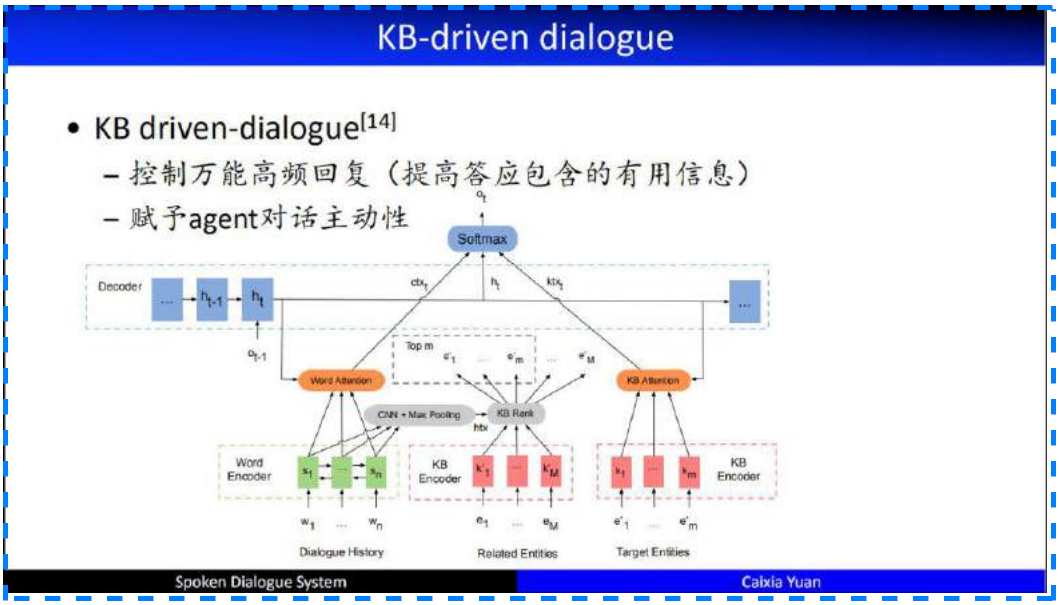


直到 2016 年，MSR 提出的 KB-InfoBot，第一次提出了进行数据库操作时，要考虑它的可导性，否则，就没办法在 RL 网络中像其它的 Agent action 一样进行求导。具体的思路：把数据库的查询和 Belief State 一起总结起来做同一个 belief，进而在这样的 belief 基础上做各种对话策略的优化。



在上述方法的基础上，我们做了有效的改良，包括 entropy regularities 工作。是每次和用户进行交互时，数据库的 entropy 会发生变化。比如当机器问 "你想订哪里的酒店？"，用户答 "阿里中心附近的。"，于是数据库立刻进行了一次 entropy 计算进行更新，接着继续问 "你想订哪一天的？"，用户答 "订 7 月 28 号的"，于是又进行了一次 entropy 计算进行更新。这样在和用户进行交互的时候，不光看用户的 dialogue 输入，还看 DB 的 entropy 输入，以这两项共同驱动 Agent action 进行优化。

这里我们做了特别多的工作，信息槽从 1 个到 5 个，数据库的规模从大到小，都做了特别多的尝试，这样在和用户交互的时候，agent 可以自主的查询检索，甚至可以填充和修改数据库。



这是我们 2017 发布的一个工作，KB driven-dialogue，其优点：

- ❖ 控制万能高频回复（ 提高答应包含的有用信息 ）
- ❖ 赋予 agent 对话主动性

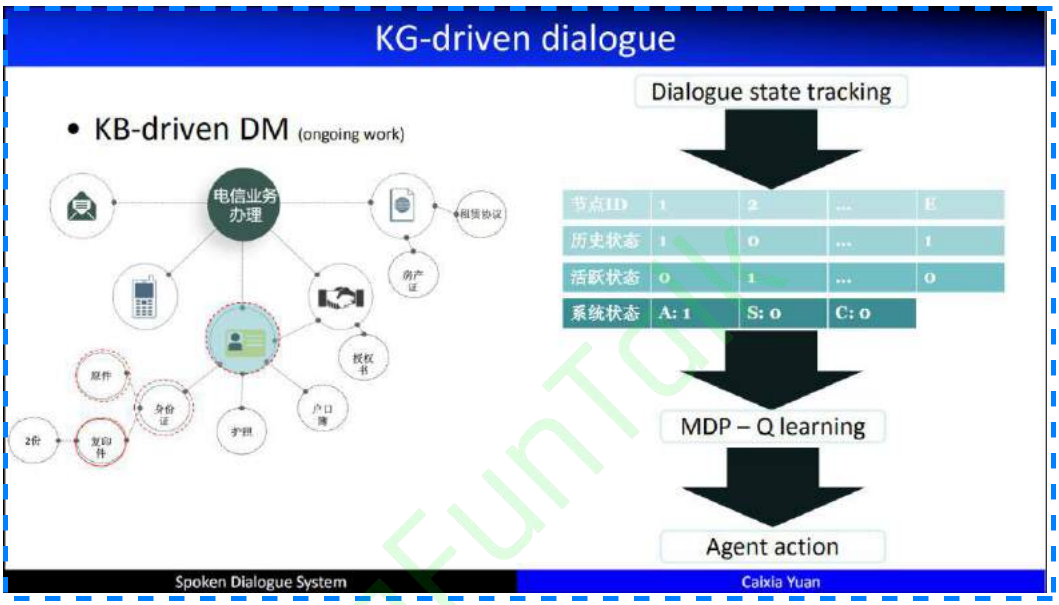
5. KG-driven dialogue

刚刚讲的基于 KB 的 dialogue 任务，基本都认为对话任务就是在进行槽填充的任务，如果一个 agent 是主动性的，通过不停的和用户进行交互来采集

槽信息，所以叫槽填充，当槽填完了，就相当于对话任务成功了。但是，当我们在定义槽的时候，我们认为槽是互相独立的，并且是扁平的。然而，实际中许多任务的槽之间存在相关性，有的是上下位关系，有的是约束关系，有的是递进关系等等。这样自然的就引出了知识图谱，知识图谱可以较好地描述上述的相关性。于是，产生了两个新问题：

- 知识图谱驱动的对话理解：实体链接
- 知识图谱驱动的对话管理：图路径规划

这里主要讲下第二个问题。

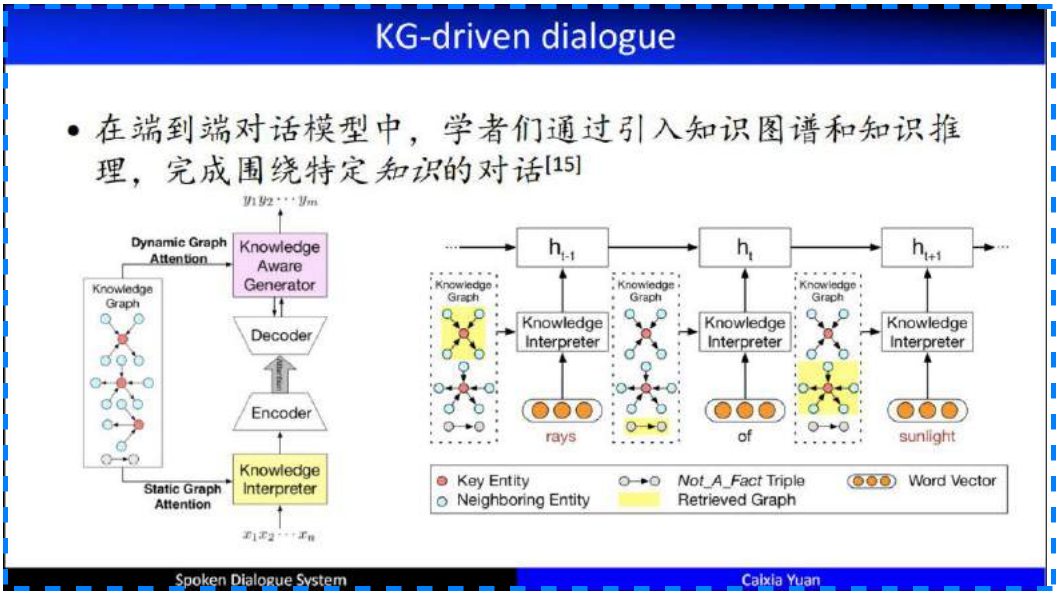


举个例子，我们在办理电信业务，开通一个家庭宽带，需要提供相关的证件，是自己去办，还是委托人去办，是房东还是租户等等，需要提供各种不同的

材料。于是这个情景就产生了条件的约束，某一个 node 和其它 node 是上下位的关系，比如证件可以是身份证，也可以是护照或者户口簿等等。所以我们可以通过知识图谱来进行处理。

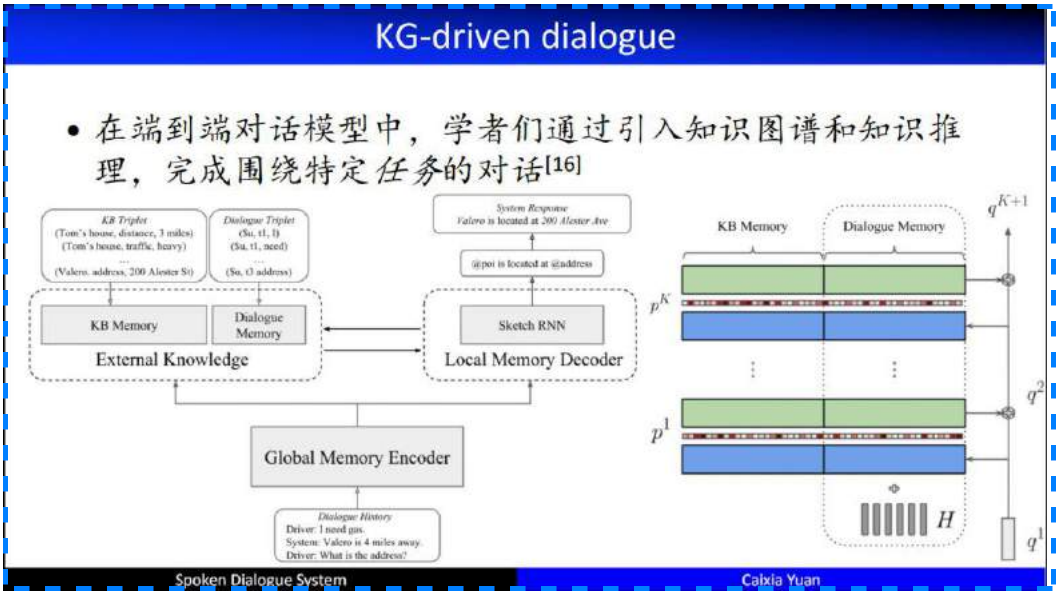
当一个用户的对话过来，首先会链接到不同的 node，再基于 node 和对话历史构造一个对话的 state，我们会维持一个全局的 state 和一个活跃的 state，同时活跃的 state 会定义三种不同的操作动作，一个是祖先节点，一个是兄弟节点，还有一个是孩子节点。所以，在这样的知识图谱上如何寻优，比如当通过某种计算得到，它应该在某个节点上进行交互的时候，我们就应该输出一个 action，这个 action 要和用户确认他是一个租户，还是自有住房等等。所以，这个 action 是有区别于此前所提到的在特定的、扁平的 slot 槽上和用户进行信息的确认、修改等还是有很大不同的。解决这样的问题，一个非常巨大的挑战就是状态空间非常大。比如图中的节点大概有 120 个，每个节点有 3 个不同的状态，知识从节点的状态来看就有 3^{120} 种可能。这也是我们正在开展的待解决的一个问题。

DataFunTalk



在端到端的对话模型（ 闲聊 ）中，也开始逐步地引入知识图谱。下面介绍两个比较具有代表性的引入知识图谱后的人机对话。其中右边是 2018 年 IJCAI 的杰出论文，清华大学黄民烈老师团队的工作，引入了通过 KG 来表示的 Commonsense，同时到底在编码器端还是在解码器端引入知识，以及如何排序，排序的时候如何结合对话的 history 做知识的推理等等，都做了特别全面的研究。

DataFunTalk



另一个比较有代表性的工作是在 ICLR2019 提出的，在架构中引入了 Local Memory 和 Global Memory 相融合的技术，通过这种融合，在编码器端和解码器端同时加入了知识的推理。

总结下 KB/KG-driven dialogue：

优点：

已经有大规模公开的数据（e.g.，InCar Assistant，MMD，M2M）。

训练过程可控&稳定，因为这里多数都是有监督学习。

缺点：

因为采用有监督的方式进行训练，所以存在如下问题：

① 环境确定性假设 ② 缺少对动作的建模 ③ 缺少全局的动作规划

Agent 被动，完全依赖于训练数据，所以模型是不赋予 Agent 主动性的。

构建 KB 和 KG 成本昂贵！

6. Document-driven dialogue

DataFunTalk

Document 驱动的对话，具有如下优点：

① 应用场景真实丰富：

情景对话（conversation）,比如针对某个热门事件在微博会产生很多对话，这就是一个情景式的对话。

电信业务办理（service），比如 10086 有非常多的套餐，如何从中选择一个用户心仪的套餐？每个套餐都有说明书，我们可以围绕套餐的说明书和用户进行交互，如 "您希望流量多、还是语音多"，如果用户回答 "流量多"，就可以基于文本知道给用户推荐流量多的套餐，如果有三个候选，机器就可以基于这三个候选和用户继续进行交互，缩小候选套餐范围，直到用户选出心仪的套餐。

电商产品推荐（recommendation），可以根据商品的描述，进行各种各样的对话。这里的输入不是一个 dialogue，也不是一个 KB，甚至结构化的内容非常少，完全是一个 free document，如何基于这些 document 进行推荐，是一个很好的应用场景。

交互式信息查询（retrieval），很多时候，一次查询的结果可能不能用户的需求，如何基于非结构化的查询结果和用户进行交互，来更好地达成用户的查询目的。

……

② 数据获取直接便捷：

相比于 dialogue、FAQ、KB、KQ 等，Document 充斥着互联网的各种各样的文本，都可以看成是文本的数据，获取方便，成本非常低。

③ 领域移植性强：

基于文本，不再基于专家定义的 slot，也不再基于受限的 KB/KG，从技术上讲，所构造的 model 本身是和领域无关的，所以它赋予了 model 很强的领域移植性。

DataFunTalk

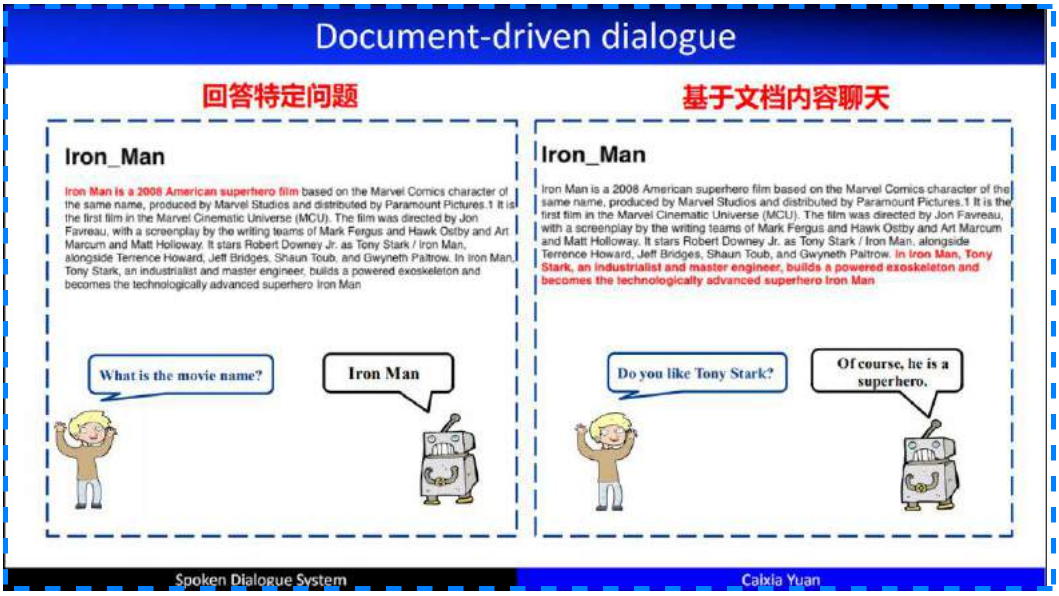
这是我们正在进行的工作，情景对话偏向于闲聊，没有一个用户目标。这里需要解决的问题有两个：

- 如何引入文档：编码端引入文档、解码端引入文档
- 如何编码文档：文档过长、冗余信息过多

数据：

我们在 DoG 数据的基础上模拟了一些数据，形成了如上图所示的数据集，分 Blind 和 Non-blind 两种情形构造了不同的数据集。

DataFunTalk



我们发现，基于文本的端到端的聊天，有些是基于内容的闲聊，有些还需要回答特定的问题。所以，评价的时候，可以直接用 F1 评价回答特定问题，用闲聊通用的评价来评价基于内容的聊天。

刚刚讲的是偏闲聊式的对话，接下来讲下任务型对话。

这里的动机分为两种情况：单文档和多文档，我们目前在挑战多文档的情况，单文档则采用简单的多轮阅读理解来做。

多文档要解决的问题：

如何定义对话动作：因为是基于 Document 进行交互，而不再是基于 slot 进行交互，所以需要重新定义对话动作。

如何建模文档差异：以刚刚 10086 的例子为例，总共有 10 个业务，通过一轮对话，挑选出 3 个，这 3 个业务每个业务可能都有 10 种属性，那么其中一些属性值相同的属性，没必要再和用户进行交互，只需要交互它们之间不同的点，所以交互的角度需要随对话进行动态地变化。

数据：

这里采用的数据是 WikiMovies 和模拟数据，具体见上图。

7. A very simple sketch of dialogue

DataFunTalk

上图为任务型对话和非任务型对话的几个重要节点，大家可以了解下。

03

Concluding remarks

- 任务型对话具有最丰富的落地场景。
- 纯闲聊型对话系统的学术价值尚不清楚。
- 任务型和非任务型边界愈加模糊，一个典型的人人对话既包括闲聊，又包括信息获取、推荐、服务。
- 引入外部知识十分必要，外部知识形态各异，建模方法也因而千差万别。

DataFunTalk

- Uncertainty 和 few-shot 问题，是几乎所有的对话系统最大的 "卡脖子" 问题。

- X-driven dialogue 中的 X 还有哪些可能？刚刚提到的 X 都还是基于文本的。事实上，从 2005 年开始，我们已经开始做 image 和文本数据融合的对话；从 2013 年开始做 Visual QA/Dialogue，挑战了 GuessWhat 和 GuessWhich 任务。

- X=multi media（MM Dialogue），X 还可以很宽泛的就是指多媒体，不光有 image 还可能有 text，2018 年已经有了相关的任务，并且开源了非常多的电商业务。这是一个非常有挑战性的工作，也使人机对话本身更加拟人化。

04

Reference

这是部分的参考文献，有些是我们自己的工作，有些是特别杰出的别人的工作。

DataFunTalk

Reference

- [1] Victor Zue, et al. JUPITER: A Telephone-Based Conversational Interface for Weather Information. IEEE Transactions on Speech and Audio Processing (Volume: 8 , Issue: 1 , Jan 2000)
- [2] Steve Young, Milica Gasić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. Proceedings of the IEEE, 101(5):1160–1179, 2013.
- [3] Budzianowski, Paweł, Wen T H , Tseng B H , et al. MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. EMNLP 2018.
- [4] Lowe, Ryan, et al. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. arXiv preprint arXiv:1506.08909, 2015.
- [5] Mulan Hou, Xiaojie Wang, Caixia Yuan, et al. Attention Based Joint Model with Negative Sampling for New Slot Values Recognition. International Workshop on Spoken Dialogue Systems Technology (IWSDS 2018), Singapore.
- [6] 宋博川. 跨领域对话理解技术研究与实现. 北京邮电大学硕士论文, 2017年6月.
- [7] Shuyu Lei, Xiaojie Wang, Caixia Yuan. Word-Based POMDP Dialog Management via Hybrid Learning. IEEE Access 7: 39236-39243(2019)
- [8] Bansal, Trapit, et al. Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748 (2017).
- [9] Hao Fu, Caixia Yuan, Xiaojie Wang, et al. Zero-Shot Question Classification Using Synthetic Samples. In 5th IEEE CCIS, 2018
- [10] Qi Mo, Xiaojie Wang. Combining Text Classification and Text Matching for FAQ- based Question Answering. Journal of Beijing University of Posts and Telecommunications, 42(4), 2018.

Reference

- [11] Guohua Wu, Caixia Yuan, Xiaojie Wang. Finite-to-Infinite N-best POMDP for Spoken Dialogue Management. In CCL 2015.
- [12] Bhuwan Dhingra et al. Towards End-to-End Reinforcement Learning of Dialogue Agents for Information Access. In ACL 2017.
- [13] 商甜甜. 融合数据库查询的对话管理模型研究与应用. 北京邮电大学硕士论文, 2018年5月.
- [14] Guiyuan Gu, Caixia Yuan, Xiaojie Wang. Non-Task Dialogue System Enriched with Knowledge. In Proceedings of 5th IEEE CCIS 2018.
- [15] Wu C S, Socher R, Xiong C. Global-to-local Memory Pointer Networks for Task-Oriented Dialogue. ICLR 2019.
- [16] ZhouH,YoungT,HuangM,etal.CommonsenseKnowledgeAwareConversationGeneration with Graph Attention. IJCAI. 2018: 4623-4629.
- [17] Zhou K , Prabhumoye S , Black A W . A Dataset for Document Grounded Conversations. arxiv. 2018.
- [18] MCTEAR M F. Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit[C] //Proc 5th International Conference on Spoken Language Processing. 1998: 1223–1226.
- [19] Ji Z , Lu Z , Li H . An Information Retrieval Approach to Short Text Conversation[J]. Computer Science, 2014.
- [20] Yang X , Chen Y N , Hakkani-Tur D , et al. End-to-End Joint Learning of Natural Language Understanding and Dialogue Manager[J]. 2016.

DataFunTalk

今天的分享就到这里，感谢大家的聆听，也感谢我们的团队。

DataFunTalk

腾讯信息流内容理解技术实践

嘉宾：郭伟东　编辑：李传勇

导读：目前信息流推荐中使用的内容理解技术，主要有两部分构成：1. 门户时代和搜索时代遗留的技术积累 :分类、关键词以及知识图谱相关技术 ;2. 深度学习带来的技术福利：embedding。但是分类对于兴趣点刻画太粗，实体又容易引起推荐多样性问题，而 embedding 技术又面临难以解释的问题。这次主要介绍在信息流推荐中，腾讯是如何做内容理解克服上述问题的。主要包括：

- 项目背景
- 兴趣图谱
- 内容理解
- 线上效果

项目背景

1. 内容理解技术演进

DataFunTalk

① 门户时代：1995~2002 年，主要代表公司：Yahoo、网易、搜狐、腾讯。互联网初期，因为数据较少，因此需要一个内容聚合的地方，人们才能够快速的找到信息。因此，门户通过 **"内容类型"** 对内容进行整理，然后以频道页形式满足用户需求。因为数据少，初期由人工对新闻进行分类。随着数据的增多，靠人工分类已经变得不现实，因此各大公司纷纷引入分类技术，自动化文本分类。此后，文本分类技术发展迅速。

② 搜索/社交时代：2003 年~至今，主要代表公司：搜狗、腾讯、Google、百度。随着网络的普及，数据的数量和类型的丰富，门户网站已经不能够承载信息分发的任务。于是，一种新的信息分发技术诞生——搜索。搜索除了需要分类信息以外，还需要精确知道文章是 **"关于什么的"**，关键词技术很好的解决了这个需求，于是也成为那个时期的研究热点。但是关键词技术有一个问题没办法克服：实体歧义问题（ 如李白，究竟用户是找诗人李白，还是

王者荣耀英雄李白)。2012 年 Google 提出知识图谱概念，可以用于解决上述的实体歧义问题，实体链指的问题也有了比较大的进展。

③ 智能时代：2012 年~至今，主要代表公司：今日头条、出门问问等。使用 2012 年来作为智能时代的开始，主要是这一年头条成立。头条定义了一种新的信息分发形式——个性化推荐。虽然个性化推荐技术早有研究，但是对于信息分发这个任务有不可或缺的推动作用。

但是在信息推荐中，我们仍然在使用分类、关键词和实体等传统的内容理解方法，那到底在智能时代下是否需要新的内容理解方案呢？

2. 推荐和搜索的区别

推荐和搜索非常相似，都是根据已有的输入，返回跟输入相关的文章，但是对于内容理解的要求区别较大，下面仔细分析下原因：

搜索是给定一个 query 后，预测 doc 被点击的概率进行排序。大致的处理流程如下：首先对 query 分词，得到 < term, weight> 的一个列表（去除停用词等不重要的词），然后根据每一个 term 拉倒排索引 document list 做召回，再对召回的所有文章取并集，最后做整体的排序。注意：这里排序的条件是所有 **term 的交集**（条件概率标红部分）。

推荐是给定一个 user 后，预测 doc 被点击的概率进行排序。大致的处理流程如下：首先查询 user 的用户画像，得到 < term, weight > 的一个兴趣点列表，然后根据每一个 term 拉倒排索引 document list 做召回，再对召回的所有文章取并集，最后做整体的排序。注意：这里排序的条件跟搜索是不同的，排序的条件是 **term 的并集**（条件概率标红部分）。例如用户阅读了王宝强马蓉离婚的新闻，会把 "王宝强"、"马蓉" 作为两个兴趣点积累到用户画像中，而对新的文章排序时候，实际上已经丢失了 "王宝强" 和 "马蓉" 兴趣点是同一篇文章同时积累的这个信息。

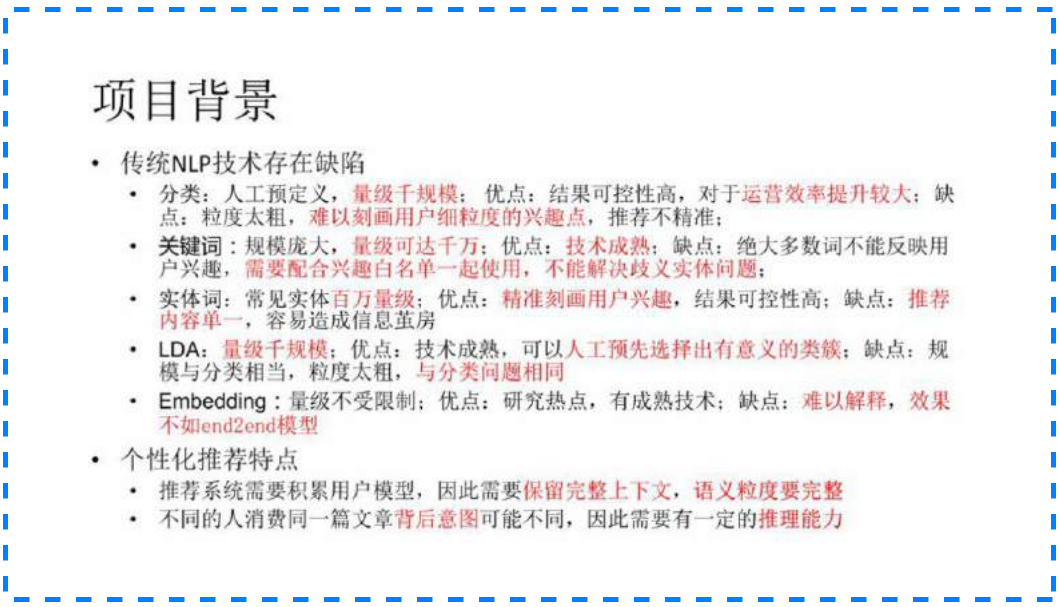
通过上述分析，我们可以得到这样的结论：搜索经过召回之后，排序有完整的上下文信息；但是在推荐中由于经过了用户画像，使用传统的内容理解方案时，排序会丢失用户阅读的上下文信息。因此，**推荐对于内容理解需要保**

留完整的上下文，即把 "王宝强马蓉离婚" 当做一个完整的兴趣点，而不仅仅像搜索一样分别保留 "王宝强" 和 "马蓉"。

3. 用户为什么会消费

传统的内容理解仅仅解决了文章是什么的问题，但是对于用户 "**为什么会消费**" 却没有考虑。举个例子来说，如果我们的一个朋友看了一款 XC60 的汽车，我们会给他推荐什么内容呢？我们一定不会不停的推荐 XC60，而是会猜测朋友可能是喜欢沃尔沃这个品牌，比较看重安全性，或者对于空间有一定要求的 SUV。因此，如果要做好一个推荐系统，内容理解同样也需要相应的能力，挖掘用户真实消费意图。

4. 传统的 NLP 的技术缺陷



以上是整个项目的背景，我们总结一下。传统 NLP 技术存在缺陷：

- 分类：人工预定义，量级千规模；优点：结果可控性高，人工可以参与运营；缺点：粒度太粗，难以刻画用户粒度的兴趣点，推荐不精准；
- 关键词：规模庞大，量级可达千万；优点：技术成熟；缺点：绝大多数词不能反映用户兴趣，需要配合兴趣白名单一起使用，不能解决歧义的问题；
- 实体词：常见实体百万量级；优点：精准刻画用户兴趣，结果可控性高；缺点：推荐内容单一，容易造成信息茧房；
- LDA：量级千规模，优点：技术成熟，可以人工预先选择出有意义的类簇；缺点：规模和分类相当，粒度太粗，与分类问题相同；

- Embedding：量级不受限制；优点：研究热点，有成熟技术；缺点：难以解释。

个性化推荐需求：

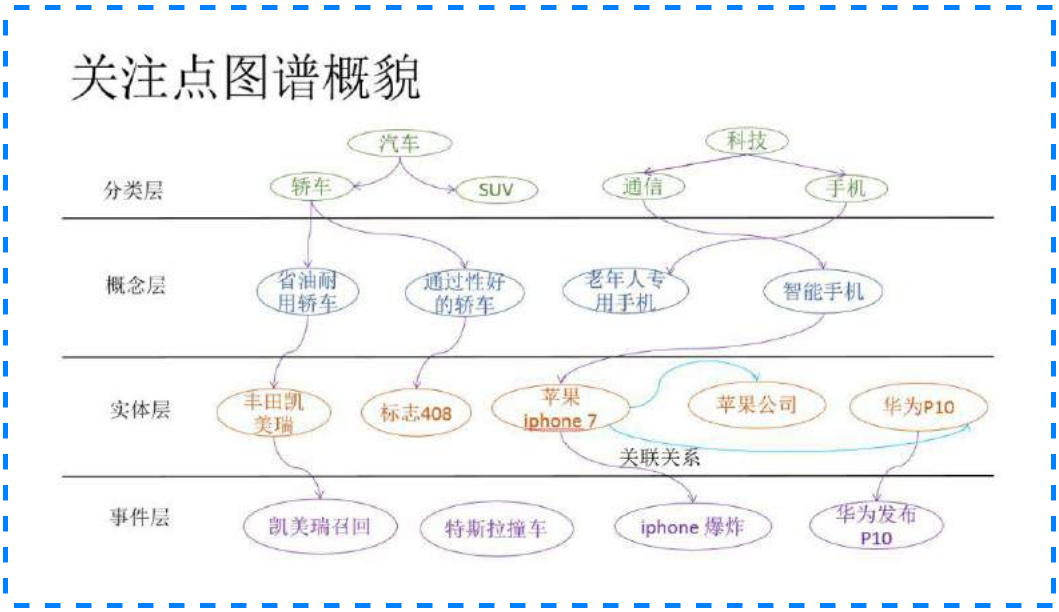
- 推荐系统需要积累用户模型 ,因此需要保留完整的上下文 ,语义粒度要完整；
- 不同的人消费同一篇文章背后原因可能不同，因此需要有一定的推理能力。

因此，传统的内容理解方案并不能很好的满足个性化推荐的需求。个性化推荐不仅需要传统的内容理解方式，还需要一种能够有完整上下文，并且具有推理用户真实消费意图的能力。

兴趣图谱

1. 兴趣点图谱

DataFunTalk



根据上述分析，我们提出了兴趣点图谱，用于解决上述个性化推荐中遇到的问题。兴趣点图谱由四层组成：分别为：分类层、概念层、实体词和事件层。下面分别介绍各层的内容：

分类层：一般是由 PM 建设，是一个严格树状的结构，一般在 1000 左右个节点；

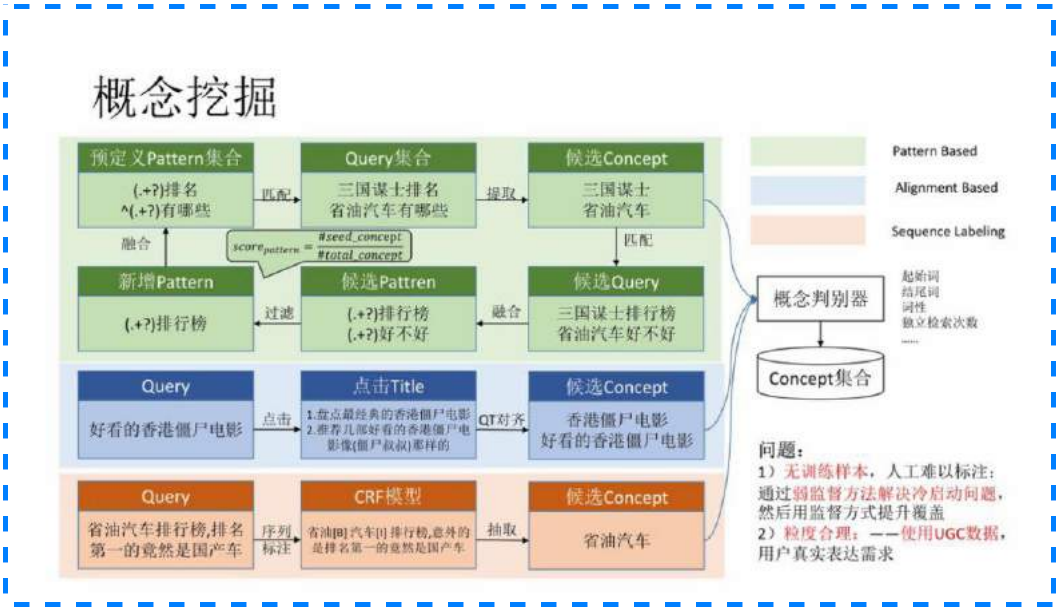
概念层：有相同属性的一类实体称之为概念，例如老年人专用手机、省油耐用车等；

实体层：知识图谱中的实体，如：刘德华，华为 P10 等；

事件层：用来刻画某一个事件，例如：王宝强离婚、三星手机爆炸等。

分类层主要解决人工运营的需求；概念层推理用户消费的真实意图；实体层负责一般兴趣点的召回；事件层精准刻画文章内容。下面介绍如何构造兴趣点图谱。

2. 概念挖掘



概念本质是一种短语，其实短语挖掘的论文非常多，像韩家炜老师团队就有很多相关的论文，但是概念有自己的独特性：

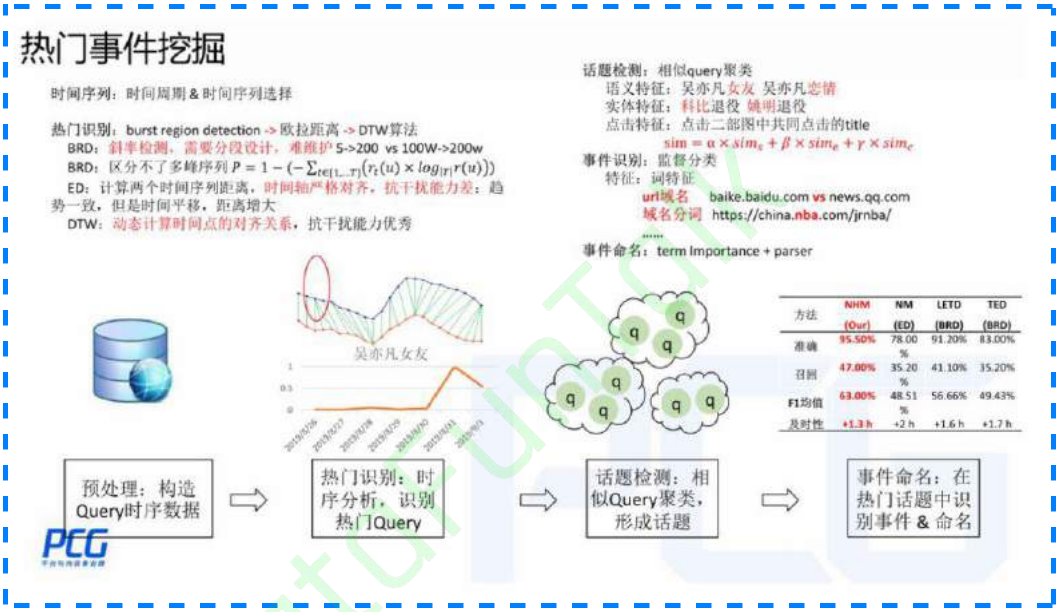
① **没有训练样本，并且人工难以标注。**因此只能通过弱监督方法解决冷启动的问题，然后使用监督方式提升覆盖。

② **粒度问题。**比如 "明星" 是一个概念，但是太泛，不能精准刻画用户兴趣，但是 "身材好的女明星" 就很合理，那如何描述粒度呢？使用 UGC 数据，用户真实表达需求。

因此，具体挖掘时，我们使用了搜索数据，通过用户的点击行为进行半监督算法的学习。具体算法如上图所示：

挖掘概念使用的是搜索数据，每一个概念都有多个点击的网页，对网页进行实体抽取，然后统计实体和概念的共现频次就可以获得较为准确的上下位关系，我们在 KDD 的 paper 中有详细的介绍，这里就不再重复。

3. 热门事件挖掘



事件指的是热门事件。如果一个事件比较热门，网友就会有了解需求，会通过搜索引擎来查询事件，因此我们使用 query 作为热门事件挖掘的来源。

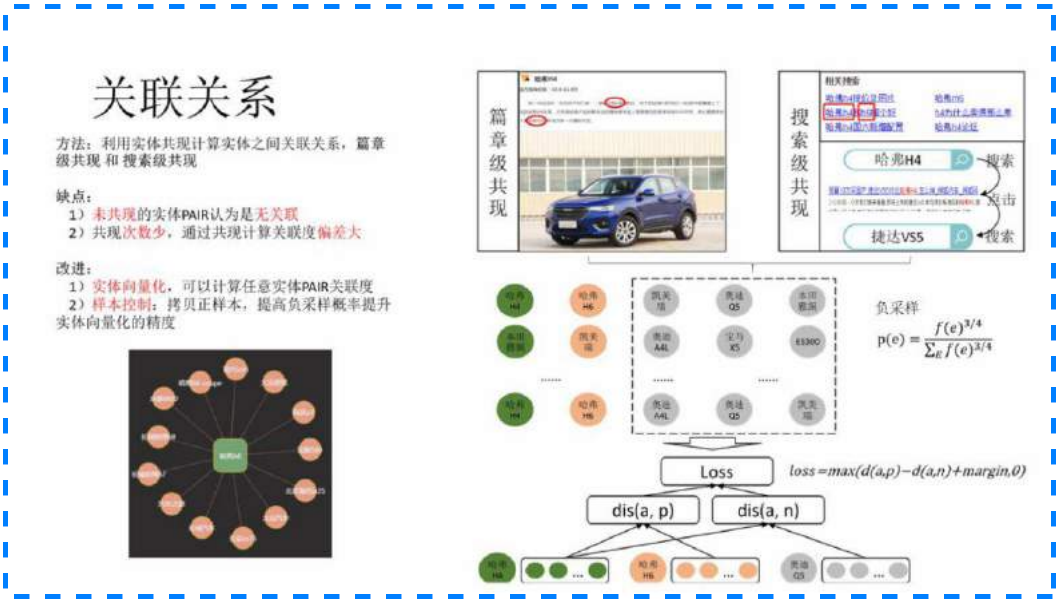
一个比较常见的方法是根据事件搜索量变化趋势判断，常规的做法是 BRD（Burst Region Detection），判断时间序列上是否有爆发点。但是 BRD 会遇到一些归一化，甚至多 point 的问题，于是我们采用了上图的方式克服上述问题：

热门识别：时序分析，识别热门 query。首先定义一个热门事件的趋势模板；然后对第一步预处理后的时序数据与热门模板进行相似度计算，如果相似度很高，说明趋势一致，则为热门事件，否则就是非热门。相似度计算的方式最早用的距离是欧拉距离，但是由于欧拉距离需要严格的时序对齐，会造成一些 bad case，因此改用 DTW 算法。

话题检测：同一个事件会有多种表述方法，对应多个 query，因此需要把相同事件的 query 聚类到一起，形成话题。

事件识别&命名：热门的话题中往往会伴随一些非事件型的话题，如热门美剧更新时，会出现一个热度高潮，上述方法会混入一些非事件，因此我们需要对热门的话题做一个分类。一个非常有效的特征是 url 中的一些单词，会很有区分性。

4. 关联关系



个性化推荐系统中除了要满足用户当前兴趣以外，还需要探索到用户未知的潜在兴趣点，扩展用户阅读视野。因此需要对节点计算关联关系。目前我们仅针对实体做了关联关系的计算。

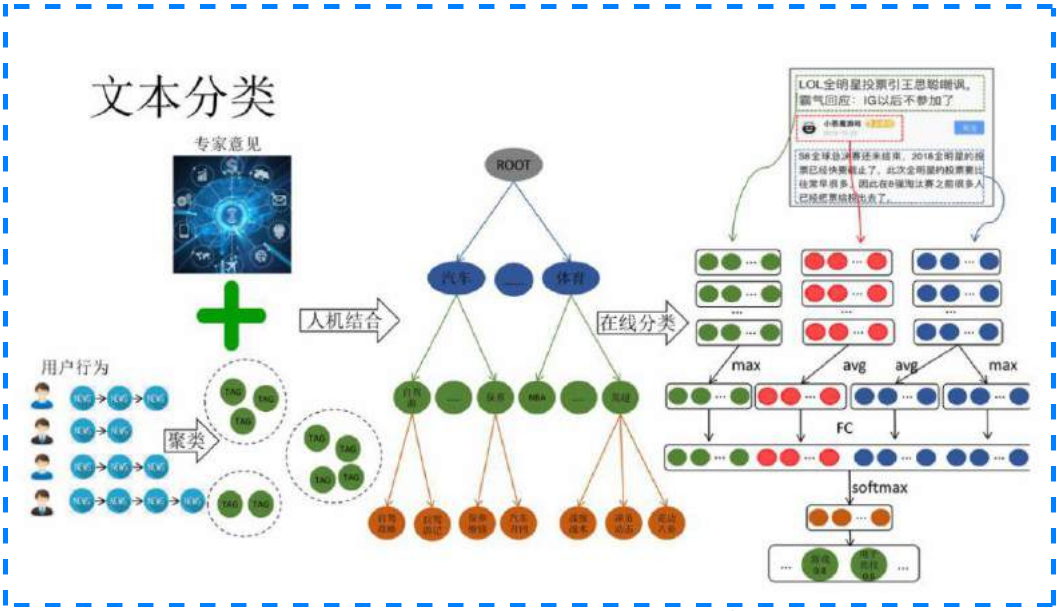
大家很容易想到，如果两个实体经常会在同一篇文档中出现，应该就是高关联的；或者用户经常连续搜索，即搜完 "刘德华"，然后会马上搜索 "朱丽倩"，应该也是高关联的。确实这种直觉是正确的。虽然这种方法准确率很高，但是会遇到一些问题：没有共现过的，会被认为没有任何的关系；对于共现少的 pair 对，关系的密切度计算误差也会比较大。

因此，需要通过实体向量化的形式克服上述问题。上述的共现数据可以作为正例，负样本采用同类实体随机负采样，正负样本比例 1:3，通过 pair wise

的 loss 进行训练，得到每个实体的 embedding，然后计算任意两个实体的关联度。

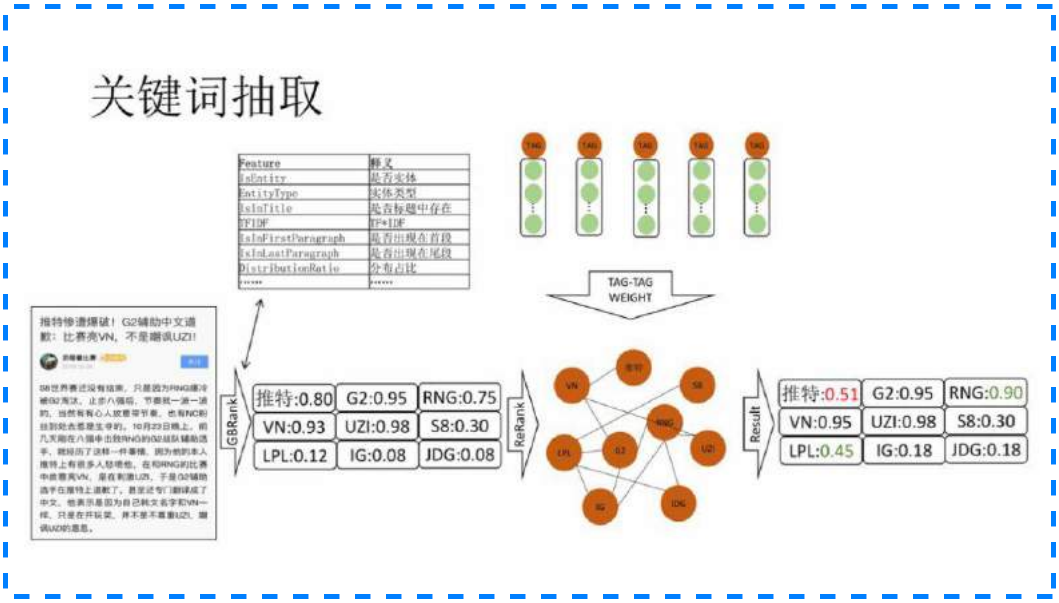
内容理解

1. 文本分类



主题分类层是 PM 整理的，但是 PM 整理的过程中可能会存在一些认知偏差。可以使用用户的点击行为对内容进行聚类，聚完类之后让 PM 去标注，从而总结出一些更适合的类别用于描述用户的兴趣。

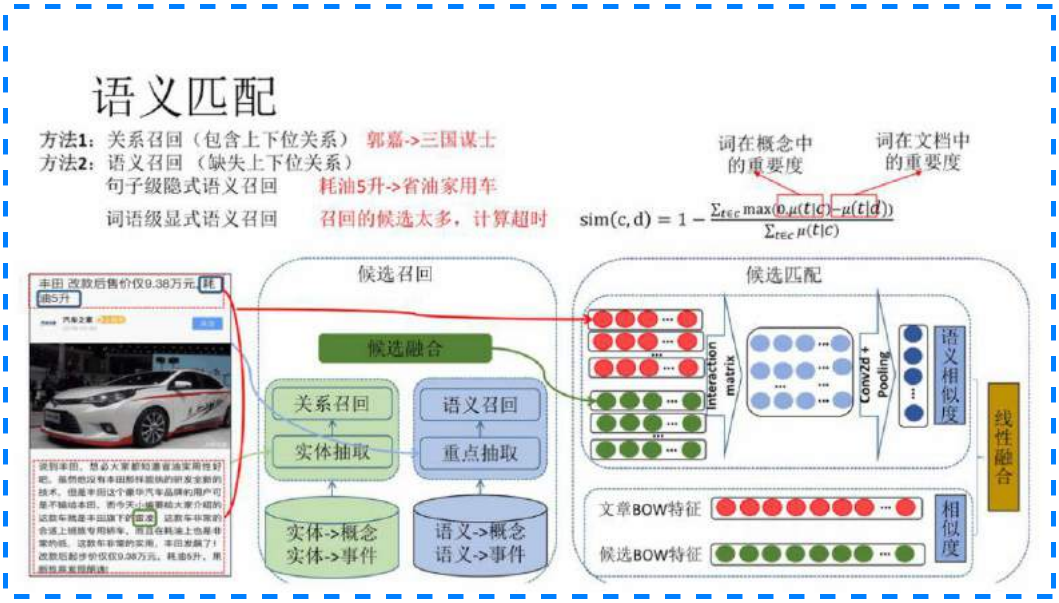
2. 关键词抽取



我们使用了比较传统的关键词提取思路，利用传统特征工程 + GBRank 算法排序。在实际中会遇到这样的问题，如示例，Twitter 出现在 title 中的实体，传统的方式会把 Twitter 分数计算的很高，但是这篇文章中却不是重点，重点是两支 LOL 战队的骂战。于是我们在 BGRank 之后，加了 re-rank 层，为所有的候选词做一个重排序。词之间边关系使用关联关系 embedding 计算相似度得到。

3. 语义匹配

DataFunTalk



对于概念和事件类型的标签，原文中可能并不会出现，用抽取的方式就没有办法解决。我们采用召回+排序的方式解决。召回的逻辑分为关系召回和语义召回，其中关系召回会用到兴趣点图谱中的关系数据，召回 1-hop 内的节点作为候选，语义召回通过语义向量召回与 title 近邻的节点作为候选，然后用交互匹配的方式进行排序。如果 1-hop 内的节点数量太多，排序耗时会非常大，因此这里采用粗糙集的方式进行候选的粗排，缩小候选集合再进行排序。

线上效果

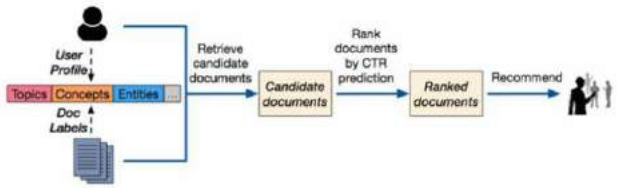
DataFunTalk

线上效果

- **Impression Page View (IPV)**: number of pages that matched with users.
- **Impression User View (IUV)**: number of users who has matched pages.
- **Click Page View (CPV)**: number of pages that users clicked.
- **Click User View (CUV)**: number of users who clicked pages.
- **User Conversion Rate (UCR)**: $\frac{CUV}{IUV}$.
- **Average User Consumption (AUC)**: $\frac{CPV}{CUV}$.
- **Users Duration (UD)**: average time users spend on a page.
- **Impression Efficiency (IE)**: $\frac{CPV}{IUV}$.

Table 4: Online A/B testing results.

Metrics	Percentage Lift	Metrics	Percentage Lift
IPV	+0.69%	UCR	+0.04%
IUV	+0.06%	AUC	+0.21%
CPV	+0.38%	UD	+0.83%
CUV	+0.16%	IE	+6.01%



实验部分，baseline 是仅用传统的实体和分类标签，而实验组除了实体和分类以外，同时使用概念和事件类型的兴趣点，最后线上效果提升明显。

今天的分享就到这里，谢谢大家。

DataFunTalk

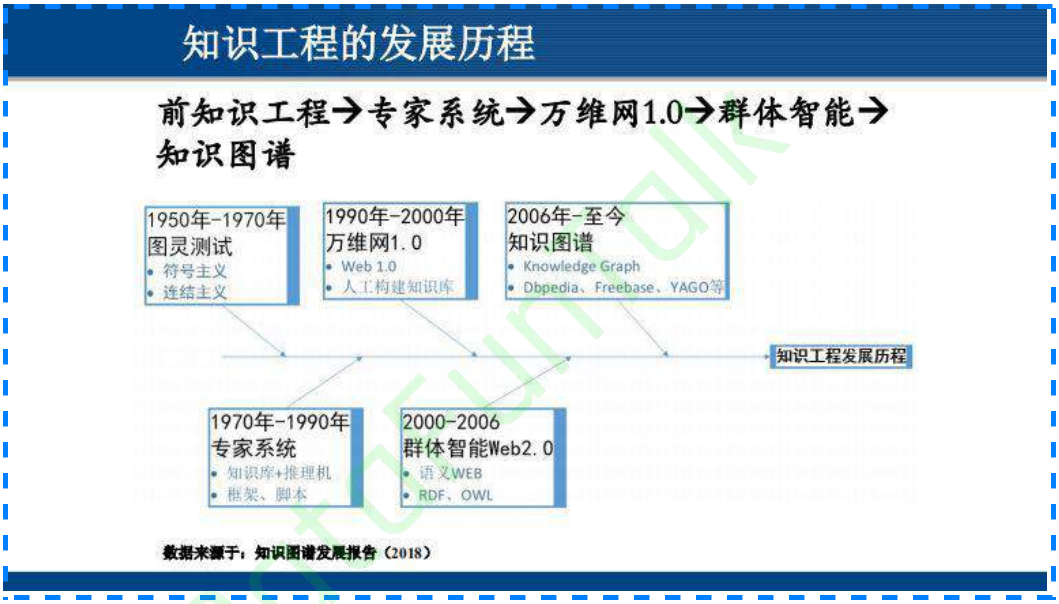
知识图谱的自动构建

嘉宾：吴信东　编辑：王吉东

导读：知识图谱的构建包括逻辑建模、隐含空间分析、人机交互和本体模型支撑等多种方法。我们将分析各种构建方法的问题和挑战，指出自动构建的要素和应用场景。

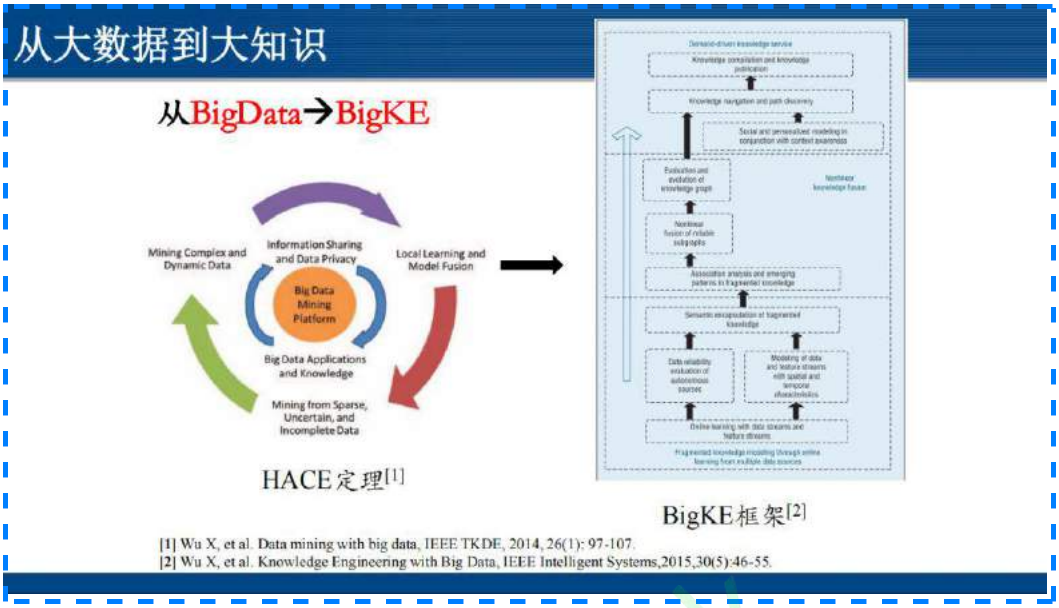
——背景——

知识图谱是明略科技的核心技术。知识图谱的自动构建和数据挖掘有一定的关联，自动构建知识图谱和手动构建不是一个概念。明略科技的新产品正在做到：专家在台上讲话，后台图谱系统可自动同步构建知识图谱。



提到知识工程的发展，首先是 80 年代蓬勃发展的专家系统，随后逐步发展至 90 年代的万维网知识库。90 年代的万维网 1.0，以信息收集为主；后面加入人工因素，即人和信息一起，此时进入万维网 2.0 阶段。

"知识图谱"这一概念是 2006 年由谷歌提出，谷歌出于搜索引擎需要而提出这样的名词，其技术核心类似于 60 年代提出的语义网络。



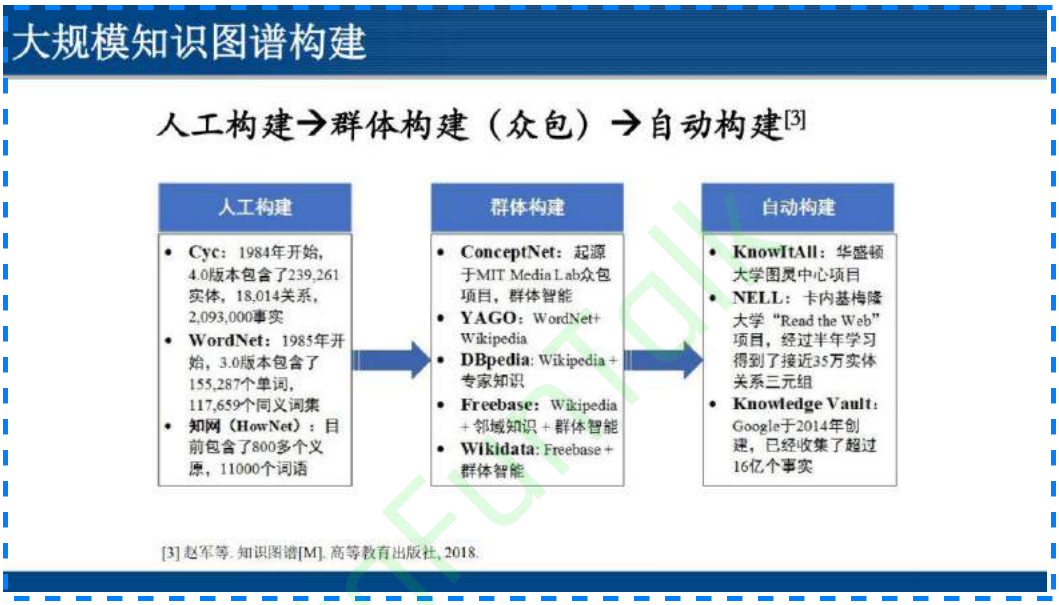
吴信东教授于 2014 提出 "HACE 定理"，指出大数据始于异构（Heterogeneous）、自治（Autonomous）的多源海量数据，旨在寻求探索复杂（Complex）、演化（Evolving）的数据关联和方法，这是对大数据本质特征的提炼。

大数据在实现的过程中分为 3 层结构 :最底层是数据平台（Big Data Mining Platform），做数据的收集、整合、加工等；中间的一层是应用领域，涉及到

语义、专业领域知识等方面；最外层是大数据分析实现的算法，涉及到机器学习、数据挖掘等。

从大数据（BigData）到大知识（BigKE），体现的是基于数据的知识提炼过程。"大"知识的特点除了量大，更主要的是"质量没有保证"；我们的目标是，在浩瀚无边的知识海洋中，如何针对当前问题找到相关的知识进行问题求解，实现"量 -> 质 -> 序"的过渡。

基于大数据的特点，实现大数据到大知识的跳跃，这一过程中经历了信息检索的不断完善和分析；从知识图谱构建的角度来看，这一过程经历了人工构建-群体构建-自动构建这样的技术路线。详见下图。



本文将着重讲解"自动构建"。自动构建的过程中，如果数据是结构化的（例如图表数据），已知属性名称、属性间的层次结构等，构建知识图谱相对较为

容易；如果缺乏以上信息，则只能通过文本信息等非结构化数据中提炼知识构建知识图谱，技术上将面临很多挑战。

结构化数据通常具有良好的布局结构，因此识别和抽取比较容易，可针对特定格式编写模板进行抽取，抽取准确率也比较高。早在知识图谱技术大受追捧之前的上个世纪 90 年代，国内便开始了 "从关系型描述数据库生成语义网络的方法" 研究工作。

非结构化数据上的知识图谱研究，主要集中非结构化文本数据处理上。由于自然语言表达的多样性、灵活性，实体和关系在文本中一般找不到明确的标识，这使得从中抽取实体和识别语义关系非常困难。

下面以一个实例来描述非结构化数据知识图谱的构建过程。文本数据来源于百度百科，介绍秦始皇的生平事迹。原文如下：

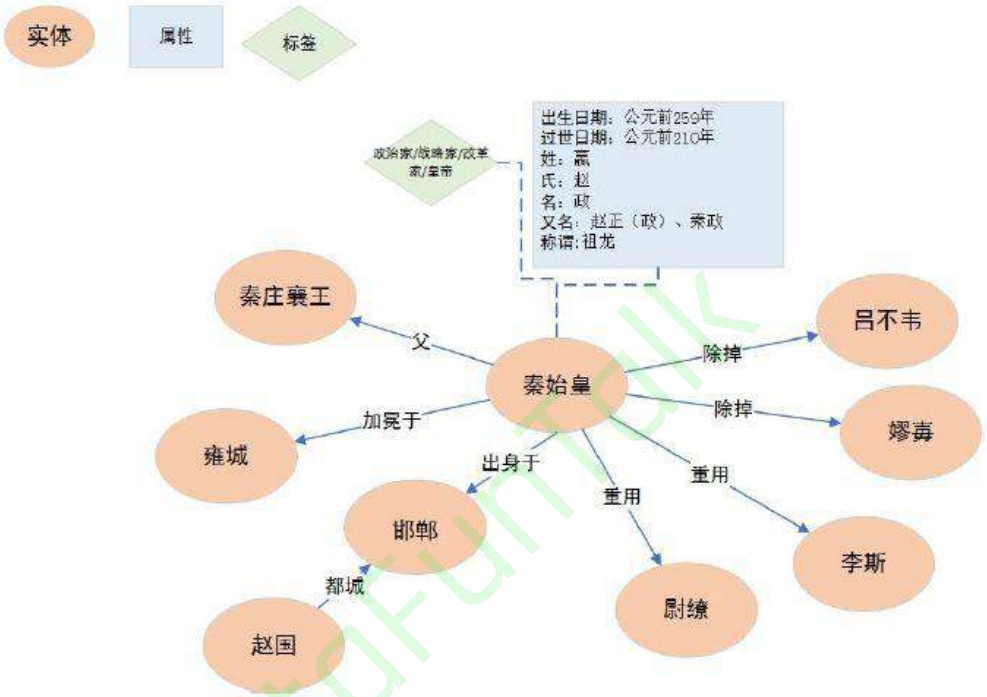
秦始皇

秦始皇（前259年—前210年），嬴姓，赵氏，名政，又名赵正（政）、秦政，或称祖龙，秦庄襄王之子。中国历史上著名的政治家、战略家、改革家，完成华夏大一统的铁腕政治人物，也是中国第一个称皇帝的君主。

秦始皇是出生于赵国都城邯郸（今邯郸），并在此度过了少年时期。前247年，13岁时即王位。前238年，22岁时，在故都雍城举行了国君成人加冕仪式，开始“亲理朝政”，除掉吕不韦、嫪毐等人，重用李斯、尉缭，……。

数据来源于:百度百科

基于以上文本，初步构建知识图谱如下：



上述图谱抽取的信息不是十分完全，但是大体上能够涵盖和秦始皇相关的各种各样的人物以及各种各样的关系。

——构建方法——

这一部分会主要介绍现有的常见的知识图谱的构建方法。

知识图谱的构建方法，主要包含 4 大类：逻辑建模，隐含空间分析，人机交互，本体模型。

1. 逻辑建模

上一部分提到的 "秦始皇" 的实例，就是根据逻辑建模提炼生成的。

将名词和关系进行抽取，如果涉及到事件，将会涉及到条件概率、先验概率等。

在逻辑建模中，逻辑+概率作为可能世界的概率度量；对逻辑进行概率化，并利用知识库中的每一条关系三元组对可能世界概率进行约束。

逻辑建模中会涉及到逻辑变量和规则推理等方面的模型，其代表模型是马尔可夫逻辑网模型。将马尔可夫逻辑网看作一个构造马尔可夫网的模板，它维护一个基于一阶逻辑的规则库，并对每一个逻辑规则附上了权重，以此对可能的世界进行软约束。其概率模型为：

$$P_w(X=x)=\frac{1}{Z}\exp\left[\sum_{f_i\in F} w_i \sum_{g\in G_{f_i}} g(x)\right]$$

其中，g(x)=1 表示该实例化的规则为真，反之为假；F 为 Markov 网中所有谓词规则的集合，Gf_i 是利用所有原子事实去实例化规则 f_i 后的集合。

然而逻辑建模的缺陷也很明显：随着知识图谱的规模爆发性地增长，即使利用马尔可夫毯（Markov Blanket）等局部依赖假设，对知识图谱中所有知识实例进行建模也是不可行的。

2. 隐含空间分析

第二种知识图谱构建方法是隐含空间分析。有时候一句简单的语句的背后会包含一些隐含的逻辑关系，例如：吴信东（首席科学家）给吴明辉（董事长）打电话。这一句简单的语句，背后会隐含各种各样的关系，例如：首席科学家应该做哪些事；和董事长应该讨论的内容等。目前隐含空间分析主要尚处于研究阶段，应用还不是很广泛。

距离模型

隐含空间分析的基本模型是距离模型，代表方法是结构表示（StructuredEmbedding，SE）：对于一个三元组（h, r, t），SE 将头实体向

量和尾实体向量通过关系的两个矩阵投影到关系的对应空间中，然后在该空间中计算两投影向量的距离。SE 模型的损失函数使用的是 L1 范数：

$$f_r(h,t) = \left| M_{r,1} l_h - M_{r,2} l_t \right|_{L_1}$$

由于 SE 模型对头、尾实体使用两个不同的矩阵进行投影，协同性较差，因此往往无法精确刻画两实体与关系之间的语义联系。由此提出了隐变量模型：

隐变量模型（LatentFactorModel，简称 LFM）

LFM 模型提出基于关系的双线性变换，刻画实体和关系的二阶联系，其评分函数为：

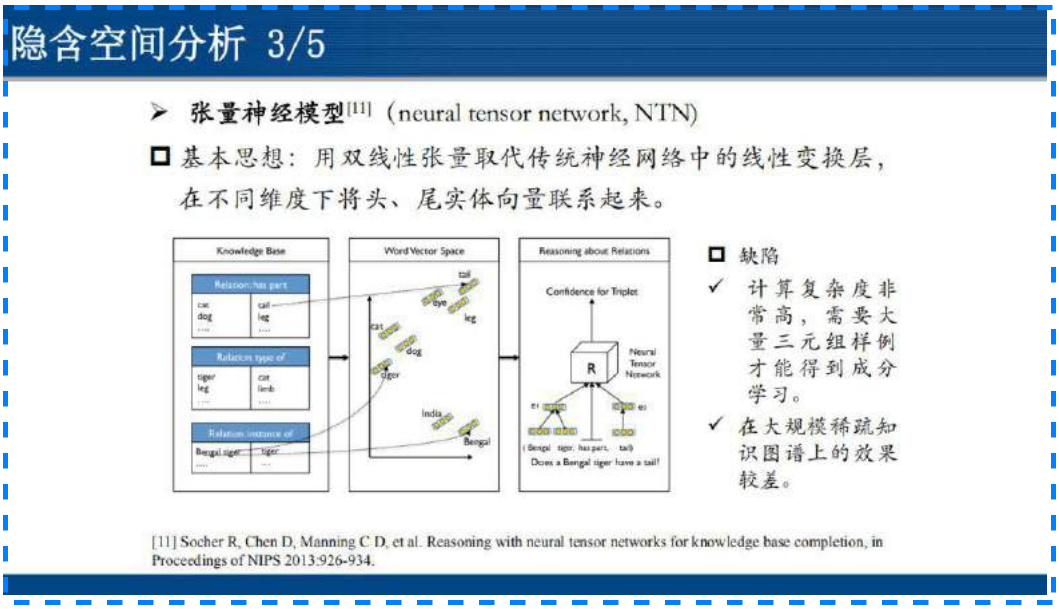
$$f_r(h,t) = l_h^T M_r l_t$$

其中，$M_r \in R^{dxd}$ 为关系 r 对应的双线性变换矩阵。

LFM 模型通过简单有效的方法刻画了实体和关系的语义联系，协同性较好，计算复杂度低。如何具体描述和刻画这个隐含的空间，会涉及到系数问题，于是引出张量神经模型：

张量神经模型（neuraltensornetwork，简称 NTN）

基本思想：用双线性张量取代传统神经网络中的线性变换层，在不同维度下将头、尾实体向量联系起来。



然而，NTN 模型计算复杂度非常高，需要大量三元组样例才能得到成分学习，因而在大规模稀疏知识图谱上的效果较差。由此引出矩阵分解模型：

矩阵分解模型

矩阵分解模型的代表方法是 RASACL 模型。

知识库三元组构成一个大的张量 X，如果三元组（h, r, t）存在，则 $X_{hrt}=1$，否则为 0。张量分解（矩阵分解）旨在将每个三元组（h, r, t）对应的张量值 X_{hrt} 分解为实体和关系表示，使得 X_{hrt} 尽量地接近于 $l_h M_r l_t$。这种模型的

缺陷是：时间复杂度和空间复杂度较高，且在大规模数据集上效率低、可扩展性差。

翻译模型

翻译模型的代表方法是 TransE 模型。对于每个三元组（h, r, t），将关系 r 的向量 l_r 看作头实体向量 l_h 和尾实体向量 l_t 的平移。

TransE 模型的参数较少，计算复杂度低，能直接建立实体和关系之间的复杂语义联系，但是在处理复杂关系时性能显著降低。

3. 人机交互

人机交互（Human-Computer Interaction, HCI）：是指人与计算机之间使用某种对话语言，以一定的交互方式，为完成确定任务的人与计算机之间的

信息交换过程。常见的方式就是：存在一个系统，什么都不懂，不断地向用户问问题；随着用户对问题的回答，系统逐步将图谱建立起来。

SIKT（structured interactive knowledge transfer program）是吴教授最早采用人机交互方式构建知识图谱的代表性工作之一，早期称之为"推理网络"，主要包括以下特色：

- 结构化人机交互知识库构建。
- 基于"rule schema + rule body"的知识表示形式。
- 程序通过交互接口指导学科专家自顶向下输入领域知识。
- 生成的知识库加上来自 KEShell 的预定义推理引擎构成最终的可执行程序。

DataFunTalk

SIKT [15]是最早的采用人机交互方式构建知识图谱的代表性工作之一。

- 结构化人机交互知识库构建
- 基于“rule schema + rule body”的知识表示形式
- 程序通过交互接口指导学科专家自顶向下输入具领域知识
- 生成的知识库加上来自KEShell [16]的预定义推理引擎构成最终的可执行程序。

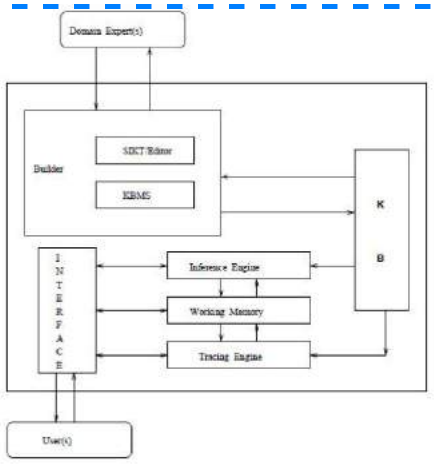


Figure 1: The System Structure of *KEshell*

[15] Wu X. SIKT: a structured interactive knowledge transfer program, International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems,1995:787-795.
[16] Wu X. KEShell: a "rule skeleton + rule body" -based knowledge engineering shell, Applications of Artificial Intelligence IX, 1991:632-639.

人机交互的另一种方法，也是吴教授的项目课题，是 IAKO（Interactive Acquisition of Knowledge Objects），即半结构化的知识图谱构建。

利用面向对象编程的优势，IAKO 基于知识对象（Knowledge Object）的表示方法，提出了一个**面向对象的交互**知识构建系统。

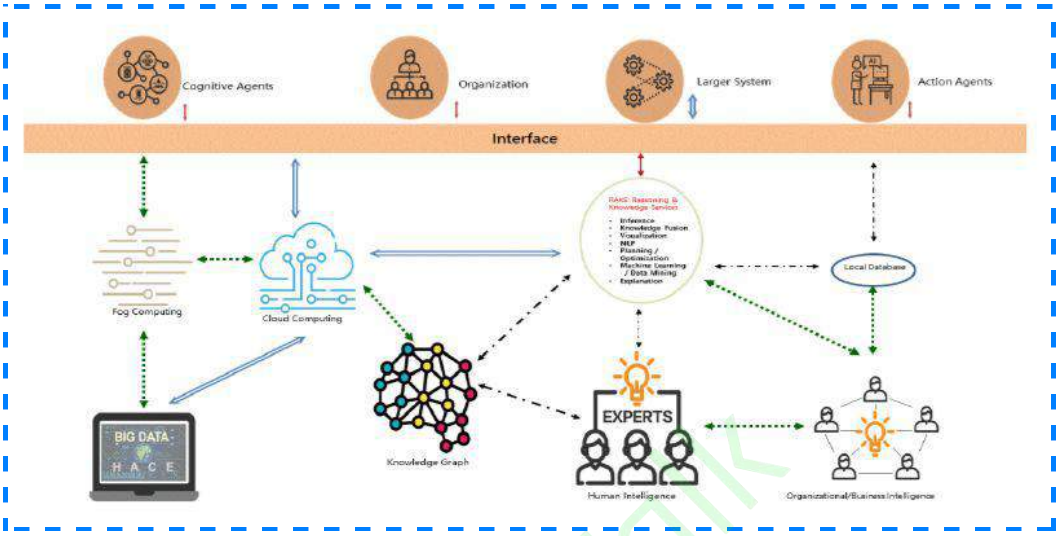
IAKO 能够从 0 开始，通过领域专家交互方式生成一套完整的知识库，且进行知识和规则校验，以保障知识库的可执行性。

基于知识对象的知识表示方式可以将规则融入对象中，达到 SIKT 中一组 "rule schema + rule body" 的知识表示能力。

IAKO 能够使得领域专家构建**便携的**和**可重用的**知识库。

人机交互的一种最新方法，是吴教授任职明略科技后同明略科技董事长吴明辉一起提出的一种模型，叫做 "HAO 模型"，该模型有效地融合了 Human Intelligence (HI)、Artificial Intelligence (AI)、Organizational Intelligence 这三种 "智慧"，在以人为本、人机协同的基础上，加入了面向行业应用、具体细分领域的 Organizational Intelligence。下图就是面向行业构建的知识图谱框架。

数据感知 -> 人机交互 -> 行动



在明略科技所涉及的公安领域，知识图谱是巨大的：在一个具体的公安数据分析系统里，有 16 亿个节点，40 亿条边，140 亿个事件，这样一个 7 度搜索可能需要 40 亿 7 次方的计算量，巨大到不可承受，因此可以看出这类知识图谱的构建中人工干预的必要性。

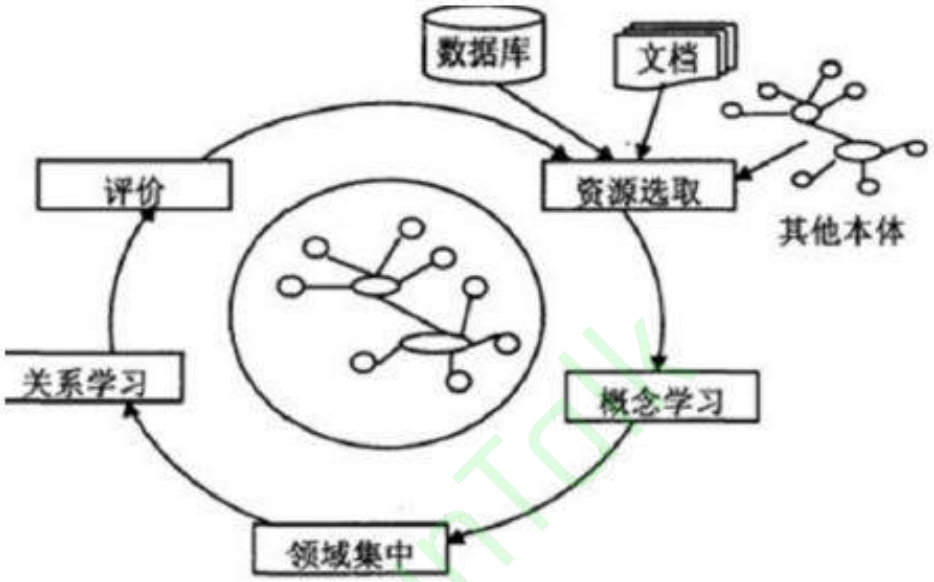
4. 本体模型支撑

本体模型，主要指：后台有一定的知识储备作为支撑。

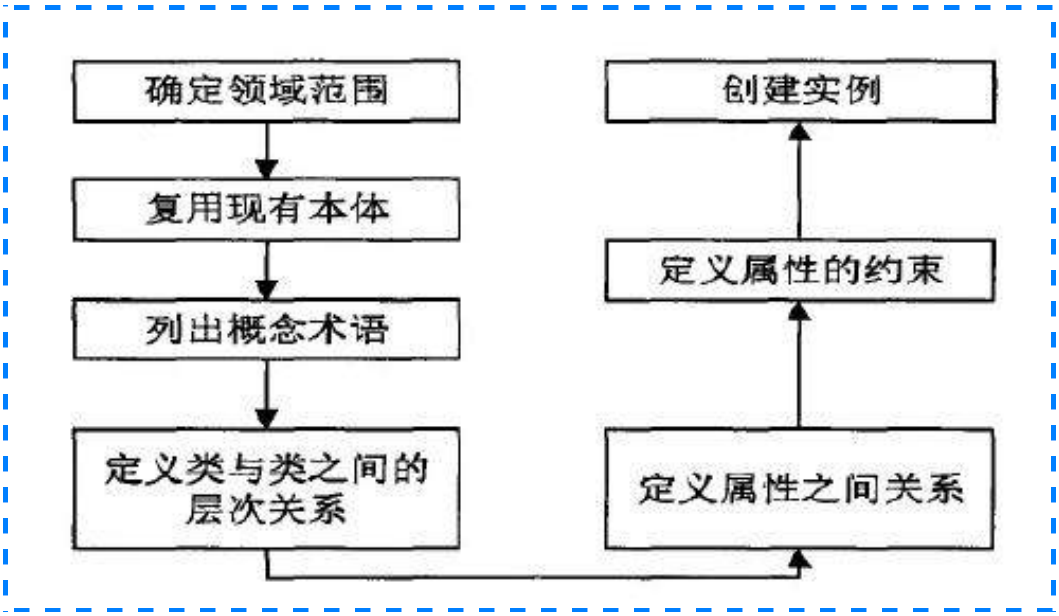
系统后台存在一个知识库作为本体的支撑，根据输入语言的特征、关键词等去匹配后台的知识库。这里会涉及到较多的机器学习模型。

人工构建

人工构建本体模型支撑，由大量的领域专家相互协作构建本体，用以支撑前台做文本分析；代表方法包括循环获取法、七步法等。



Cyc 循环获取法



Ontology Development 101（七步法）

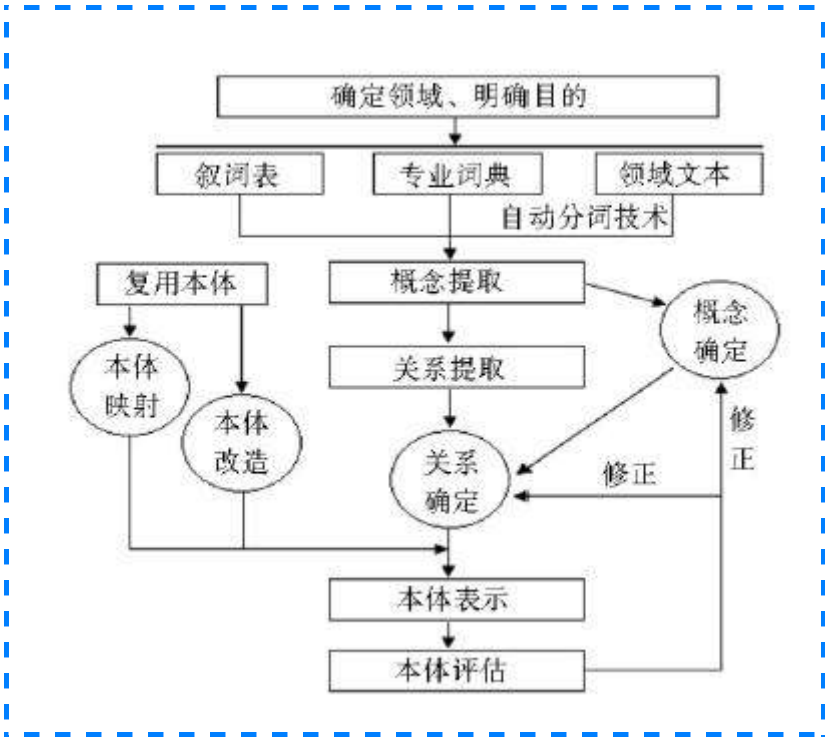
七步法的缺点是：主观性太强，且比较随意，缺少科学管理和评价机制。

半自动构建

半自动构建本体，利用相关领域内的专业词典、叙词表等专家知识，从中抽取感兴趣的概念和关系，构建需要的本体。

缺点：复用本体中的概念和关系，带来了不同本体匹配的问题。

DataFunTalk

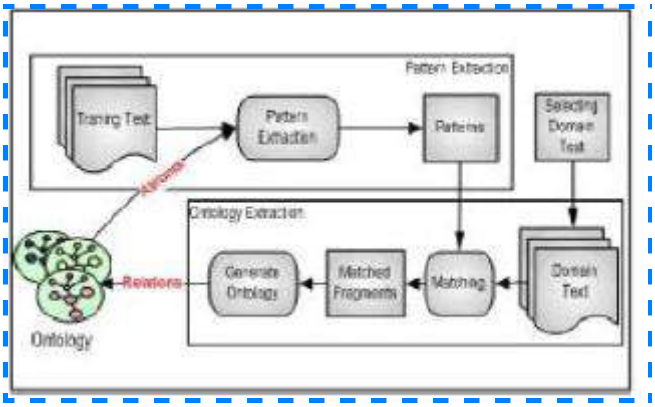


自动构建

自动构建本体，指的是利用知识获取技术、机器学习技术以及统计技术等从数据资源中自动获取本体知识。

主要涉及到两种方法：一种是基于语言规则的方法，另一种是基于统计分析的机器学习方法。

DataFunTalk



基于语言规则的方法，主要指基于语义模式，从自然域文本构建本体，通过对自然域文本的分析，提取候选关系并将其映射到语义表示中，实现本体的构建。

如上图所示，其基本框架为：

- 使用模式编辑器构造语义模式
- 选择自然域文本
- 从自然域文本中提取领域本体

其优点是：以非分类学关系丰富了浅层本体，一个动词可以表示两个或多个概念之间的关系。然而，这种方法不会发现新的关系，只是发现已知关系实例；而且本体构建的效果依赖于语义模式，因而需事先构建较完备的语义模式。

DataFunTalk

基于统计分析的机器学习方法，主要是基于数据聚类和模式树挖掘，进行结构化的本体构建。

- 对用于构建每个组的本体书类似文档进行分组
- 使用模式树挖掘从部分本体树构建集成本体

这种方法主要有两个主要模块：

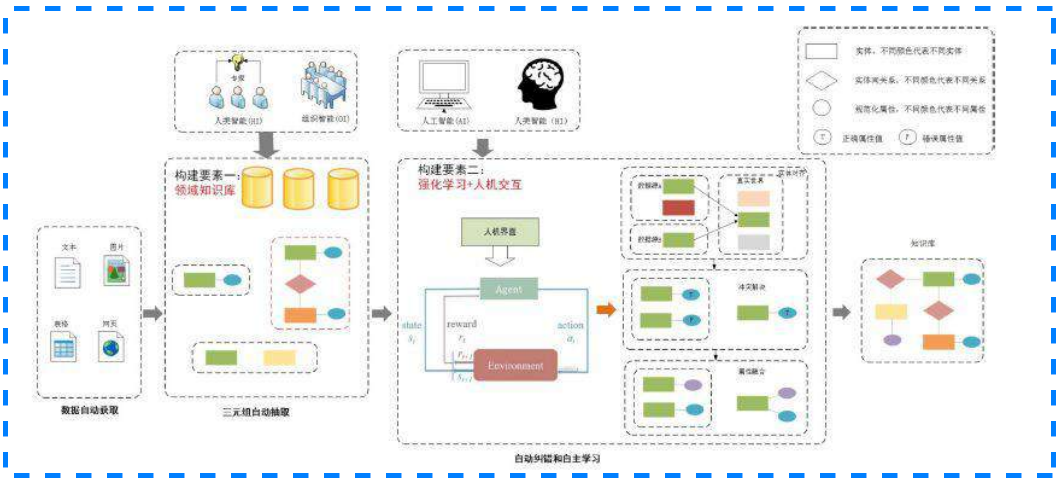
- 文档聚类：使用检索关键字的关系矩阵通过潜在语义分析（LSA）和 K-means 方法来聚类文档。
- 本体构建：通过形式概念分析和本体集成构建每组文档的本体。

统计机器学习方法可适用于范围更广的领域，可构建的本体倾向于更好地描述概念间的关系，结构也更加复杂；然而这种方法缺乏必要的语义逻辑基础，抽取概念关系松散且可信度无法得到很好的保证。

——图谱自动构建要素——

1. 总体设计框架

DataFunTalk



总体架构的两个核心要素，一个要素是后台的领域知识库，另一个要素是强化学习配合人机交互。

步骤一：数据自动获取

通过使用较为流行的网络数据获取工具，如：Scrapy，Jspider，Larbin 等获取多源异构数据。

步骤二：三元组自动抽取

结合自然语言处理工具和领域知识库，初步识别和抽取文本中的三元组信息。

构建要素一：领域知识库

在图谱自动构建过程中，由领域专家（HI）和专业组织（OI）提供的领域知识库能够有效提高实体、关系的识别和抽取精度。

DataFunTalk

步骤三：自动纠错和自主学习

结合 HAO 智能模型和强化学习方法，通过人机交互接口对代表性错误三元组进行人工纠正，并以此对强化学习模型进行训练和提高，实现自动纠错和自主学习。

构建要素二：强化学习+人机交互

为保证图谱构建质量，需要通过人机交互接口对错误信息进行人工纠正，并以此作为种子案例，通过强化学习加强模型的识别精度和鲁棒性。

——应用场景——

应用场景一：网络行为动态分析

通过网络爬虫获取最新的网络信息数据，运用知识图谱自动构建技术，动态更新和扩充现有知识库，为网络行为分析提供知识支撑。包括：

舆情监测

热点跟踪

用户情感倾向分析

用户设计网络影响力分析

DataFunTalk

这一部分也是明略科技涉及业务比较多的方面，涉及品牌分析，为大客户做广告设计，推荐广告布局，并做广告监测，以及品牌效应分析；这一部分的后台就是基于知识图谱的基础上做舆情分析、热点跟踪等。

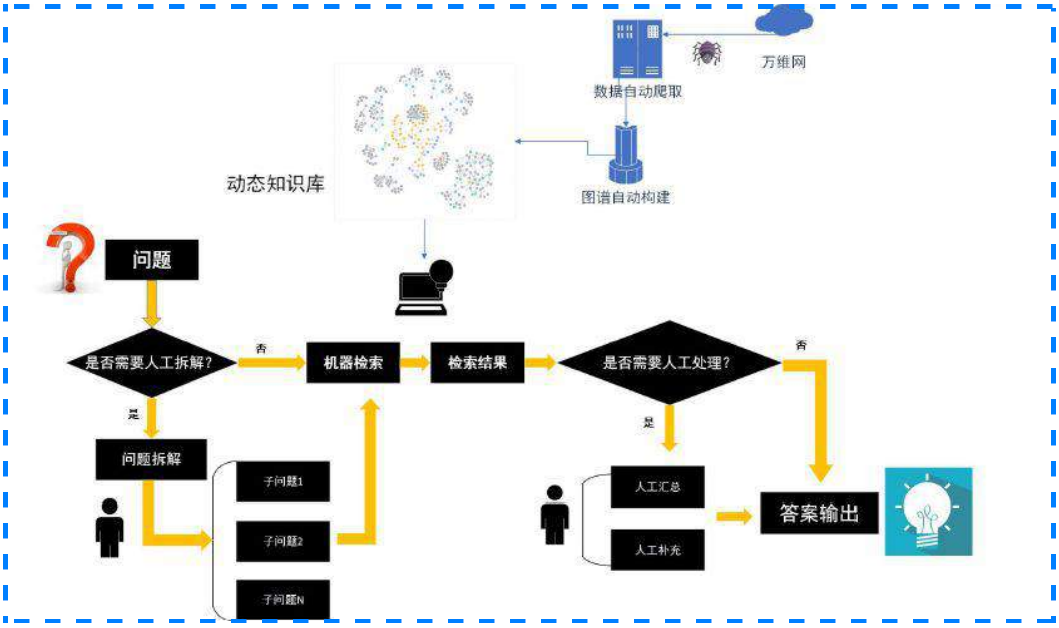
应用场景二：智能 Q&A

通过人机交互+知识图谱自动构建，设计更智能的知识问答系统。

人机交互：人与机器进行工作互补，共同完成问答场景。

知识图谱自动构建：动态更新和扩充问答知识库，响应最新的网络知识。

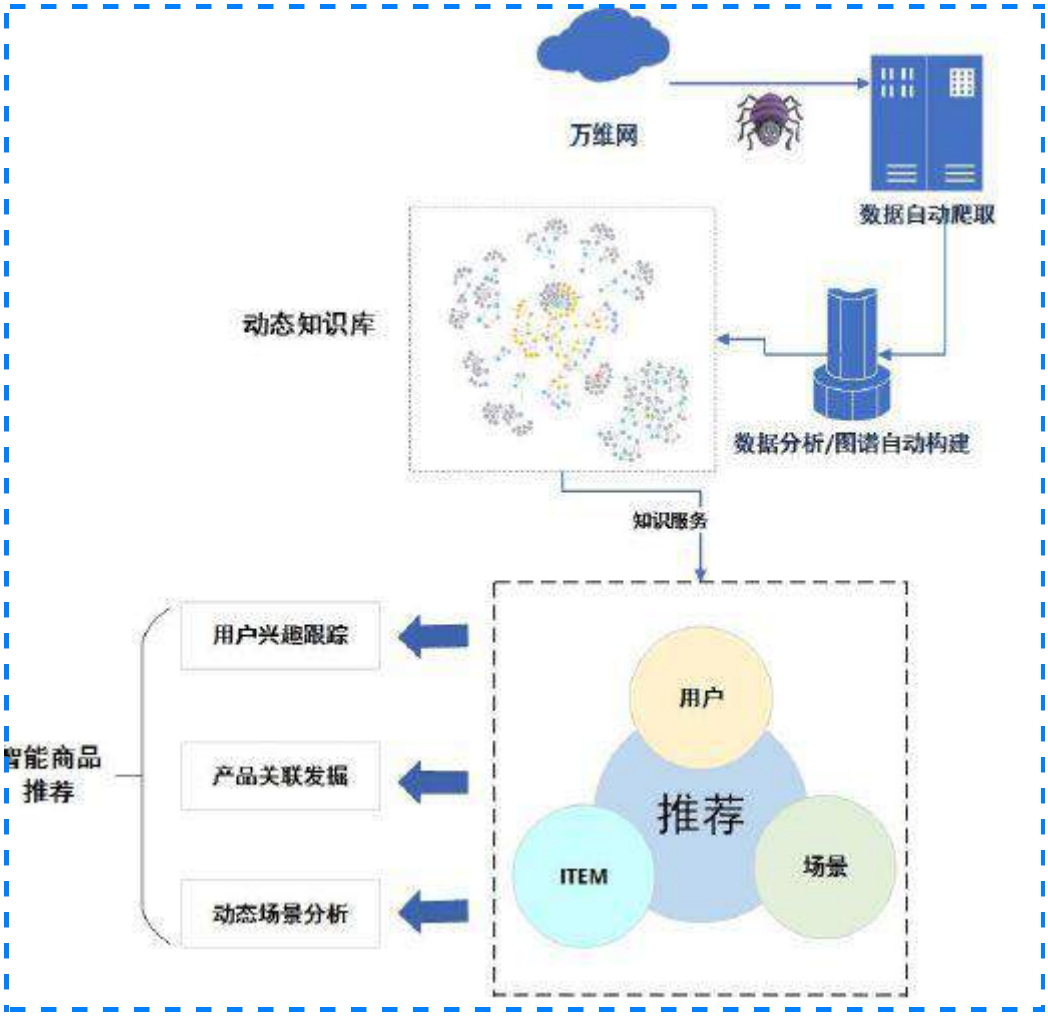
DataFunTalk



明略科技今年的战略发展之一就是智能 Q&A 产品，类似于科大讯飞的相关产品，前台问答的同时，后台同步构建知识图谱。

应用场景三：智能推荐

DataFunTalk



对于推荐系统中的核心元素：用户、场景和 ITEM，利用知识图谱自动构建技术，实时获取互联网海量信息并更新推荐知识库，通过用户兴趣跟踪、产品关联发掘、动态场景分析等方法提供更智能的推荐服务，涉及到用户跟踪，涉及产品动态变化等。

今天的分享就到这里，谢谢大家。

——参考资料——

[1] WuX, et al. Data mining with big data, IEEE TKDE, 2014, 26(1): 97-107.

[2] WuX, et al. Knowledge Engineering with Big Data, IEEE IntelligentSystems,2015,30(5):46-55.

[3] 赵军等. 知识图谱[M]. 高等教育出版社, 2018.

[4] 吴信东,张奠成.从关系型描述数据库生成语义网络的方法,科学通报,1990:1674-1676.

[5] FahiemBacchus. Representing and reasoning with probabilistic knowledge, The MITPress, 1990.

[6] Nils J Nilsson. Probabilistic logic, Artificial Intelligence, 1986, 28(1): 71-87.

[7] Matthew Richardson, Pedro Domingos.Markov logic networks, Machine Learning, 2006, 62(1-2): 107-136.

[8] BordesA, Weston J, CollobertR, et al. Learning structured embeddings of knowledge bases, in Proceedings ofAAAI 2011, 2011:301-306.

[9] SutskeverI, Tenenbaum J B, SalakhutdinovR, et al. Modelling Relational Data using Bayesian Clustered TensorFactorization[C], in Proceedings of NIPS 2009:1821-1828.

[10] JenattonR, Roux N L, BordesA, et al. A latent factor model for highly multi-relational data[C], inProceedings of NIPS 2012:3167-3175.

[11] SocherR, Chen D, Manning C D, et al. Reasoning with neural tensor networks forknowledge base completion, in Proceedings of NIPS 2013:926-934.

[12] Nickel M, TrespV, KriegelH. A three-way model for collective learning on multi-relational data, inProceedings of ICML 2011:809–816.

[13] CaiH Y, Zheng V W, Chang K. A Comprehensive Survey of Graph Embedding: Problems,Techniques and Applications. IEEE Transactions on Knowledge and DataEngineering, 2018. 3(9): 1616-1637.

[14] BordesA, UsunierN, Garcia-Duran A, et al. Translating Embeddings for Modeling Multi-relationalData, in Proceedings of NIPS 2013:2787-2795.

[15] Wu X. SIKT: a structured interactive knowledge transfer program, InternationalConference on Industrial & Engineering Applications of ArtificialIntelligence & Expert Systems,1995:787-795.

[16] Wu X. KEShell:a "rule skeleton + rule body" -based knowledge engineering shell,Applications of Artificial Intelligence IX, 1991:632-639.

[17] Xu L , Wu X . Interactive Acquisition of Knowledge Objects, In Proceedings ofIEEE Knowledgeand Data Engineering Exchange Workshop, 1997:97-105.

[18] MinghuiWu and XindongWu, On Big Wisdom, Knowledgeand Information Systems, 2019, 58(1): 1-8.

[19] SwartoutB, Patil R, Knight K, et al.Toward distributed use of large-scale ontologies. In: Proc. of the 10thWorkshop on Knowledge Acquisition for Knowledge-Based Systems, 1996:138-148.

[20] NoyNF, McGuinness DL. Ontology development 101: A guide to creating your firstontology. 2001.

https://doi.org/10.1016/j.artmed.2004.01.014

[21] SuryantoH, Compton P. Discovery of ontologies from knowledge bases, ACM InternationalConference on Knowledge Capture, 2001.

[22] Dahab M Y , Hassan H A , Rafea A . TextOntoEx: Automatic ontology constructionfrom natural English text[J]. Expert Systems With Applications, 2008,34(2):1474-1480.

[23] Yu Y T , Hsu C C . A structured ontology construction by using data clustering andpattern tree mining, International Conference on Machine Learning andCybernetics, ICMLC 2011.

DataFunTalk

知识结构化在阿里小蜜中的应用

嘉宾：李凤麟　编辑：付一韬

导读：阿里小蜜是阿里巴巴服务领域的重要人工智能产品，是服务于阿里巴巴经济体、商家、企业和政府的对话机器人家族，包括阿里小蜜、店小蜜、云小蜜。小蜜机器人是基于大数据和人工智能技术形成的智能化、体系化客服系统，可以辅助企业智能决策、节约服务成本、提高服务效率、提升用户体验。作为阿里小蜜的基石，知识是小蜜体系中不可或缺的组成部分。小蜜的知识不仅仅包括常见的 FAQ，还包括词组、知识图谱、机器阅读文档等丰富的结构形态。本文主要分享阿里小蜜体系中知识的结构化以及在实际场景中的应用。

本文的主要内容包括以下几点：

- 阿里小蜜的简介
- 知识的结构化
- 基于结构化知识的应用：KBQA 和 EBQA
- 展望和挑战

阿里小蜜的简介

DataFunTalk

阿里小蜜机器人是一个对话机器人家族体系，不仅是指支持经济体生态的阿里小蜜，还包括了支持商家生态的店小蜜以及支持企业生态的云小蜜。在机器人的底座，是一套体系化的支撑平台，涵盖机器人的开发、配置、训练和知识等重要方面。其中，本文主要介绍的知识云是阿里小蜜的新一代知识库系统，支持基于词组、FAQ、知识图谱、机器阅读文档在内的丰富的知识体系，为阿里小蜜提供高效可靠的知识服务。

知识的结构化

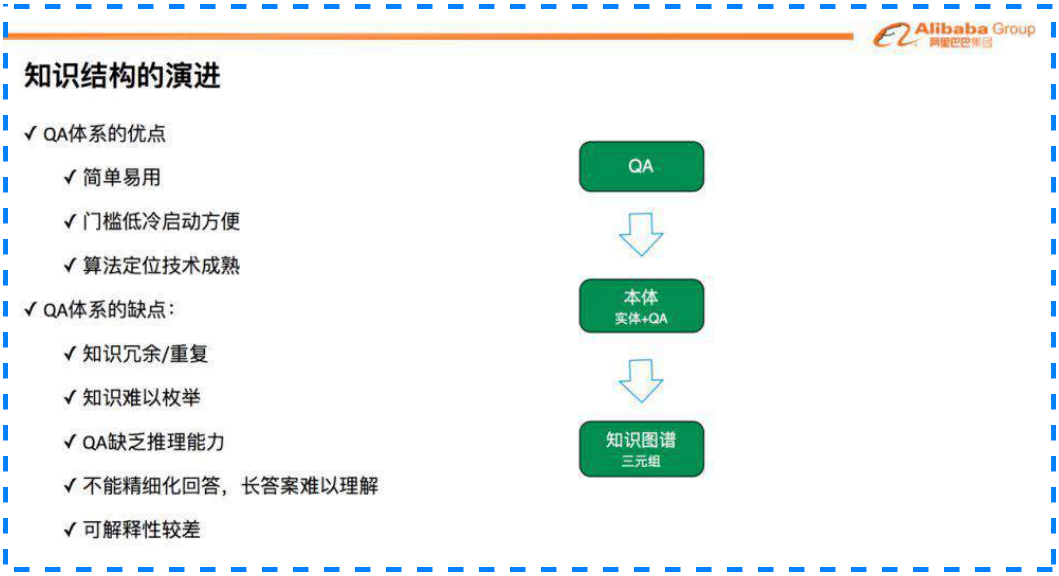
1. 小蜜的 AI 闭环

DataFunTalk

在小蜜的体系中，我们通常基于知识来开发算法模型。在模型的冷启动阶段，我们会借助智能辅助产品 AIBoost，通过人工标注、算法推荐、数据增广相结合的方式，来产生模型的训练语料。在模型开发上线以后，针对模型不能回答的问题，我们会回流补充到训练数据集中，继续训练和优化模型。如果发现是因为知识缺失造成不能回答，我们会对知识进行相应的补充。整个过程构成了一个完整的 "AI 闭环"。

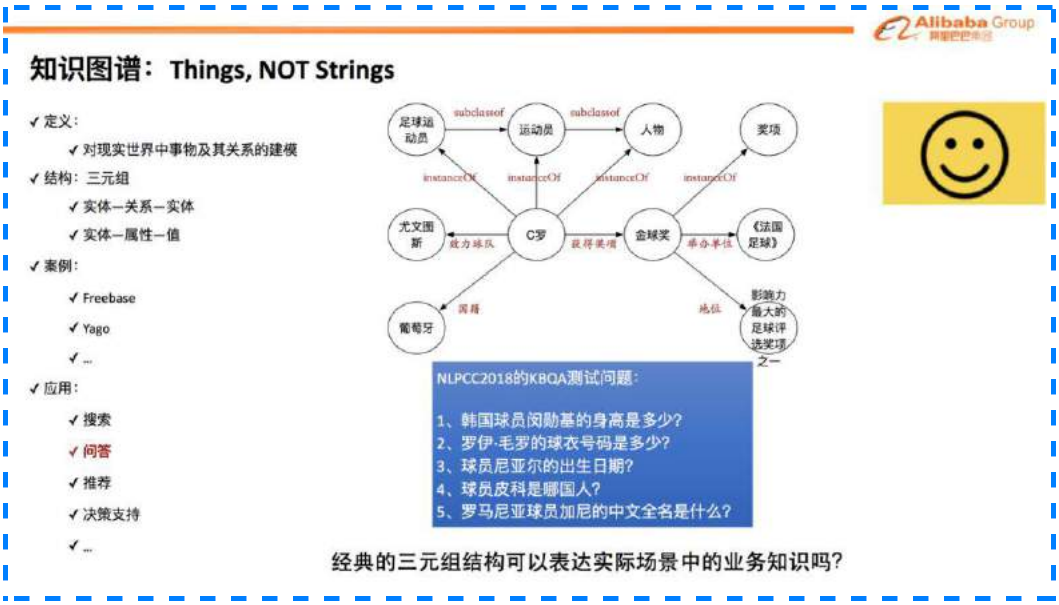
2. 知识结构的演进

DataFunTalk



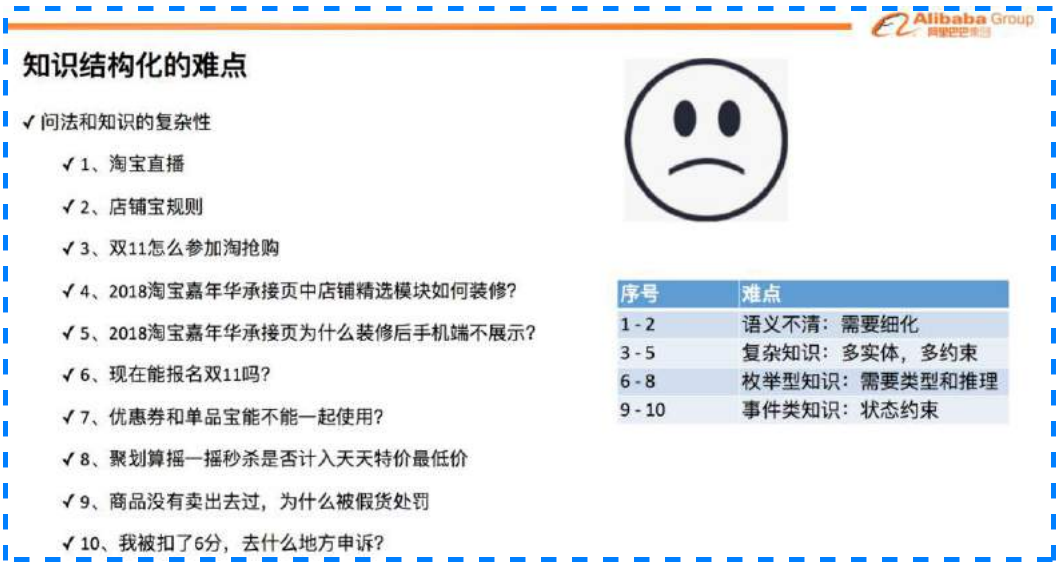
在开始阶段，小蜜采用 QA 体系作为主要的知识结构，并且取得了非常不错的效果。QA 结构具有简单易用、冷启动方便、算法成熟的一系列优点。但是，随着场景越来越丰富和复杂，QA 的解决能力遇到了瓶颈的问题。这是由于 QA 体系存在着一些缺陷。其一是知识的类似性或者冗余性。例如，"双 11 的报名入口在哪" 和 "双 12 的报名入口在哪"，这两条知识都是关于 "报名入口"，但是属于不同的活动。如果采用 QA 的方式，每个活动这种类似的知识都必须单独枚举出来，实际业务中不仅需要大量的知识维护成本，同时对算法模型的定位匹配也会造成很大的干扰。另一个缺点是 QA 缺乏推理能力，例如，"我现在还能不能参加今年的双 11" 这个问题，是一个关于时间节点的知识。其实，在双 11 的知识条目里面，已经清楚的描述了参与报名的时间段。但是，在传统的 QA 体系下，这个问题是没法去进行推理和准确回答的。

3. 知识图谱

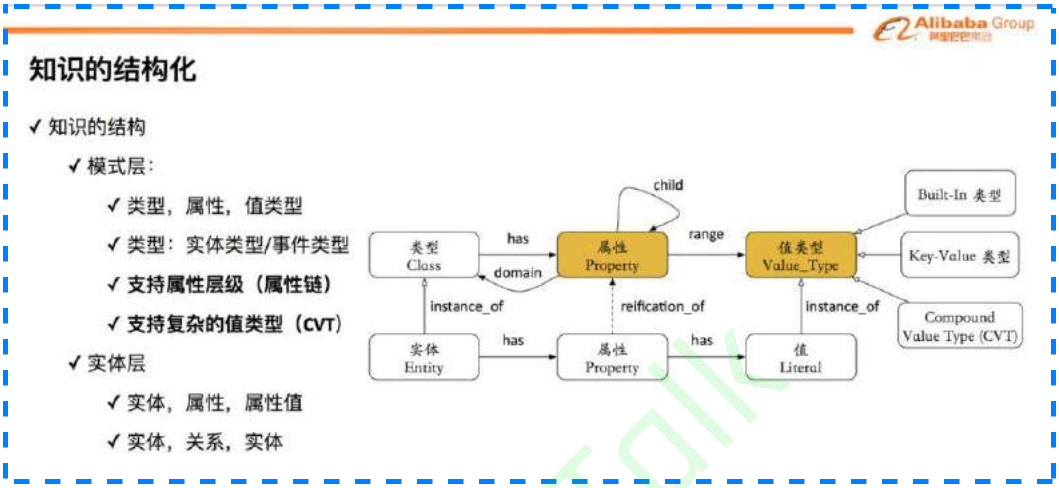


提到知识的结构化，我们首先会想到的通常是知识图谱（KG）和经典的三元组结构："实体-关系-实体" 和 "实体-属性-值" 这样的表达。但遗憾的是，这样的经典三元组结构表达能力是有限的，在实际场景中无法全面的表达业务知识。下图列举了实际场景中的一些 QA 知识，并总结了知识结构化的一些难点。

DataFunTalk



4. 知识的结构化表示



因为经典三元组表达能力的局限，在阿里小蜜中我们重新设计了知识结构。主要分为两个层级：模式层和实体层

❶ 模式层

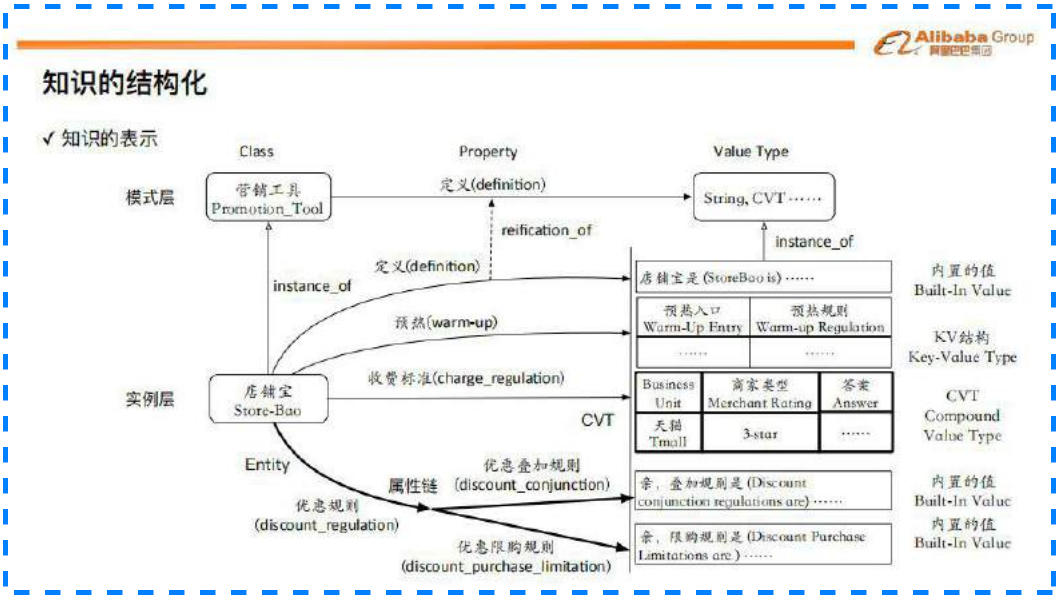
在模式层里面，我们定义类型、属性和值（以及关系）。其中类型可以包括实体类型和事件类型；对于属性，我们也做了一个扩展，支持子属性，也就是说我们的属性是有层级关系的，可以组成属性链；对于值的定义除了保留原来 Text/字符串（Built-In 类型）外，还加入两个新的类型：Key-Value 类型和 Compound Value Type（CVT）类型。我们的这些扩展，可以充分增强 KG 的表达能力，应用于实际业务知识的表达和建模。

❷ 实体层

实体层是对模式层的实例化。在实体层里面，我们采用类似的 "实体-属性-值" 和 "实体-关系-实体" 来对知识进行建模。

下图是结构化知识表示的一个示例。其中，"营销工具" 是一个类型，店铺宝是一个实例，一种具体的营销工具。"定义"，"预热"，"收费标准"，"优惠规则" 是属性。其中，"预热" 的属性值是一个 KV 结构（可以支持更为精细的定位），收费标准的属性值是一个 CVT（不同的 BU 不同的商家类型收费标准不一样），"优惠叠加规则" 和 "优惠限购规则" 是 "优惠规则" 的子属性。

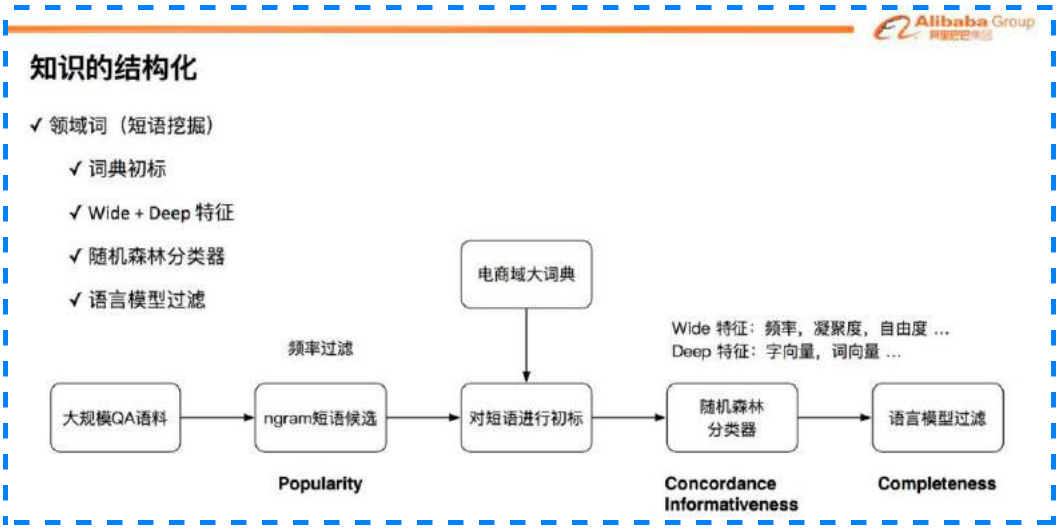
DataFunTalk



5. 知识的结构化实践

在实际的落地过程中，我们从 QA 知识和语料出发来进行具体的知识结构化。整体步骤是先进行基于短语的领域词挖掘，然后基于挖掘的领域词进行概念/实体提取，进一步获得属性和关系，然后人工校验并且设计更为精细和复杂的 CVT 结构。下面分别对每一步骤进行详细介绍。

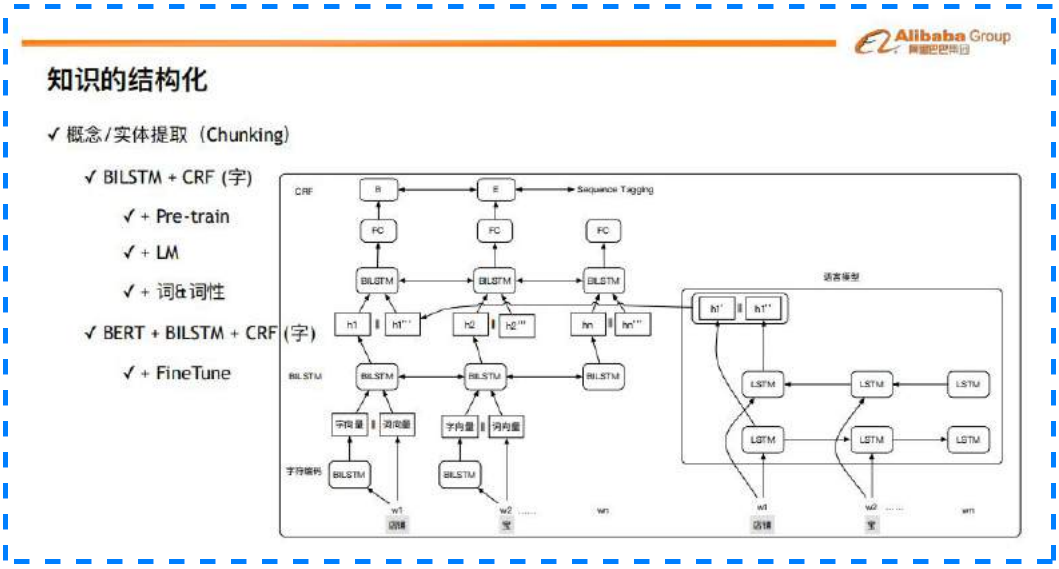
❶ 领域词（短语挖掘）



对于基于短语的领域词挖掘，我们先从语料库中自动产生大量的 n-gram 候选集（n 是一个正整数，例如 2~8），然后使用电商词典对候选集进行初标，得到一个训练样本集。以此为基础，我们采用随机森林，采样正负样本训练多棵树来构建分类器，以对候选短语的质量进行评分。分类器的特征包含两类：一类是 Wide 统计特征：频率、凝聚度、自由度等等；另一类是 Deep 语义特征：字向量、词向量、基于词向量的内部相似度等。最后，我们使用预训练的 BERT 领域语言模型对前述分类的候选短语进行修剪（pruning）得到领域词。

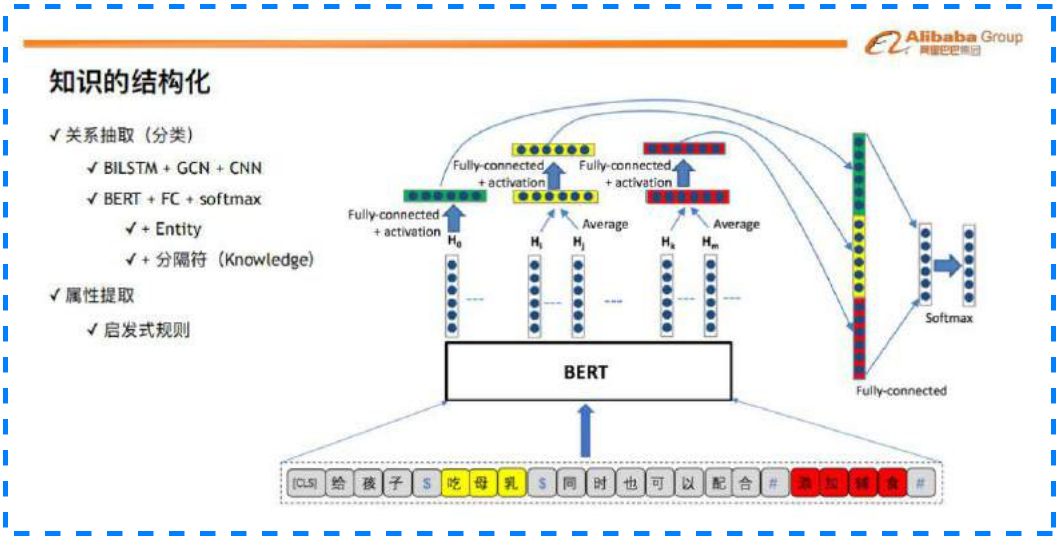
❷ 概念/实体提取（Chunking）

DataFunTalk



在概念/实体提取部分，我们在基于字的 BILSTM+CRF 框架基础之上，融入了词性、Pre-train 和语言模型知识。我们对嵌入层（Embedder）做了两点扩展：1. Embedder 的输入是已经分好词的句子，词的向量由两部分拼接组成，一部分是 Word2Vector 预训练好的词向量，另一部分是经过一个 BiLSTM 模型处理后的首尾字符拼接向量；2. 在第一层向量拼接之后，送入到第二层 BILSTM，然后其输出和我们预先训练好的 ELMO 领域语言模型的结果进行拼接。编码层（Encoder）采用一个 BILSTM 模型，并且我们用通用领域的语料进行了 Chunking 任务的预训练（Pre-training）。解码层（Decoder）采用 CRF。实验结果表明，词性、Pre-train，语言模型知识有助于效果的明显提升。在此基础上，我们进一步把嵌入层换成了基于领域预料微调的 BERT 模型，使用 BERT + BILSTM + CRF 的框架，效果得到了继续有效的提升。

❸ 关系抽取（分类）



对于关系的抽取，我们尝试了图神经网络 GCN 和 BERT，BERT 的表现效果更好。在 BERT 的输入中，我们引入概念/实体信息（图中的 "吃母乳" 和 "添加辅食"），对 BERT 的输出中实体对应的多个字向量进行 averaging 和 FC（full-connection）操作；另外，我们尝在实体两侧加上分隔符（$ 和 #）来显示的告诉模型分隔符中间是概念/实体，可以进一步得到明显的提升。系列实验不仅证明了预训练语言模型的有效性，而且验证了引入实体知识可以有效的提升效果。

❹ Schema 编辑器

通过上述的短语挖掘、概念/实体提取、关系/属性提取，我们可以得到初步的知识结构。这个结构需要业务同学进行人工校验，同时对属性值的

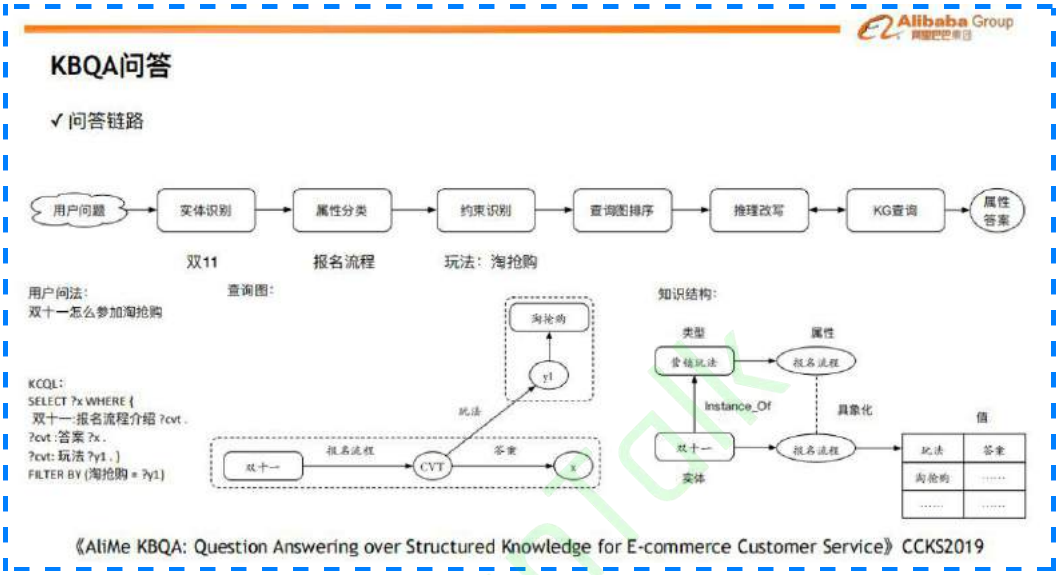
CVT 结构进行设计和加工，最终形成我们的 Schema 结构，并且完成实体级别知识的录入。

❺ 实体编辑器

基于结构化知识的应用：KBQA

在知识结构化之后，我们选择采用 KBQA 来在实际场景中进行落地。具体地，我们选择了 KBQA 的多约束图排序方案：通过识别用户问题中的实体、属性、约束（CVT 子属性），将用户的问题转换成可能的多个候选图，然后对候选图进行排序，选择得分最高的图，根据情况进行推理改写，并最终查询 KG 返回相应的结果。在这个过程中，可能还需要进行推荐或者反问以应对语义不清或者问题不完整的情况。下图展示展示了 KBQA 的基本问答流程。

1. KBQA 问答流程

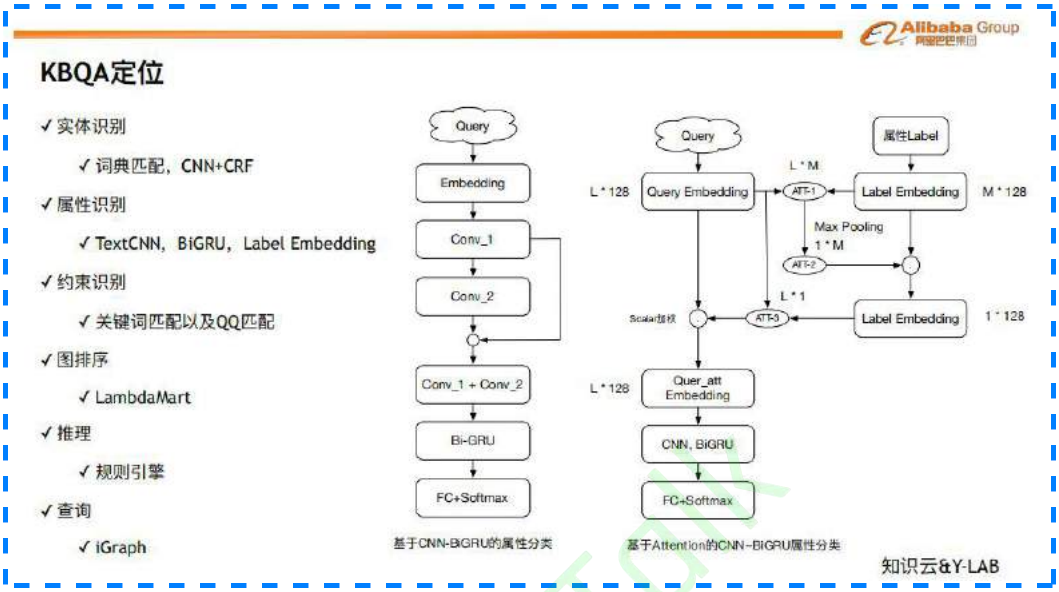


例如 :用户问 "双十一怎么参加淘抢购"（ 淘抢购是双十一的一种玩法 ），这里面识别的主实体是 "双十一"，属性分类结果是 "报名流程"，约束识别是 "玩法：淘抢购"，最后根据这些片段组成的查询图，去 KG 中获取答案。

2. KBQA 定位

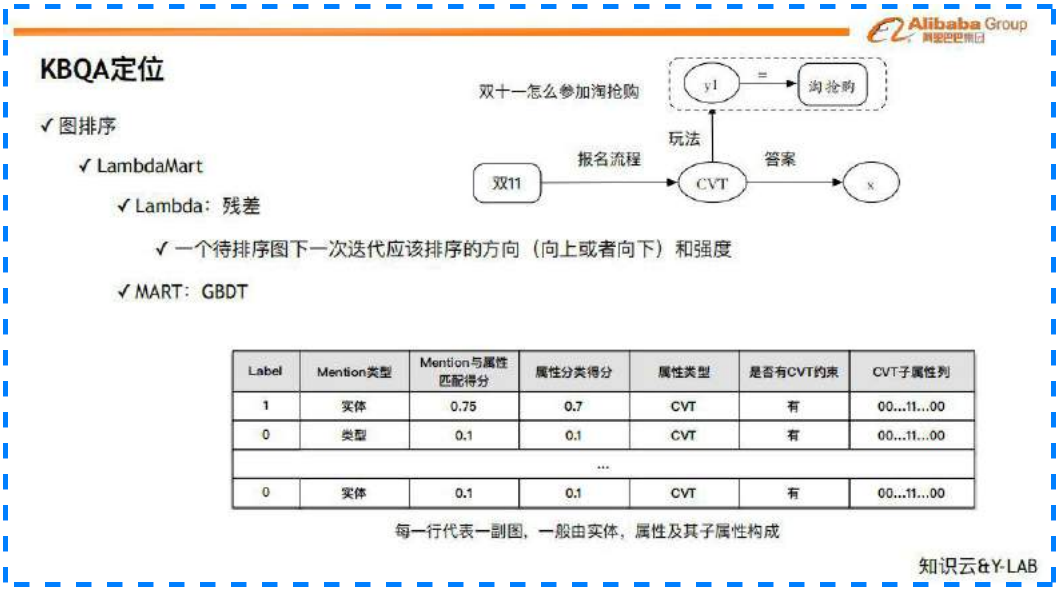
❶ 实体&属性&约束

对于 KBQA 的实体识别，我们采用了基于词典的规则方法和基于 CNN+CRF 的实体识别模型。对于属性识别，我们采用了基于 CNN+BiGRU+ Attention 等比较成熟的深度学习分类模型。对于约束识别，采用了关键词匹配和相似匹配的方法。



接下来我们着重对图排序、类型推理和推荐进行介绍：

❷ 图排序

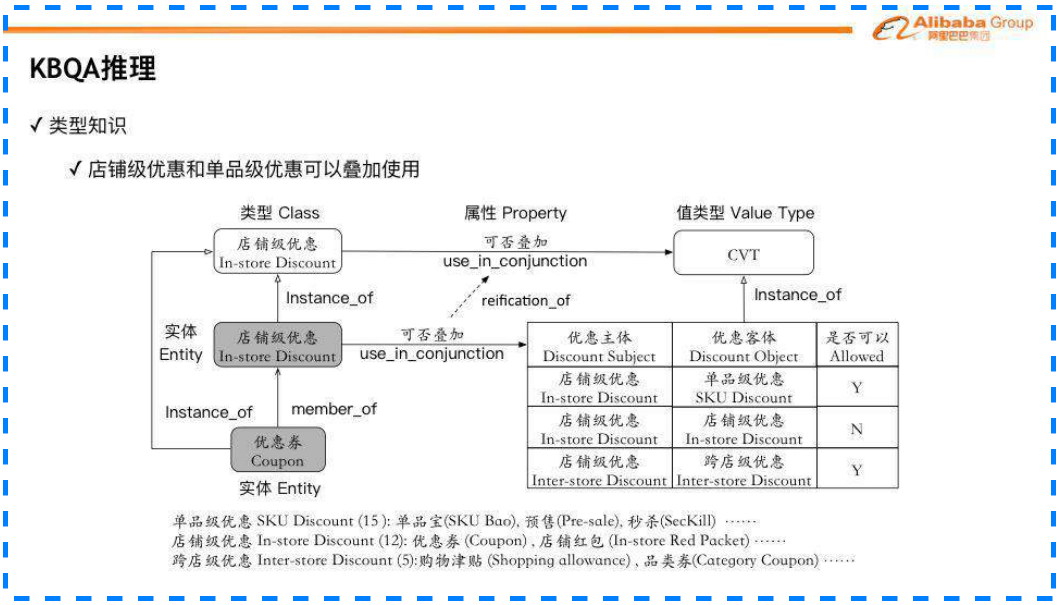


在识别出来实体、属性、约束的候选之后，我们会把相应的候选组合成一幅候选图。例如，上图右上角就对应了用户问题"双十一怎么参加淘抢购"的一个候选图。需要注意的是，因为实体、属性、约束都可能存在多个候选，所以这幅图仅仅是是多幅候选图中的一幅。

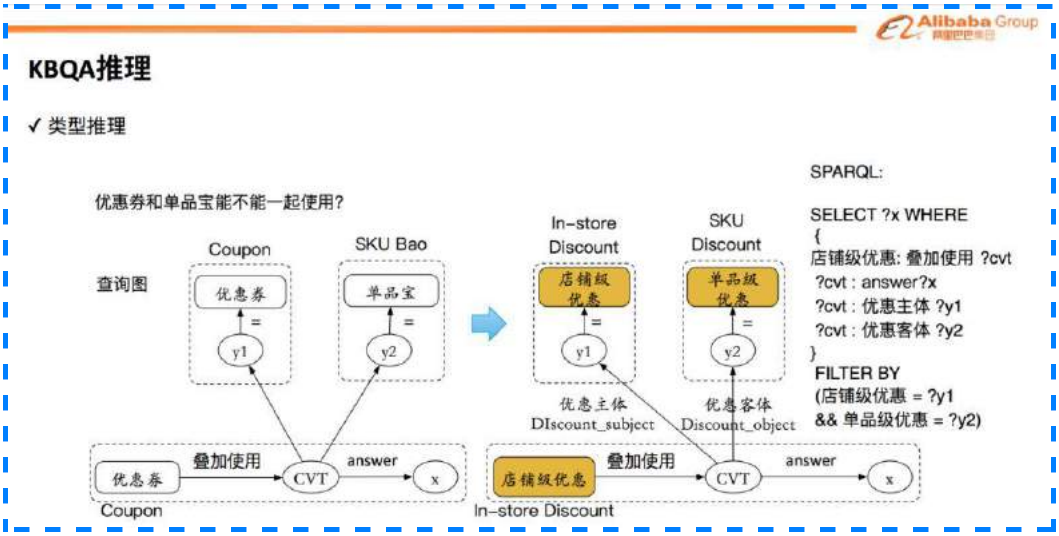
上图表格中的每一行都是对一幅候选图的特征刻画，Label 表示了当前图是否是对应的答案。我们采用 LambdaMart 进行排序，这里 Lambda 刻画了排序模型训练中每一幅候选图的梯度（这幅图该向前排还向后排，以及朝这个方向移动多少），MART 就是我们熟悉的 GBDT，来学习这个梯度。

❸ 类型推理

在实际场景中，我们常常遇到很多需要枚举的实体型知识。例如，"优惠券和单品宝能不能一起使用？"，这里面 "优惠券" 和 "单品宝" 是两个实体，其中 "优惠券" 是店铺级的优惠，"单品宝" 是单品级的优惠。因为无论是店铺级，还是单品级的优惠，可能都存在很多种，如果我们进行枚举的话，不仅数量很大而难以维护，也容易相互干扰造成定位错误。



针对实体型知识，我们从实体级别抽象到类型级别，例如 "店铺级优惠和单品级优惠可以叠加使用"，进行相应的表示和推理来进行问答。上图展示了如何对类型级别的知识和实体级别的知识进行表示和关联。

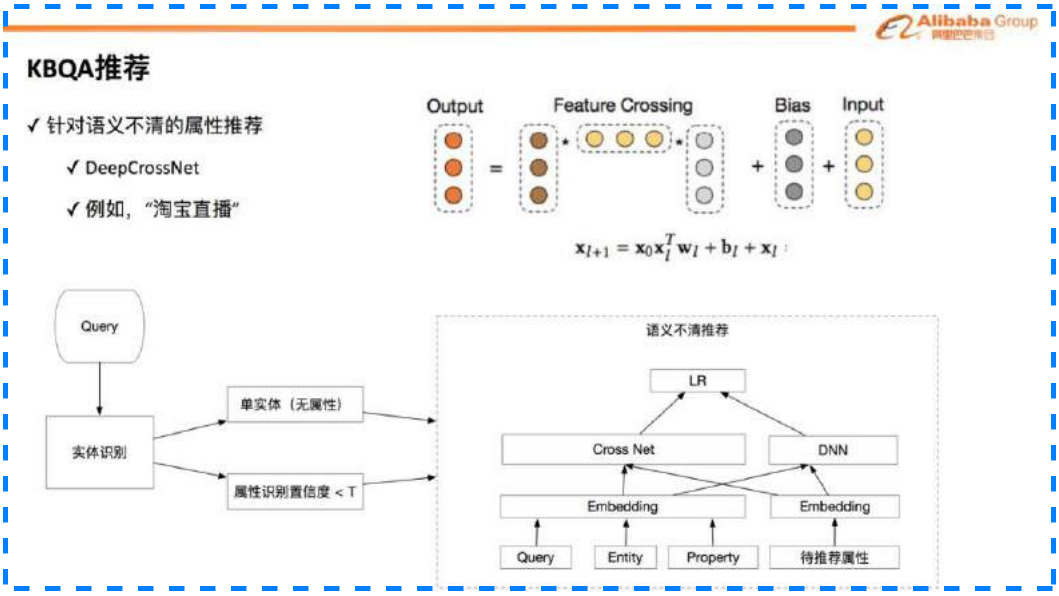


问答过程中，如上图所示，经过类型推理以后，问题 "优惠券和单品宝能不能一起使用？" 会被改写成 "店铺级优惠和单品级优惠能不能一起叠加使用"，并且最终通过查询 KG 返回对应的答案。

采用类型级别知识的表示和类型推理，一方面可以大幅降低需要维护的知识数量，同时可以有效的改善定位准确率，并且在答案中清晰的给出原因解释。

❹ KBQA 推荐

DataFunTalk



用户问题还有一个经常出现的情况就是"语义不清"。例如，用户说了"淘宝直播"，其实我们并不知道他具体想要问什么。这种情况下，一般做法是根据热点进行反问。在这里，我们做了进一步的扩展，采用了 DeepCrossNet 技术来进行推荐。我们对 Query 进行实体识别，然后基于热度、问题上下文、Schema 结构等信息对候选的属性进行排序推荐，有效的提升了推荐点击率。

3. KBQA 的 DEMO 示例

KBQA 目前已经在阿里小蜜的大促机器人、商家机器人（万象）、淘宝直播的直播小蜜机器人等场景实际落地，并且经受了双 11 的洗礼，取得了非常不错的效果。下列 Demo 对 KBQA 的问答效果进行了一个简单的展示。

基于类型推理的问答示例：

基于 KV 的问答示例：

基于 CVT 的精准问答示例：

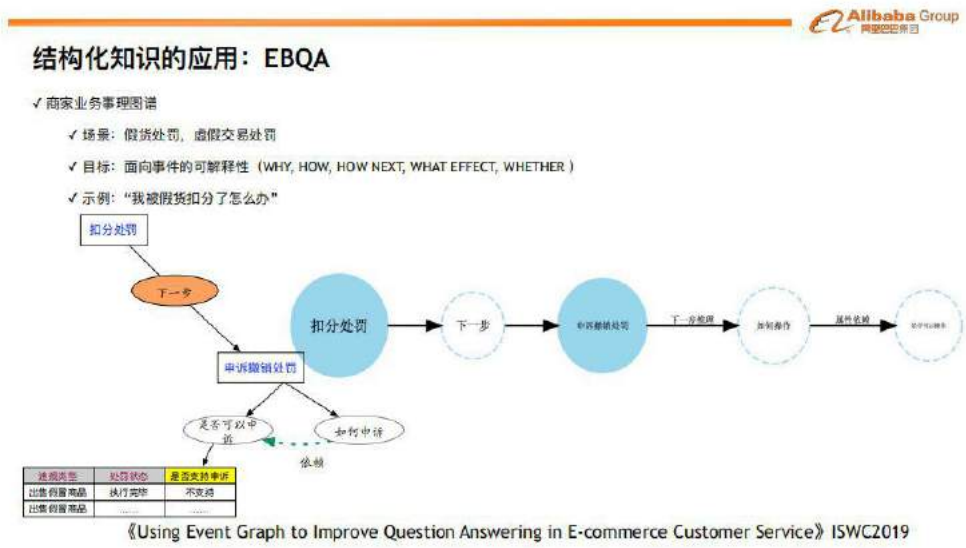
DataFunTalk

基于知识结构的推荐示例：

淘宝直播的问答示例：

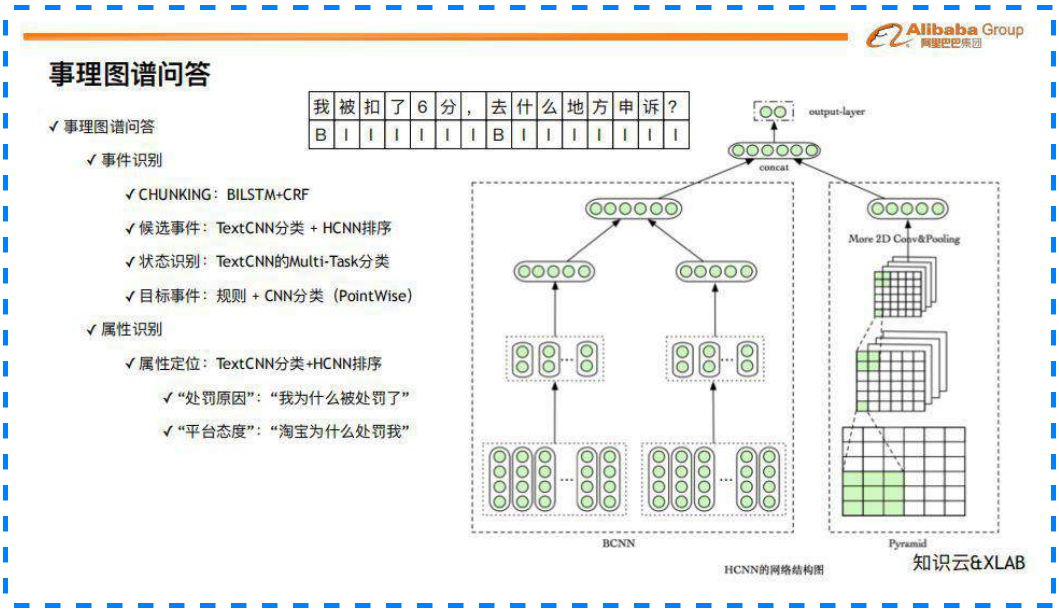
基于结构化知识的应用：EBQA

KBQA 着重解决实体类知识问答。在我们的实际场景中，还经常存在事件类的知识问答场景（EBQA，Event-Based Question Answering）。事件问答场景，针对用户的 WHY、HOW、HOW NEXT、WHETHER 等类别的问题，不仅仅需要精确的给出答案（能不能），通常还需要解释为什么（为什么能或者为什么不能）。这类问题，QA 体系也是很难有效支持的，除非把所有的情形及其组合进行枚举，但是这样做实际上是走不通的，必须通过对底层的知识结构进行改造。



上图的左边展示了结构化以后的知识表示，右边部分展示了 EBQA 的定位和推理过程。针对用户的问题 "我被假货扣分了怎么办"，我们首先定位出目标事件 "扣分处罚" 以及相应的属性 "如何处理"。在我们的知识体系中，从 "扣分处理" 我们可以依据 "下一步" 关系推理到关联事件 "申诉撤销处罚"，然后根据 "如何处理" 关联到该事件对应的属性 "如何申诉"。因为 "如何申诉" 依赖于 "是否可以申诉"，我们需要进一步的推理，并最终根据 CVT 表格来获悉是否支持申诉，如果支持的话需要同时给出申诉入口（"如何申诉" 的属性值答案），如果不支持的话需要根据 CVT 结构给出对应的解释。

需要注意的是，EBQA 不仅支持针对已经发生事件的问答，还支持假设性的事件问答（"如果被假货扣分了，我应该怎么办？"）。



EBQA 和 KBQA 的问答流程链路大体相似，采用基于约束图排序的方案，但是在事件识别和属性/关系识别部分存在一些差异。

❶ 事件识别

事件识别时，我们首先采用 BILSTM+CRF 模型对句子断句，然后采用 TextCNN 分类+HCNN 排序的方法给出候选事件，最后根据规则和 CNN 分类来确定目标事件。需要注意的是，在给出候选事件的时候，我们同时也给出了候选事件的状态，是否已经发生，来协助 CVT 中条件的判断。

❷ 属性识别

对属性识别，我们也采用了 TextCNN 分类+HCNN 排序的方案。EBQA 中的事件识别和属性识别之所以采用分类+排序的叠加方式，是因为事件相关的问题表述非常 subtle。例如，"我该怎么处理" 和 "我该怎么办"，前者对应 "如何处理" 这个属性，而后者对应 "下一步" 这个关系。

EBQA 的问答示例：

总结和展望

1. 知识结构化的思考

经过在实际业务场景中的探索和实践 ,我们对知识的结构化及其应用有了比较深入的思考和认知。一方面，知识结构化有不少可观的优点，可以准确回答、支持推理、可解释、可复用等等；与此同时，她也存在其局限性，主要的一点是 Schema 的构建需要业务专家参与而难以规模化。

❶ 适合场景

- 适合实体或者事件比较多的场景
- 需要精细化回答的场景
- 需要推理、解释的场景

❷ 优点

- 模式层知识结构稳定，知识可以传承和培训
- 支持精细化问答，推理，可解释，支持针对语义不清的推荐，可以更好的解决用户问题
- 支持多样化的展示方式，更好的用户体验
- 训练语料可高度复用，新增实体不需要重新训练，可以快速上线

❸ 局限

- Schema 成本：模式层构建成本较高（CVT），规模化有较大挑战

❖ 附加成本：熟悉的门槛较高，配套的工具体系需要完善

2. 展望和挑战

接下来，我们一方面致力于打磨算法和工具来降低业务知识图谱 Schema 的构建成本，另一方面会着眼于领域性常识知识图谱的构建，以商品和服务为核心，融合不同领域的常识性知识，以让小蜜能更好的理解和服务用户。这其中，低资源的挖掘，多模态的融合（文本，图片，视频），知识的有效表示和融入等问题是我们需要着力解决的重要技术挑战。

NLP 技术在微博 feed 流中的应用

嘉宾：董兴华　编辑：凌铭

导读：新浪微博截止 2019.9 统计的数据，月活跃用户数为 4.97 亿，日活跃用户数为 2.16 亿，其中约 94%为移动端用户，今天会和大家分享新浪微博在 feed 流中遇到的 NLP 问题和解决思路。主要包括：

❶ 难点与现存问题

❷ 标签系统

❸ 物料库

❹ 多任务、多模态探索

❺ 大规模预训练模型技术

——难点与现存问题——

❶ 博文内容大多比较短

DataFunTalk

第一个问题，微博的内容都是比较短的（ 一般都是 100 字符以内 ），比如右图中提到文本 "下午茶"，但是图片内容并不是美食 "下午茶"，考虑整个微博文本和图片内容应该分类到美女频道而不是美食频道更合适。另外对于短文本使用 LDA/PLSA 等 topic model 效果都不太好。

❷ 语言表达随意化

DataFunTalk

第二个问题，文本的随意化和口语化，语法结构不严谨，对于内容分析带来较大挑战。

❸ 用户搜索行为序列不能准确获取

第三个问题，比如：在微博搜索结果页中，由于文本简短，大部分的结果在页面中能看到全文，没必要点击进入看内容（ 除了第二条，需要点击展开全文 ），而从停留行为来看，由于一页展示多条博文，也不能准确定位用户感兴趣的是哪条微博。

❹ 用户 feed 行为序列不能准确获取

这个问题与搜索结果页相似，用户停留在 feed 流的页面中，不能准确区分哪条微博为用户感兴趣的。我们对点击较高的博文分析发现，点击较高的博文很多包含多张图片。由于一条微博可以包含多张图片，部分高点击的博文是由于用户想查看图片的内容才点击进入微博的正文页，但是这样的点击不能代表用户对该博文感兴趣。

综上所述，目前的微博场景很难获取十分准确的用户行为序列（ 用户行为序列包括展示页，点击，停留，转发，评论，赞，收藏等 ），导致使用 LDA/PLSA 主题的方式和用户行为序列方式建模效果都不太好。接下来，将为大家分享下我们的解决办法和思路。

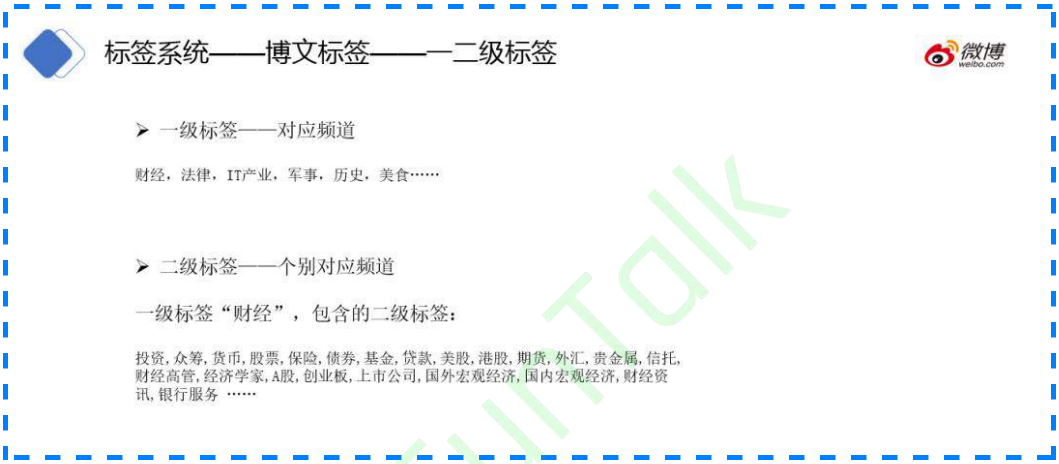
——标签系统——

标签系统主要包括：博文标签、用户兴趣（画像）标签、博主标签。今天主要介绍博文标签和用户兴趣（画像）标签。

❶ 博文标签

博文标签，主要分为：一二级标签、实体标签、关键词标签。

① 一二级标签

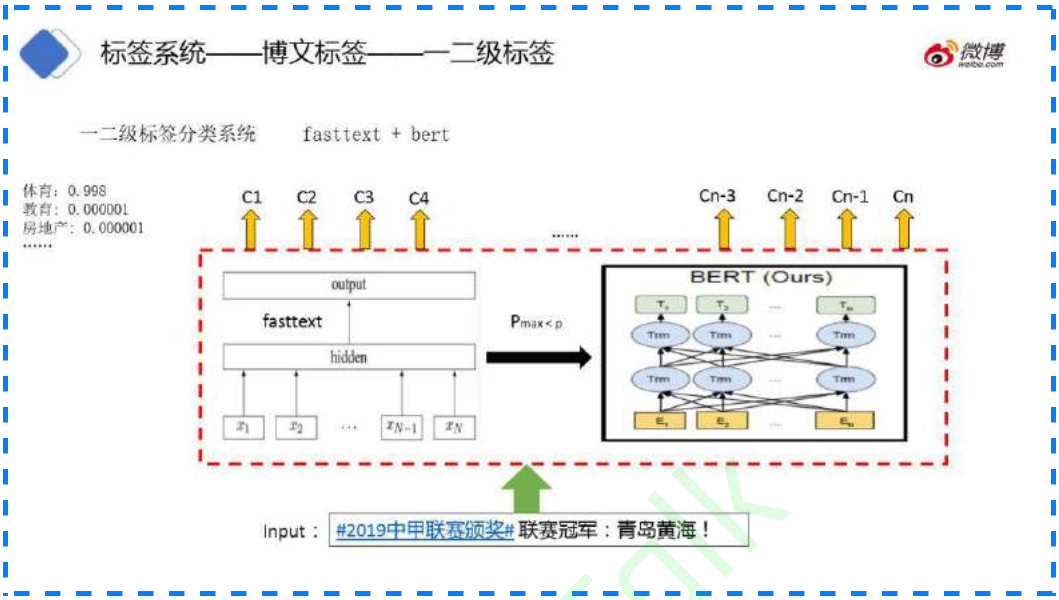


一级标签：对应频道，例如财经，法律，IT 产业，军事，历史，美食等标签。

二级标签：一级标签 "财经"，包含的二级标签：投资，众筹，货币，股票，保险，债券，基金，贷款，美股等。

一二级标签运用：一级标签与少量的二级标签和垂直频道对应，打上这部分标签的博文会分发到对应的频道下，对应的博文会在该频道进行展示。其次一级和二级标签也可用于画像的构建以及推荐中的召回和排序，但是作为标签，粒度太粗，不能很好的刻画用户兴趣。比如，有的用户只对英语感兴趣，如果把大量的教育相关的博文推荐给他，用户体验会比较差。

一二级标签分类系统：

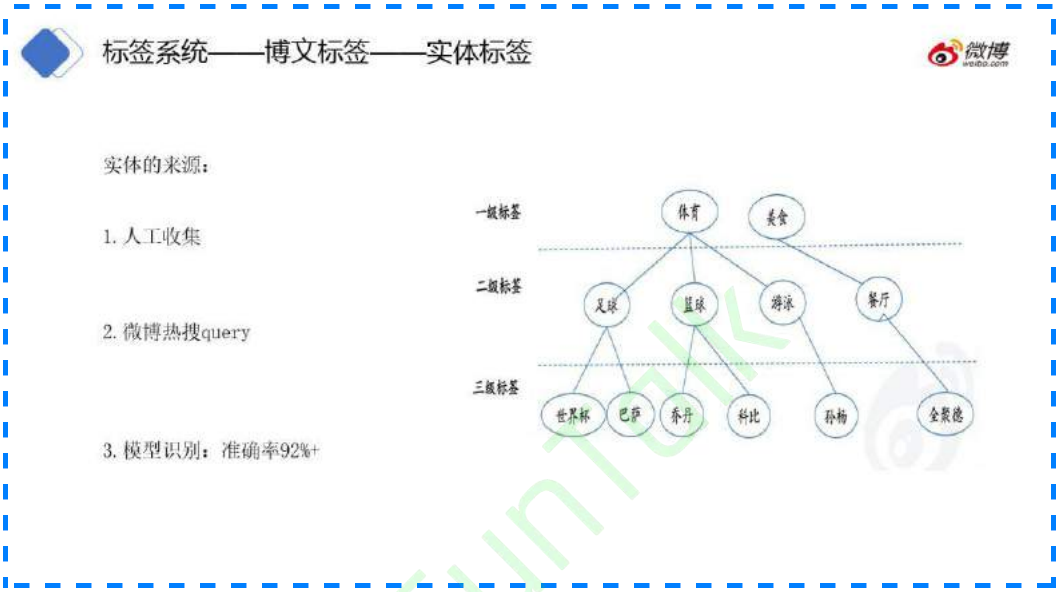


目前分类系统使用 fasttext+bert 结合的方案，是由于 bert 效果好，而 fasttext 性能好。我们有如下的方案：针对关注度高（ 明星 ）和高质量的博文使用 bert，其他的博文先过一遍 fasttext，若标签得分高于阈值（95

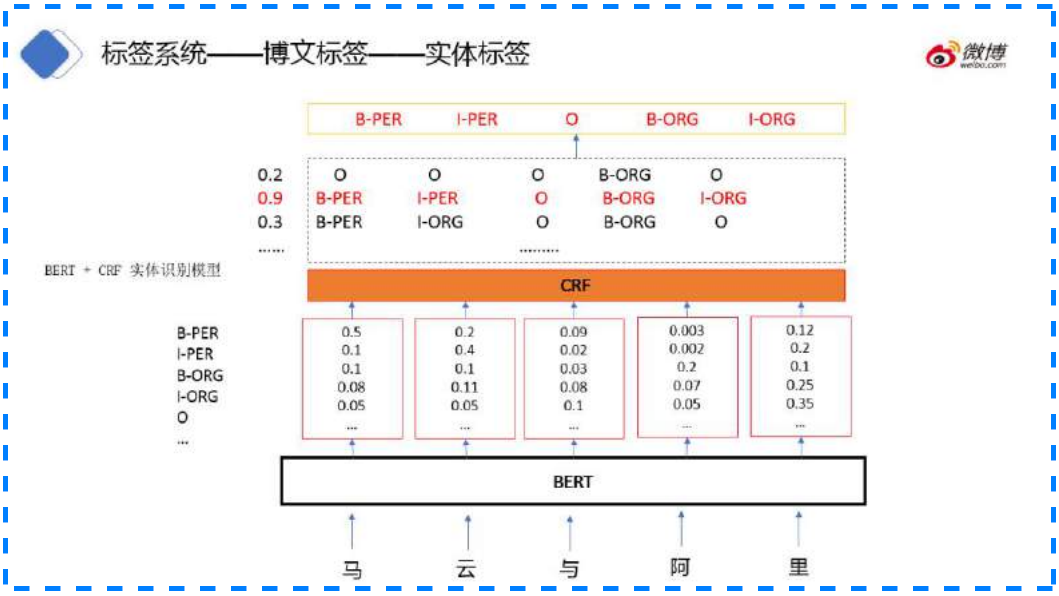
分 ）,则不再使用 bert 进行处理 ;若最高得分只有 70 分 ,那么使用 bert 再预测一次。 Bert 是一种多层的编码器，最近的研究表明，bert 不同层的 embedding 捕获到了不同的句子知识。比如底层 embedding 捕获到词法方面特征，中间层 embedding 捕获到句法特征，高层 embedding 捕获到语义特征。因此，我们对 bert 的结构做了优化，使用对各层 embedding 加权后获得的 embedding 作为整个博文的表征，后续的博文质量模型也是这种做法。

② 实体标签

实体标签在我们的场景中也叫三级标签，实体标签的来源：人工收集、微博热搜 query 和模型识别。如下图可看到一二级标签和实体标签之间的关系。



实体识别模型：



模型如图结构所示，首先经过 bert 层输出每个词对应每个序列标注的概率分布，再经过 crf 层输出最终的标注结果。而这些序列标注的训练数据是由人工标注的。

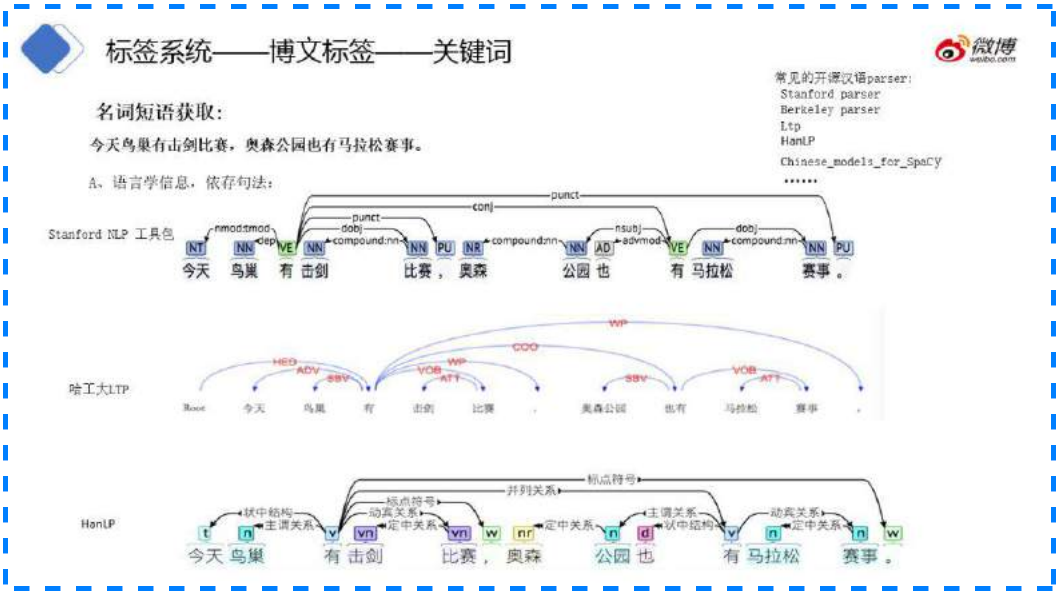
③ 关键词标签

关键词来源为两块：名词短语和用户 query。这是由于博文的文本较短所决定的（90%以上都是 100 个字符以内 ）,使用传统的主题模型方式并不奏效。

a. 名词短语抽取

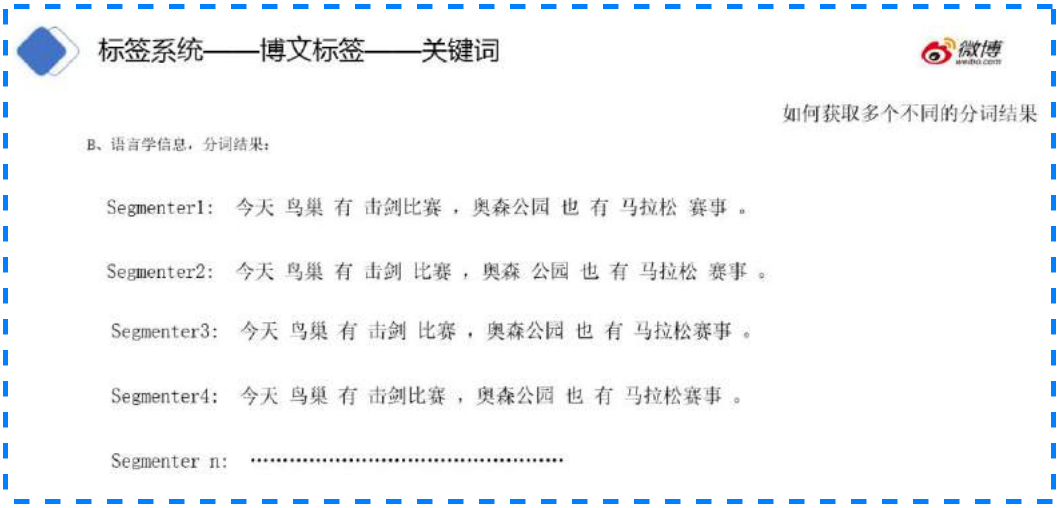
首先我们先从名词短语抽取关键词说起，我们可从某个句子中获取依存句法相关信息。

DataFunTalk



如上图所示，有 Stanford NLP、哈工大 LTP 和 HanLP 等工具包可以获取句子的依存句法，可以根据多个 parser 解析出来的依存句法对名词短语进行提取，例如，击剑、比赛两个词。首先在 stanford NLP 中击剑、比赛为一个名词短语且在一个语块（chunker）下，同样其他依存句法工具也是击剑、比赛都为名词短语并且都为同一个语块（chunker）。这样多个句法分析工具输出结果都是一致的，我们就认为击剑比赛可以作为标签使用。

DataFunTalk



其次也可以从不同的分词结果对名词短语进行提取和校验。通过下列方式可以获取多个不同的分词结果：

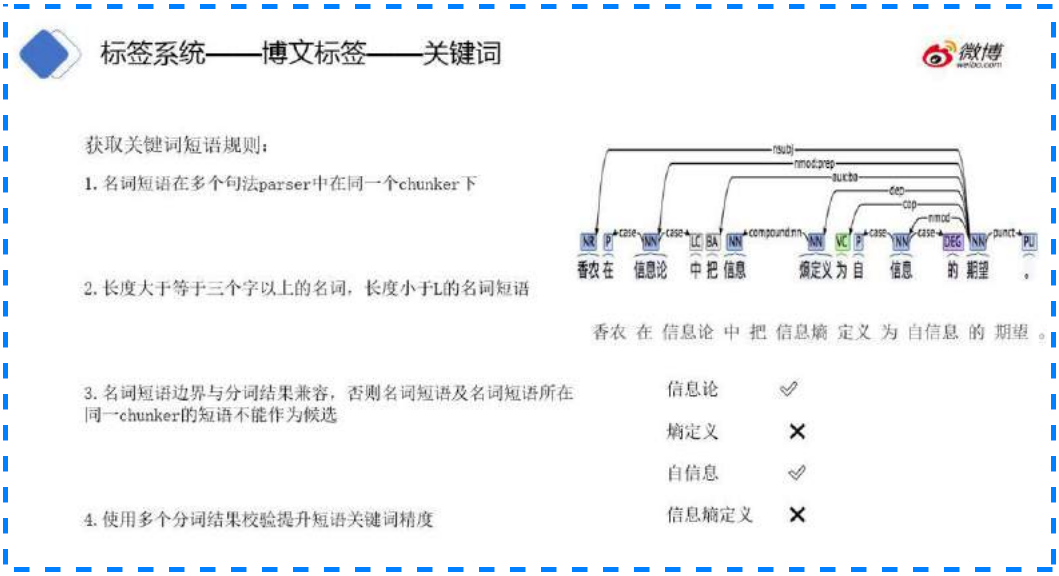
❶ 同一个分词器不同的分词粒度。

❷ 同一个分词器 nbest（ 其它可能 ）的结果。

❸ 不同的分词器。

使用不同的分词结果可以用于做名词短语提取正确性的校验。下文我们会详细阐述提取规则。

DataFunTalk



我们提取关键词短语有如下规则：

❶ 名词短语在多个句法 parser 中，在同一个 chunker 下，例如 "击剑比赛" 三个工具输出都是同一个 chunker。

❷ 长度大于等于三个字以上的名词 ,长度小于 L 的名词短语 ,不宜太长（ 实际操作中长度不超过 6)。

❸ 名词短语边界与分词结果兼容，否则名词短语及名词短语所在同一 chunker 的短语不能作为候选 。例如我们使用 Stanford NLP 依存句法结果与一个分词结果做兼容，依存句法结果：香农/在/信息论/中/把/信息/熵定义/为/自/信息/的/期望 ，分词器的结果：香农/在/信息论/中/把/信息熵/定义/为/自信息/的/期望。

可得到如下结论：

DataFunTalk

"信息论" 依存句法和分词都是同一个词说明兼容；

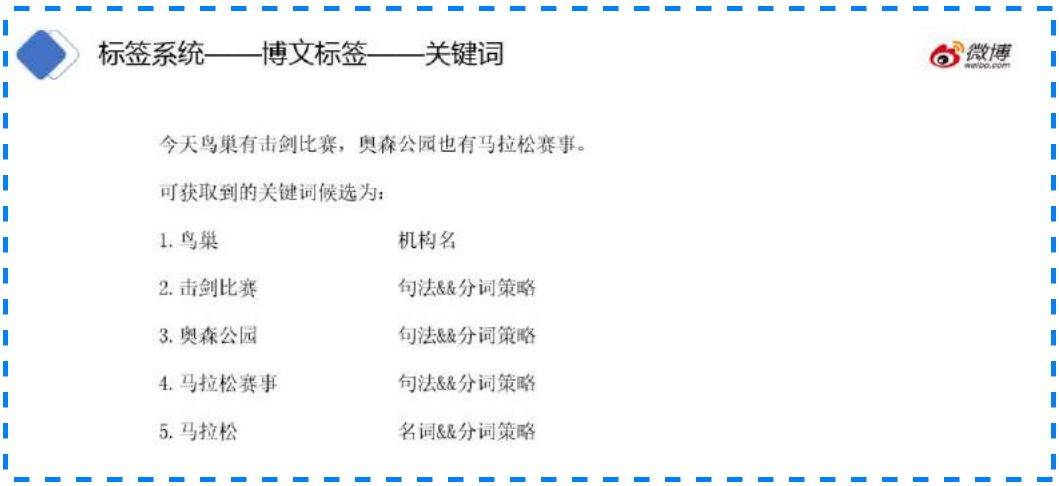
"信息熵定义" 依存句法分割为 "信息" 和 "熵定义",而分词结果为 "信息熵" 和 "定义",这两个切分的边界不一致。即依存句法边界在信息、熵的中间，而分词边界在熵、定义中间说明与分词不兼容。若依存句法为信息/熵/定义，那么熵、定义是拆开的，边界算是与分词保持一致，那么 "信息熵" 则与分词结果兼容；

"自信息" 依存句法为/自/信息/ ,分词结果为/自信息/ ,这里虽然依存句法自、信息被拆开了，但是与分词结果 "自信息" 是兼容的。

"信息熵定义"，依存句法结果为/信息/熵定义/，分词结果为/信息熵/定义/，因为分词结果/信息熵/定义/是拆开的，并且依存句法拆分不一致所以与分词结果不兼容，除非依存句法与分词结果拆分一致都为/信息熵/定义/，那么才说明与分词兼容。

❹ 使用多个分词结果校验提升关键词精度

DataFunTalk



最后，我们可根据以上所说的实体识别和名词短语规则将博文中词语提取出来做为关键词的候选。

b. 用户 query 提取

用户 query 作为关键词的条件：

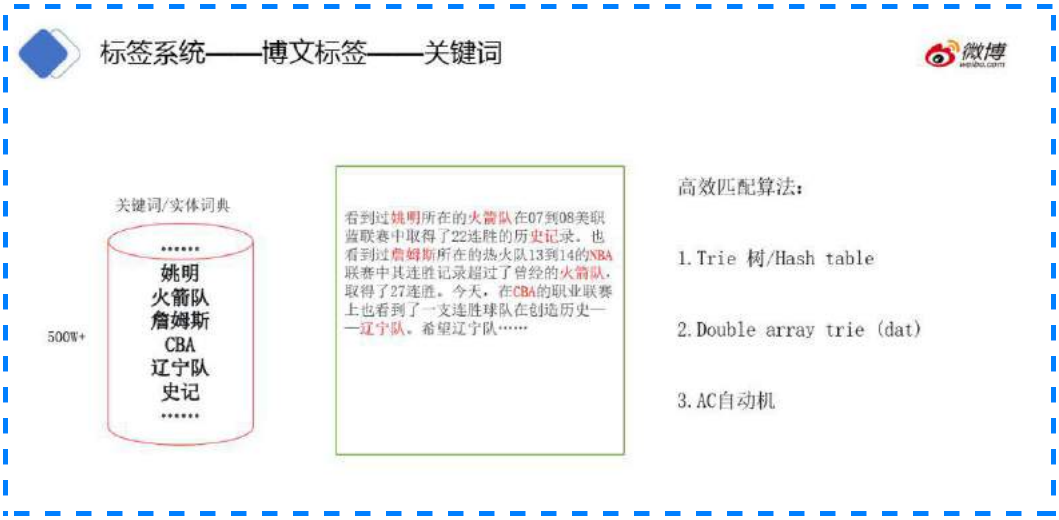
❶ Query 必须为高频

❷ 需要限制长度，不易太长

❸ 另外，query 会存在边界错误的情况。可使用新词发现的方式，通过左右信息熵和紧密度进行噪音过滤。

c. 匹配算法：

DataFunTalk

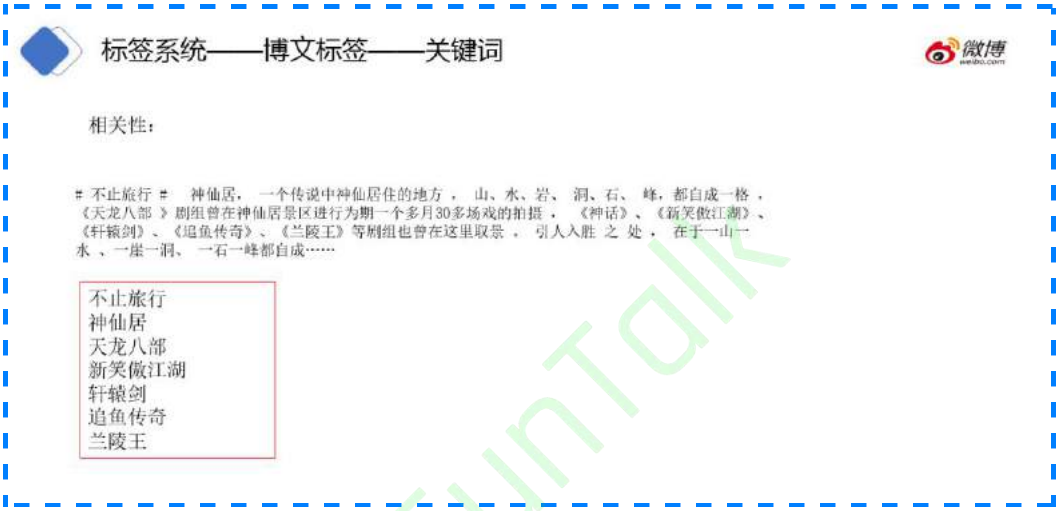


首先，我们可以使用高效匹配算法，Trie 树/Hash table、Double array trie (dat)、AC 自动机（三者中 AC 匹配效率最高），500w+ 的词典通过这些匹配算法，在博文上进行匹配。接着我们使用分词工具对文章进行分词，来校验分词的结果边界是否与匹配算法的结果兼容，输出最终兼容的结果。

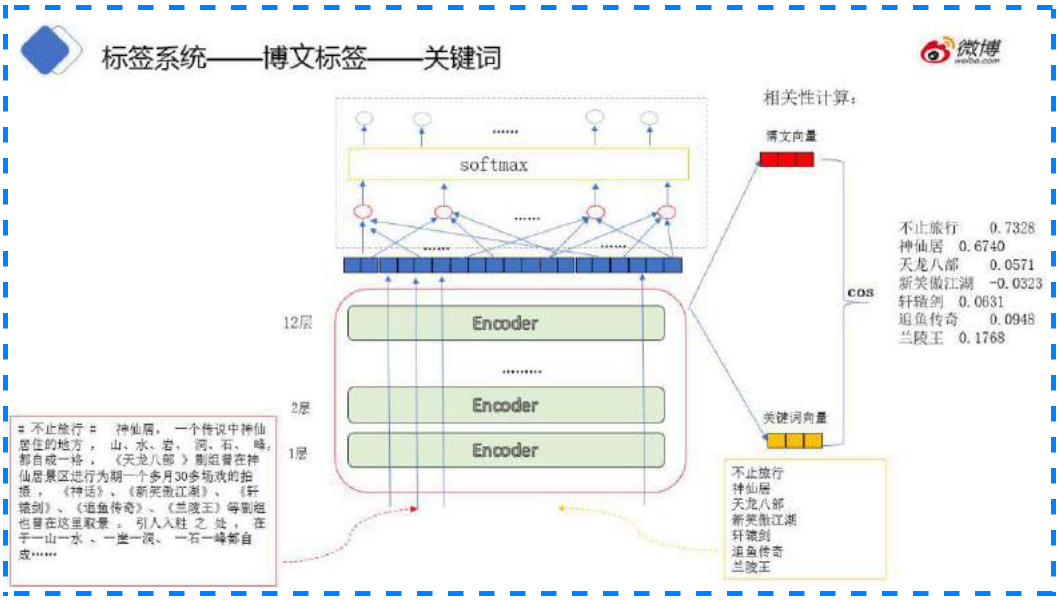
其实该方法与上文说的提取名词短语思路相似，例如：关键词为 "史记"，而文中为 "历史记录"。在使用匹配算法时会将 "历史记录" 中的 "史记" 匹配上，但是经过分词工具时，分出来历史/记录，而史记被分开，"史" 与 "历" 链接，没有单独成块，同时 "记" 也与 "录" 链接，那么可以判定与其不能兼容。

有的同学会问，如果把关键词/实体词典加入分词用户词典，是不是同时解决了匹配问题和消歧问题？其实这是很难做到的，因为一般的分词词典词汇量约 30-50 万，分词的过程本身就是对句子消歧的过程，在分词的过程中往往借助词频、词性等其它信息，关键词/实体词 500W+，远超分词词典中词数量，如果放入分词用户词典，这些词会在匹配之后，在分词结果中保证不会被切开（ 不同策略有所区别 ），将对分词的效果产生很大的负面影响。

d. 相关性



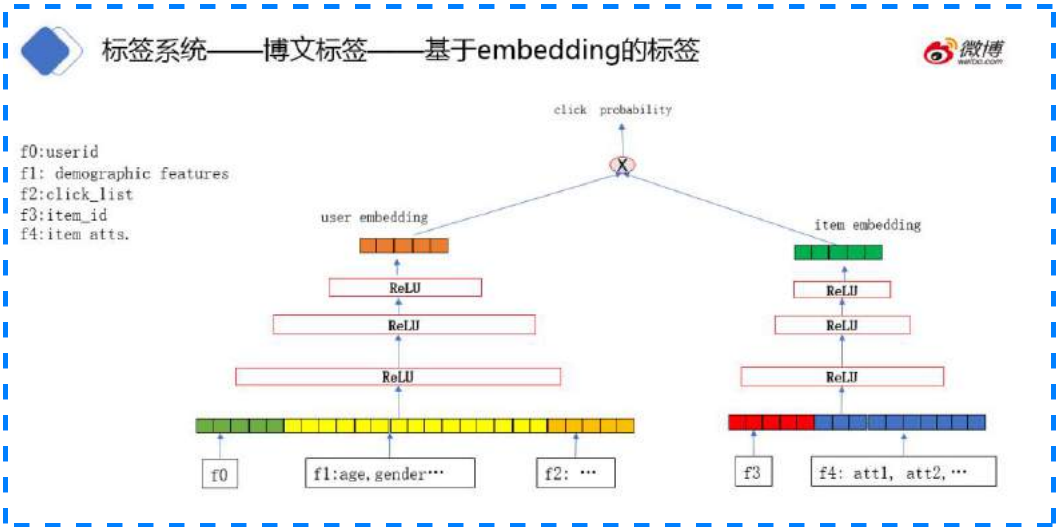
使用匹配算法和分词验证兼容性得到以上关键词，其中天龙八部、新笑傲江湖等是和电影相关的词，但是与博文中的主题相关性不一致（主题是旅游），由于关键词需要映射到用户画像上来统计用户兴趣，这些词不能代表用户的偏好。



标签与博文的相关性计算流程为：获取文本向量 -> 向量相似度计算 -> 设置阈值过滤标签。其中，我们使用 bert 获取文本向量，博文和标签（关键词）分别以字的形式输入至 bert 模型中，对模型最后一层的每个字相加求均值，效果最好（不同文本相似度计算任务向量取法有所区别，有的任务取最后一层 cls 的向量效果较好）。我们用于计算相似度的 bert 模型是基于博文的分类 fine-tuning 结果的模型。

除了标签还进行了同义词、近义词和上下文词的挖掘，这里不展开做过多阐述。

④ 博文 embedding 标签

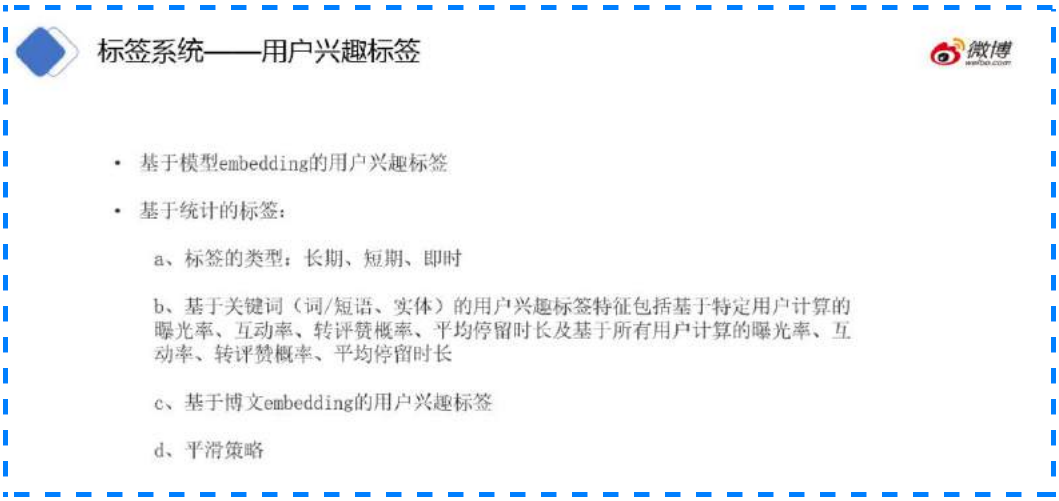


Embedding 标签通过用户的点击序列建模获取。模型如图所示，特征为 f0: 用户 id，f1: 用户的自身属性，f2: 用户点击列表，f3: 博文的 id，f4：博文自身属性，label 为是否点击博文。如前文所述，用户的行为序列不能十分精准的获取，所以 embedding 的效果并不好。

以上就是基于博文的一些打标签的方式，下面我们再来讨论一下基于用户方面的标签。

❷ 用户兴趣标签

DataFunTalk



① 首先是基于模型 embedding 的用户兴趣标签：上文中有介绍，此方法效果不太好。

② 基于统计的标签：

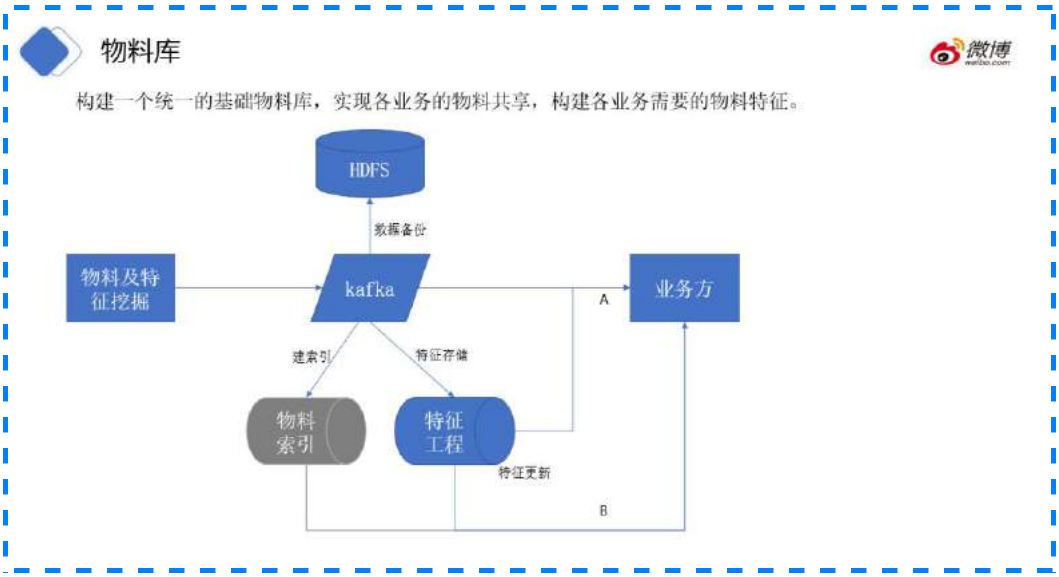
❶ 标签可以分成三种类型，分别为长期、短期和即时。分别进行统计，就有了不同类型的标签，可用于做召回或者排序模型的特征。

❷ 关键词用户兴趣标签的特征有以下两个方面：一是，关键词在特定的用户行为中的曝光率、互动率、转评赞率和平均停留时长；二是，关键词在全量用户行为中的曝光率、互动率、转评赞率和平均停留时长，用来表示在大众整体的偏好，可用于排序的输入特征。

❸ 将用户生成博文 embedding 映射到用户兴趣标签中。

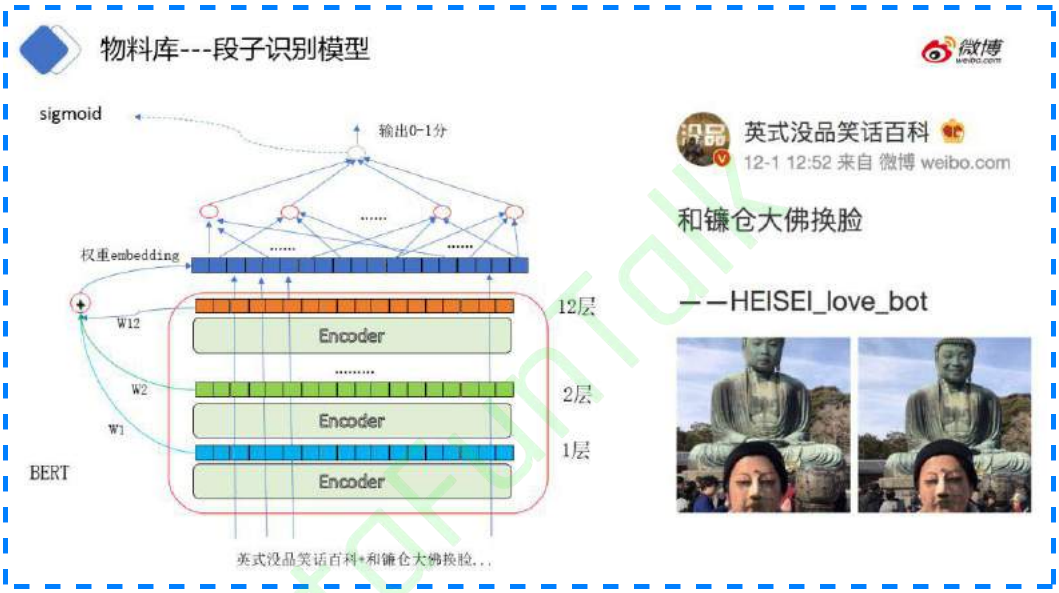
❹ 在以上计算用户兴趣的标签时，会用到一些平滑的策略。

——物料库——



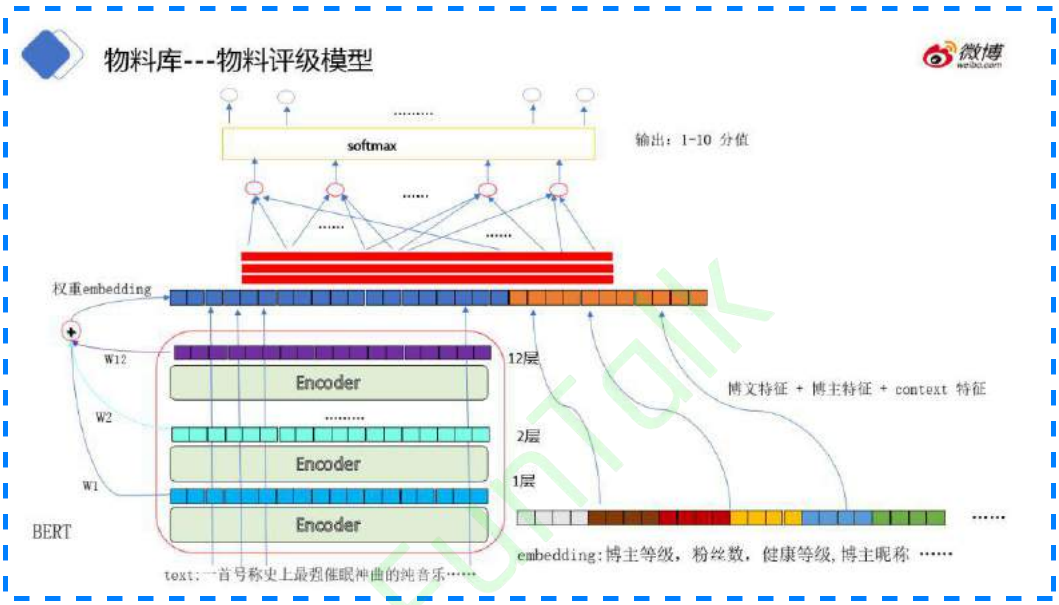
对用户发博的博文做预处理，最终构建一个统一的基础物料库，实现各业务的物料共享，构建各业务需要的物料特征。

❶ 段子识别模型



各层的权重在训练过程中优化获得。在权重 embedding 的基础上进行 reduce_mean 操作获取到最终的 embedding 表示 ,进行 sigmoid 操作获取到相应的概率分数。我们基于人工标注的数据进行 bert 的 fine-turning。

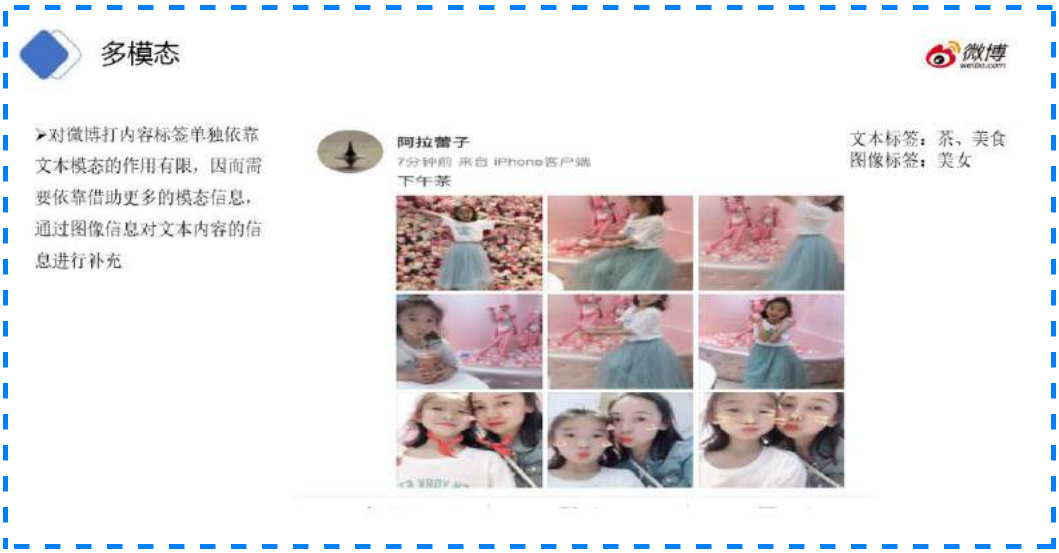
❷ 物料评级模型



物料评级模型分成两块：一块是博文的内容，使用 bert 进行文本向量的抽取；另一块为博主的特征和上下文特征（ 博主发博的上下文 ），比如博主等级，粉丝数，健康等级，博主昵称等，把相关特征拼接在一起，进行多分类。

——多模态——

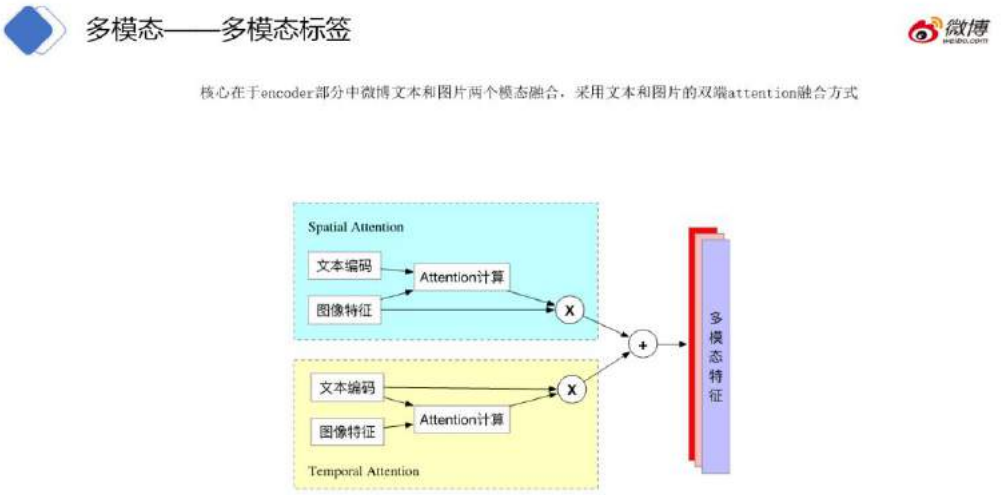
上文也介绍过，对于文本较短的博文，我们没法获取太多的信息，这样需要借助多模态信息。



比如通过图像信息对文本内容的信息进行补充。那么通过该图像模态，我们可以打上美女这样的标签。

博文中出现了苹果，可能指代吃的苹果或者苹果公司。若借助图像可一目了然去除歧义。

❶ 多模态对偶模型

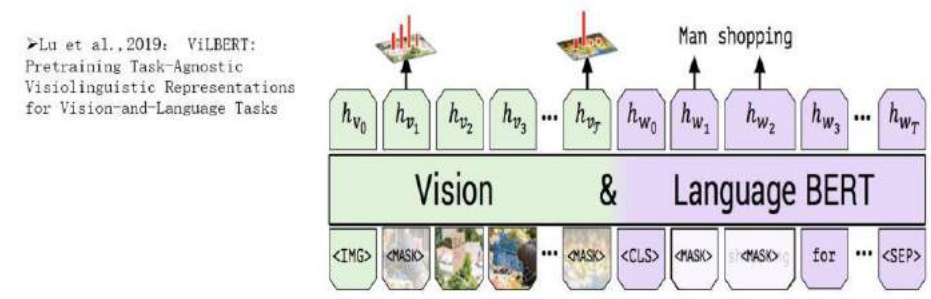


上图是在 BERT 发布之前的做法 :多模态对偶模型 ,其中文本编码使用 lstm 模块，图像编码则使用 Inception-Resnet-V2 模块。两者编码对齐到同一个长度进行 attention-both 计算，之后两个模块分别与文本编码和图像编码进行加权计算，然后向量 add 输出多模态向量，最后解码进行标签分类。Attention-both 可以这样理解：由于 image、文本都编码为语义向量，单独的使用文本进行解码时，可以利用图片的 attention 信息，而单独使用图片解码时，可以利用到文本的 attention 信息，Attention-both 就是这两者的结合。模型效果相对于文本模型有一定的提升，但是性能会比较慢。

❷ BERT 多模态模型

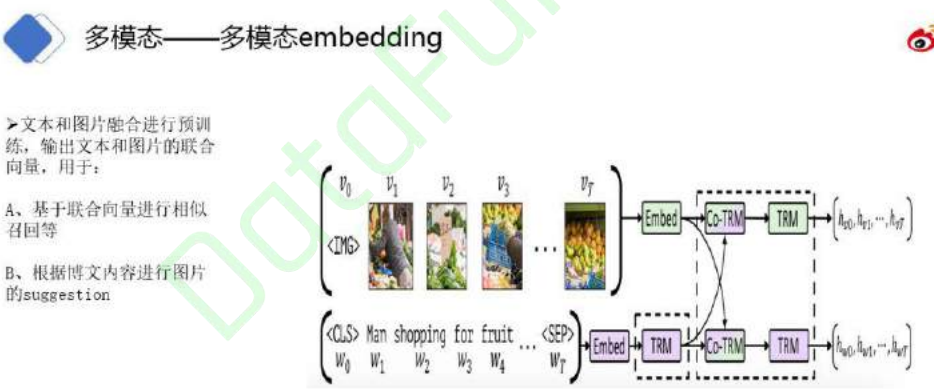
① 预训练模型

BERT 出来之后，微软发表了图像结合 BERT 的多模态预训练文章。



ViLBERT 预训练

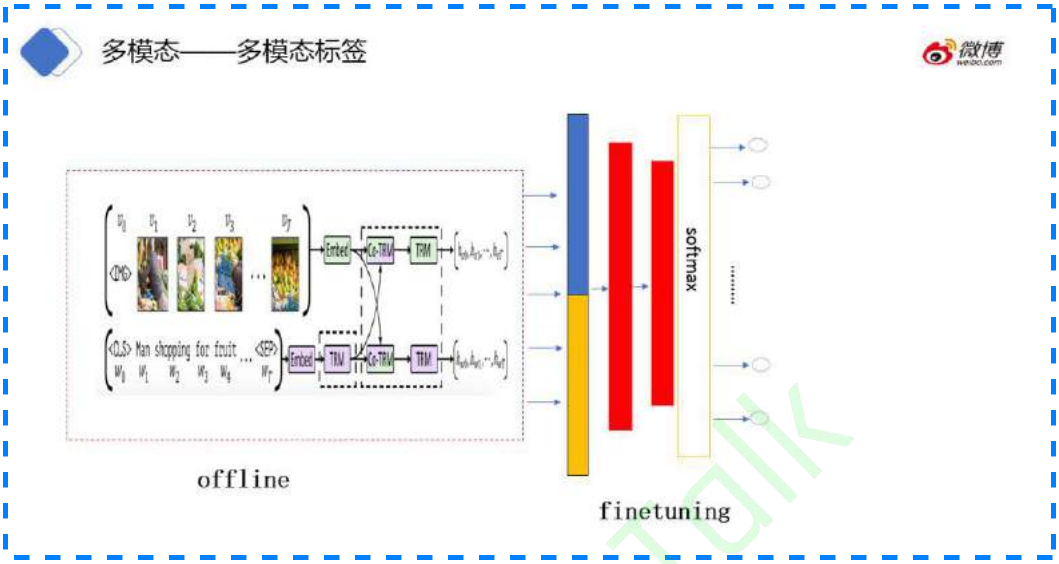
其预训练除了文本还加入了图像的部分。首先把完整的图像做切分（Fast RCNN），每个为一个个小块图片，并且做 mask 输入到模型中进行预测还原。



图像、文本co-attentional transformer

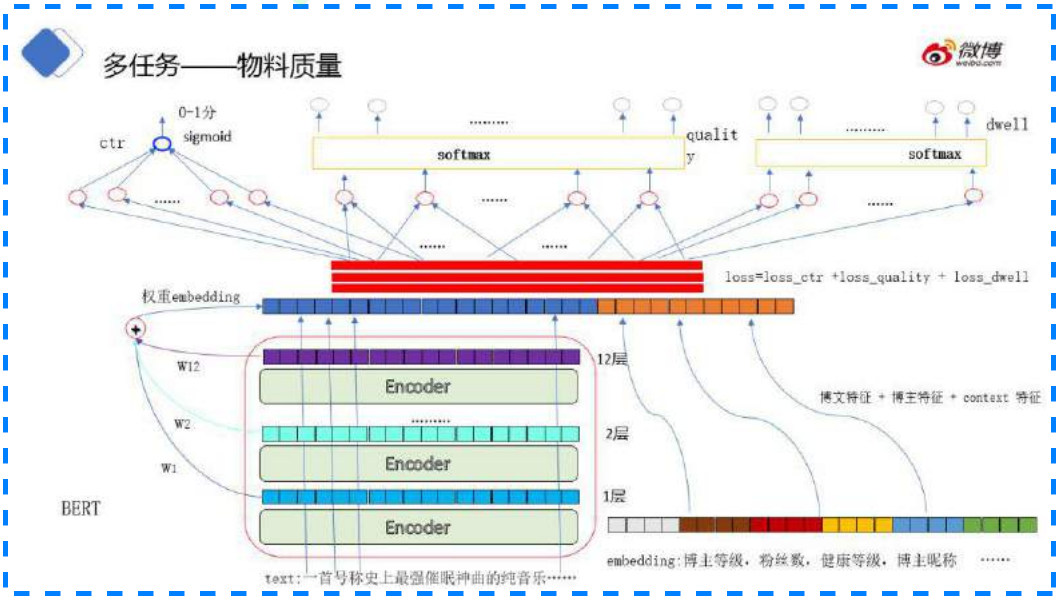
文本和图片的向量融合再进行预训练，输出文本和图片的联合向量 。联合向量可用于相似博文的召回和根据博文内容来检索相似图片（ 自动配图 ）。

② 多模态打标签



我们也尝试在多模态预训练之后做打标签的任务，预训练的向量之后进行多分类的 finetuning。

——多任务——



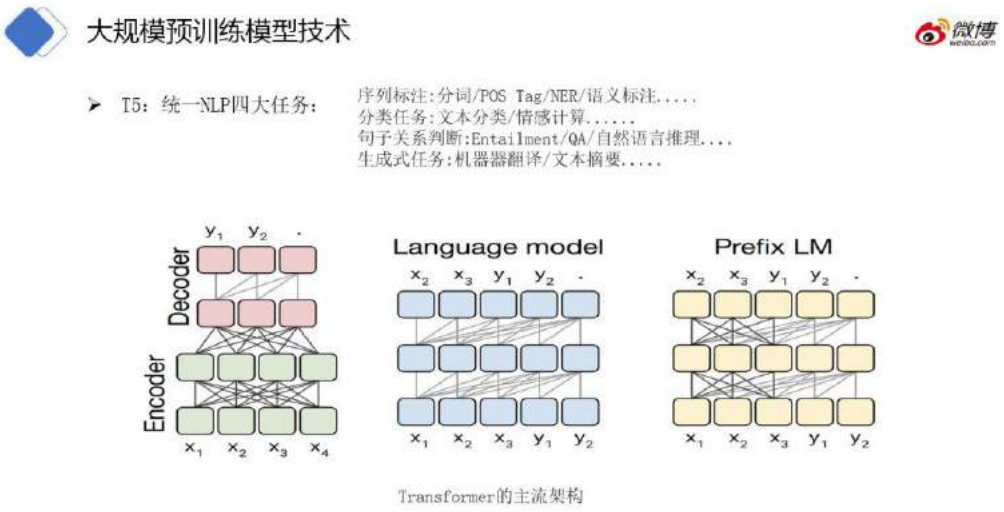
对比之前版本的质量模型，多任务新增了 ctr 和停留时长的任务。整体效果会有些提升。

——大规模预训练模型技术——

有以下两种模型：BERT 和 GPT2，而这两种模型是有区别的。BERT 主要做编码，对文本进行理解。GPT2 则为做生成任务，可辅助写作。杜则尧同学开源了基于中文的 GPT2 模型：

https://github.com/Morizeyao/GPT2-Chinese

感兴趣可以具体了解。



除了 BERT 和 GPT2，还有 T5。T5 整合了 NLP 的四大任务：序列标注 ，分类任务 ，句子关系判断 ，生成式任务。

我们目前在大规模预训练模型技术，做了以下几块工作：

❶ 实现了 GPT2 模型的训练和推理

❷ 同时也实现了 T5 模型训练和推理

❸ 由于模型参数太多，性能太慢，我们尝试了模型的蒸馏、量化和 Tensor RT 以便提升性能

❹ 由于 T5 训练时是大杂烩，把所有任务和内容放在一块训练，我们认为把更相似的、可以相互影响的任务一起训练效果很更好

除此之外还孵化了以下几个项目：

❶ 机器翻译

❷ 复述系统

❸ 人工辅助写作系统

❹ 拼写纠错、语法检查和用词润色

——总结——

今天主要讲了新浪微博在 feed 流中遇到的 NLP 问题和解决思路。我们内部正与产品、架构团队积极沟通，以获取高精度的用户行为序列用于用户行为的分析和建模。由于博文较短，我们没有采用主流 LDA/PLSA 技术解决相关问题，而采用了更为传统的 NLP 技术和一些 trick 完成了实体/关键词的挖掘和相关性计算。

从技术来讲，随着预训练模型的出现，NLP 最近这几年取得了实质性的进展，但不可忽略的是：传统的 NLP 技术在某些场景下依然发挥着重要的作用。另外，一个技术趋势是在解决相关问题时逐渐从单一的技术走向多任务、多模态、多语言的融合。

机器学习篇

阿里新零售中的智能补货（Ⅰ）— 库存模型

作者：阿里零售通算法团队 编辑：HohXil

零售通作为阿里巴巴新零售的八路大军之一，肩负着“共建智能分销平台”和“让百万小店拥抱 DT 时代”的重要使命。一方面，我们通过线上平台（零售通 APP）将零售品牌商的货品展现给小店的店主，并提供交易渠道让店主进行批发进货；另一方面，我们通过天猫小店和如意 POS 对小店进行赋能，帮助他们在线下的销售中能迅速发现 C 端消费者的需求，提升小店的坪效。在这种 B2B2C 的创新分销模式中，智能化的供应链体系不仅要保证平台能持续稳定的为小店进行供货，还要控制供应链成本，让数字化渠道成为最经济的分销方式。

我们在不断迎接新零售模式下供应链各种挑战的过程中，逐渐沉淀出一套利用大数据和智能算法解决供应链补货的方法论，在这里与大家分享。本系列分为两部分：第一部分主要介绍智能补货的库存模型；第二部分会详细介绍补货模型中的销量预测算法以及我们在预测中用的一些经验和方法论。本篇将对第一部分，即智能补货的库存模型进行介绍。

为什么需要智能补货：

零售通作为连接商家、小店和 C 端消费者的数字化分销平台，保障货品供应的持续性和稳定性成为供应链的首要目标之一。不同于天猫、淘宝等直接面向 C 端消费者的平台，零售通直接触达的用户是小店的店主。他们在平台上

进行采购的主要目的之一是用于补货。如果此时需要补充的货品在平台上已售罄，这不仅会损失平台的成交额(GMV)，也造成了店内的货品供应不足，导致小店的收入降低。另一方面，如果为了最大化保障供应率，而将一大批货品囤积在零售通的仓库中进行售卖，则有可能导致某些货品长时间卖不出去，增加了库存周转时间和滞销率，降低了供应的效率。

因此，供应链补货的目标是通过供应方式（供应时间，供应数量，供应周期等）的决策达到仓库中需求和供给的平衡，使得仓库中货品的库存既可以最大化满足用户的需求，也能将库存周转时间控制在一定范围之内，保证供应的效率。（因此在很多行业中，供应链补货又叫做“库存平衡”。）

库存平衡

在某些传统行业中，供应链的补货任务大部分还是通过人工完成。一方面，传统行业的渠道信息数字化大多构建得不完善，无法完全通过数据实现精准补货；另一方面，传统行业的计算能力有限，在求解某些优化问题中无法实现大规模的复杂计算。而零售通背靠阿里巴巴集团这颗参天大树，不仅在数据获取渠道上突破了线下渠道数据的限制，还在计算资源上得到了充分的保

证。这些优势为零售通供应链在智能补货算法的应用和发展提供了成长的土壤，使得通过基于机器学习和运筹学构建的智能补货算法体系成为可能。

库存理论（inventory theory）：

无论是传统零售还是新零售，货品都会经历从生产制造到分销、再到消费整个周转过程。分销中的每个环节都需要维持一定货品库存来应对需求的不确定性。因此，分销商或者零售通都需要解决以下三个问题并作出正确的决策：

1. 未来某段时间的需求是多少？
2. 需要为未来的需求准备多少货？
3. 每天的库存水位是多少？

无论作出何种决策，这三个问题都可以用**“进-销-存”模型**来描述。

"进-销-存"模型

该模型需要进行连续决策，即以上三个问题不是一次性决策，而需要周而复始的不断进行下去。另外，上一次的决策结果（补货数量）会影响下一次决策状态（当前库存）。用数学可以对此过程进行如下描述：

$$x_{t+1} = x_t + u_t - w_t$$
$$w_t \geq 0$$

其中 x_t 是 t 时刻的货品在库数量，u_t是需要进行决策的 t 时刻的补货量，w_t 是 t 至 $t+1$时刻的需求量总和。 t 到 $t+1$是两次决策的时间间隔，取决于补货的具体形式。

在“进-销-存”过程中，会发生成本的支出。进货时会产生订购和运输成本；货品储存在仓库中会产生存储成本(比如仓租和管理成本) ;货品售卖后会产生配送成本。

t 时刻的总成本可以用c_t来表示，它是进货量、在库数量和需求量的函数 $c(x_t,u_t,w_t)$。供应链补货所采取的决策目标就是**在满足需求的情况下，最小化过程中产生的总成本。**

$$\hat{U}_T = \{\hat{u}_t\} = \arg\min_{U_t} \sum_{t=1}^{T} c_t$$

在运筹学中，该模型被形式化，并扩充为**库存理论**(Inventory theory)，作为该学科的一个分支。库存理论在建立后的近 80 年中不断演化发展，为各行各业的生产制造和库存存储提供指导，发挥着十分重要的作用。有兴趣的读者可以进一步阅读[1][2]，以获得库存理论更多的细节。

安全库存模型(safety stock)：

上面的库存理论讲到补货的决策目标是在满足需求的情况下，最小化供应成本。作为模型的一种简化，我们先忽略成本因素，仅考虑如何决策才能满足用户的需求。

显而易见，如果不考虑成本，最佳的供应方式就是“用户买多少就进多少货”。但现实中进货总是发生在用户购买之前的，即“先进货，再售卖”。因此决定当前进多少货就变成了对未来的需求量预测。从统计学的角度，未来某段时间的需求服从一定的统计分布：

$$w_t \sim p_t$$

对模型作进一步的简化，我们假设需求服从的分布是正态分布：

$$w_t \sim N(m_t, \sigma_t^2)$$

其中均值m_t和标准差σ_t是待确定的参数。根据“进-销-存”模型，当给定 t 时刻的库存时，$t+1$时刻的库存x_{t+1}也服从正态分布：

$$x_{t+1} \sim N(x_t + u_t - m_t, \sigma_t^2)$$

如果$x_{t+1} \geq 0$，说明 t 到 $t+1$时刻的需求能被充分满足；反之则会发生缺货，导致成交额的损失。

从正态分布的性质我们知道，无论我们如何增大补货量，都无法 100%的保障用户的需求，因为正态分布分布的最大值是∞。（当然现实中不可能发生需求∞的情况，但是有可能是一个很大的数值，比如双 11 促销时的需求。）这里我们引入一个名词叫做**服务水平**(service level - sl)，它表明需求有多大的概率可以被满足。对于单个货品来说，服务水平 = 1- 缺货概率；而对于多个货品，服务水平 = 1 - 缺货率。通常我们所说“将缺货率控制在 5%”，

即相当于“将服务水平提升至 95%”以上。对于服从正态分布的x_{t+1}来说，我们可以将服务水平通过积分表达出来：

$$sl_t = P(x_{t+1} \geq 0) = \int_0^{\infty} N_y(x_t + u_t - m_t, \sigma_t^2)\, dy$$

等式右边的积分可以用误差函数进一步表示：

$$sl_t = \frac{1}{2}\left[1 + erf(\frac{\mu_t + x_t - m_t}{\sigma_t\sqrt{2}})\right]$$

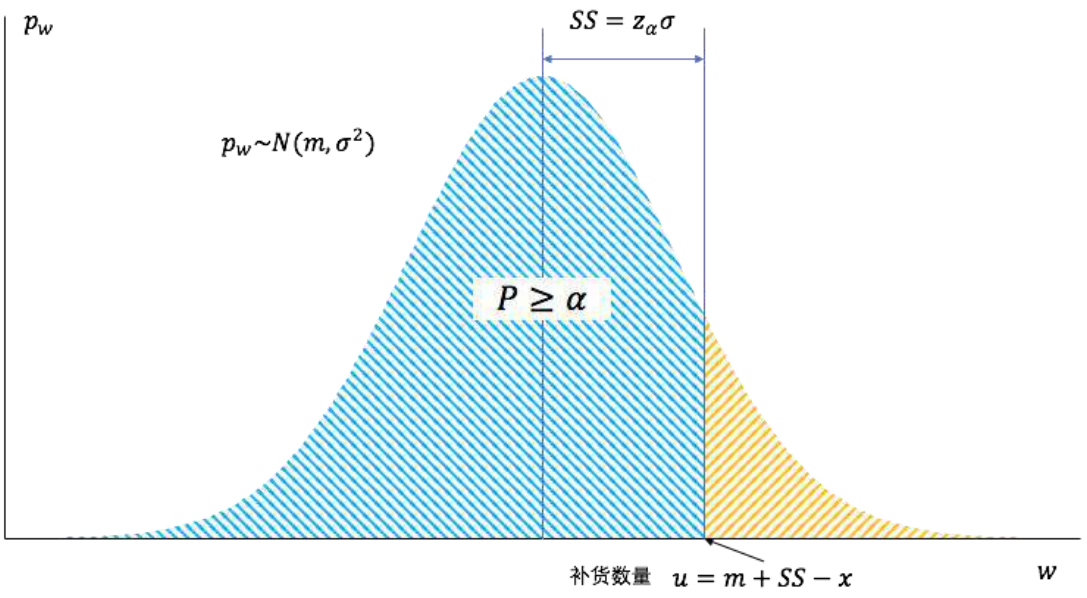
如果要保证服务水平大于 95%，则需要误差函数满足：

$$erf(\frac{\mu_t + x_t - m_t}{\sigma_t\sqrt{2}}) >= 0.9$$

通过查表可得：

$$\mu_t \geq m_t - x_t + 1.16\sqrt{2}\sigma_t$$

上面不等式的右边部分就是在 t 时刻满足 95%服务水平的最小补货量。



安全库存模型

不难看出，为了满足较高的服务水平，我们不仅需要补足正常需求量（这里叫做需求量的均值），还要考虑需求波动带来的不确定性。为保证需求波动所额外准备的库存，就叫做**安全库存**(safety stock - SS)[3]。一般来说，安全库存可以表示为以下形式[4]：

$$SS = z_\alpha \sigma_t$$

其中z值对应了α服务水平下的**显著系数**，服务水平越高，z值越大。**安全库存仅与服务水平和需求的标准差有关，与需求本身的数量无关。**

现实中的限制：

安全库存模型仅考虑了为满足需求而求解得到的补货量最小值，但现实情况比该模型复杂得多。首先需要指出当补货订单下发后，货品不会马上入仓进行销售。在这期间供应商需要及时作出响应，并进行生产备货。在供应商发

货后，货品还需要通过物流到达目的仓库，在这期间也会产生时间的延迟。我们可以定义补货订单下发到货品到仓的总体时间延迟为**供应商交付时间**(vendor leading time-VLT)。该交付时间我们我可以与供应商协商，保证交付时间小于一定的天数，也可以根据该供应商的历史表现统计得到。在第一种情况下，我们可以认为 VLT 是一个定值，那么安全库存模型得到的最小补货量只需要做稍微的改变：

$$\mu_t \geq \bar{m}_t - x_t + z_\alpha \bar{\sigma}_t$$

其中需求的均值和标准差只需要改变为“$t+1$时刻再向前数 VLT 时间至 t 时刻”的需求分布均值$\bar{m}_t$和标准差$\bar{\sigma}_t$。后一种情况稍微复杂一些，因为根据历史统计得到的 VLT 也是一个分布。可以证明，在单位时间的需求分布不变且需求和 VLT 分别独立的服从正态分布的情况下，最小补货量可以表示为[5]：

$$\mu_t \geq (\Delta t + E(VLT))E(w) - x_t + SS_t$$
$$SS_t = z_\alpha \sqrt{E(L)\sigma_w^2 + E(w)^2 \sigma_L^2}$$

其中$\mathrm{E}(w)$和σ_w分别是单位时间需求的均值和标准差，$\mathrm{E}(L)$和σ_L分别是 VLT 的均值和标准差。

除了 VLT 之外，成本也是安全库存模型没有考虑的因素。成本分为物流成本和库存成本。在需求一定的情况下，如果订货太多，则会提升库存周转天数，增加库存成本。如果订货太少，则在供应商配送过程中货品装不满一车，导致物流成本率的增加。库存理论中有一种**“经济订购批量”模型**(economic

order quantity - EOQ)[6]来最优化订货量，使得总成本最低。在零售通的供应链模式中，我们主要通过**库存周转天数**和**最小起订量**(minimum order quantity - MOQ)来近似表达成本的限制。

在零售通平台上，库存周转天数是指货品从入仓到销售之间存储在仓库中的天数。某个单品的库存周转天数越长，说明该单品所消耗的库存成本越高。最小起订量是一个补货订单中包含货品金额的最小值。如果订单中的补货量小于最小起订量，说明供应商的物流成本过高，不会进行发货供应。最小起订量是对一个供应商下的多个货品而言的，因为某一次补货订单可以包含多个单品的补货量。

除了考虑成本的因素，我们还需要考虑仓库收货的产能。仓库的收货发货需要人工作业，每天处理的订单量是有限的。如果某次补货的数量过大，导致仓库无法及时收货，则有可能导致货品无法入仓而发生损坏。另一种限制是“整箱补货”，因为在销售上可以零售，但供应上只能整箱批发。补货模型需要作出相应的调整来满足以上种种限制。

这一系列现实中的限制条件都使得一个简单的需求平衡问题变得复杂化。事实上，这里仅列出了零售通平台上需要考虑的主要限制。在不同的场景下(比如双 11 大促)，限制条件将会变得更多更复杂。这导致了很多现实问题无法被数学形式化和模型化，从而给不出算法最优解。这也是新零售场景中供应链补货算法需要迎接的挑战和解决的难题。

整数线性规划(ILP)：

一个简单的补货模型

基于上述限制条件，我们可以基于安全库存模型进一步改进补货模型。首先我们应该定义一个优化的目标，比如“最小化整体库存周转天数”。优化该目标需要满足以下限制：

1）保证每个货品的服务水平大于α；

2）供应商的最小起订量大于元Q；

3）订单中补货总量不超过仓库产能件K。

其它的限制暂时不考虑，因为它们并不影响我们对该问题的阐述。但其中有一个隐含的限制需要说明，就是即使不用整箱补货，每个单品也只能“一件一件的补”，比如不可能补 1.5 个货品。

下面我们将用数学形式化的表达优化目标和每一个限制。首先对一些变量和下标进行定义：

u_i	表示货品i的补货量，是模型中待决策的变量
x_i	表示货品i的当前库存
m_i, σ_i	分别表示当前至下一个到货日期货品i需求分布的均值和标准差
p_i	表示货品i的单价(元)

限制 1：每个货品的服务水平大于α(安全库存模型)：

$$\mu_i \geq m_i - x_i + z_\alpha \sigma_i \quad \forall i \in I$$

限制 2: 供应商的最小起订量大于 Q 元：

$$\sum_{i=1}^{I} u_i p_i \geq Q$$

限制 3: 订单中补货总量不超过仓库产能 K 件：

$$\sum_{i=i}^{I} u_i \leq K$$

限制 4: 单品的补货数量大于 0 且为整数

$$u_i \geq 0$$
$$u_i \in Z \quad \forall i \in I$$

优化目标：最小化整体库存周转天数。（因为单品库存周转天数是需求的函数，如果需求随着时间变化，则无法将其表示为线性的形式。这里为了简单起见，假设每天的需求分布不变。）

$$\min \sum_{i=1}^{I} \frac{u_i}{m_i}$$

优化目标和限制条件均为关于 u_t 的线性表达式，加上 u_t 必须是整数的限制，可以很容易看出该问题是一个**整数线性规划**(integer linear programming - ILP)[7]，可以通过很多算法工具包（比如 cplex，gurobi 等）进行求解。上面的公式写成更加紧凑的形式为：

$$\min \sum_{i=1}^{I} \frac{u_i}{m_i}$$

$$s.t. \begin{cases} \mu_i \geq m_i - x_i + z_\alpha \sigma_i & \forall i \in I \\ \sum_{i=1}^{I} u_i p_i \geq Q \\ \sum_{i=i}^{I} u_i \leq K \\ u_i \geq 0 \\ u_i \in Z & \forall i \in I \end{cases}$$

该整数规划模型虽然是对复杂的真实环境进行了简单的抽象，但它却反映了供应链补货的本质。我们可以形象的描述为“在钢丝上跳舞”。首先我们的决策不是自由的，它需要在各种“条条框框”（比如 MOQ、产能、整箱补货等）中进行；其次补货时不能“用力过猛”，补得太多会导致库存周转天数上升，成本增高；也不能用力过轻，补得太少会导致需求无法被充分满足，导致成交额的损失。而整数规划模型正是描述了这种“恰到好处”的决策。而这种恰到好处不仅需要我们深入的理解新零售业务，并对此进行模型抽象，也需要我们对未来的需求以及需求的波动性进行准确的预测。下一期我们将着重讨论如何预测未来的需求分布，以及大数据和智能算法在其中的应用。

参考文献：

1. Zipkin Paul H., Foundations of Inventory Management, Boston: McGraw Hill, 2000, ISBN 0-256-11379-3

2. Porteus, Evan L. Foundations of Stochastic Inventory Theory. Stanford, CA: Stanford University Press, 2002. ISBN 0-8047-4399-1

3. Monk, Ellen and Bret Wagner. Concepts in Enterprise Resource Planning. 3rd Edition. Boston: Course Technology Cengage Learning, 2009

4. Steven Nahmias, Production and Operation Analysis, Irwin 1989

5. https://en.wikipedia.org/wiki/Safety_stock

6. Harris, Ford W. Operations Cost (Factory Management Series), Chicago: Shaw (1915)

7. https://en.wikipedia.org/wiki/Integer_programming

新零售中的智能补货（II）— 需求预测

作者：阿里零售通算法团队 编辑：HohXil

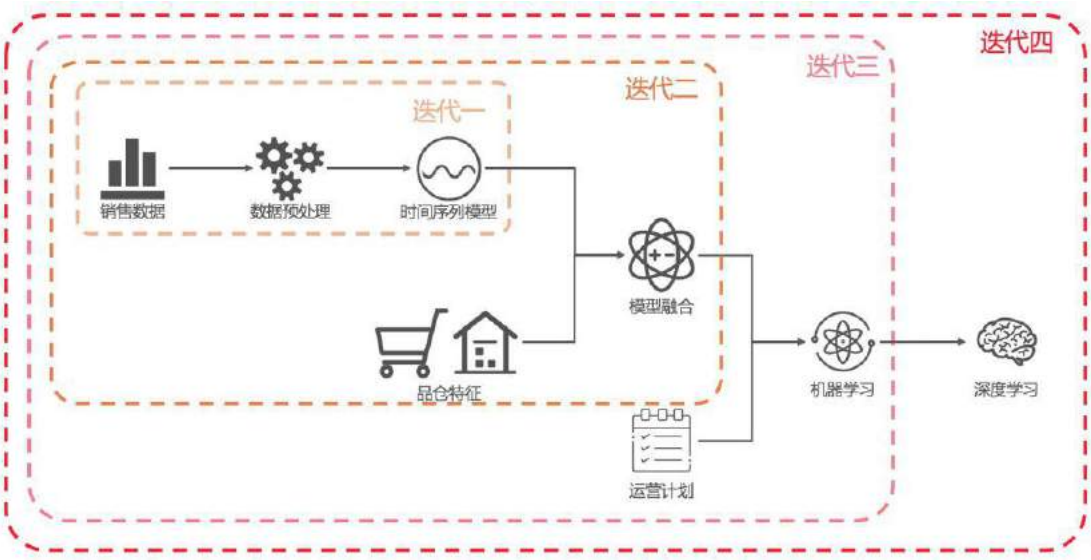
上一期讲到，要让新零售的补货系统根据当前库存状况智能地生成补货策略，除了要通过库存模型对现实的情况进行抽象，简化为整数规划模型之外，我们还需要对未来的需求分布进行精准的预测并作为库存模型的重要输入。这一期分享一下我们在零售通场景下销量预测的一些实践和经验。

▪ 总体方法

不管是传统零售还是新零售，需求预测永远都是一个意义重大且极具挑战性的问题。一方面，如果能准确预测出未来的需求，那么它将作为最可靠的依据，帮助公司精准决策；另一方面，未来的需求受到很多不确定因素（比如用户教育、市场推广、营销计划甚至天气等因素）的影响，导致准确的需求预测就像窥探“天机”一样困难。

传统零售在线下渠道的数字化上投入的资源不多，导致许多需求预测全靠从业人员的“专家经验”。这种纯粹凭个人经验和感觉的预测方式使得供应链库存经常高企，同时又有很多爆品的缺货发生。新零售带来的渠道数字化变革使得所有的预测和决策都“有数可循”，从根本上解决了“人治”带来的诸多困难和不确定性，使得完全通过数据驱动的需求预测和智能补货成为可能。

在机器学习的范畴，需求预测是一个典型的“回归”(Regression)问题。算法可以通过历史数据，以真实销量作为数据样本，不断调整模型参数，自动的学习到预测的最优解。我们需求预测的总体方法也是以机器学习模型作为主线的不断迭代。



基于机器学习的销量预测总体方法

这里，我们的预测模型经历了从均值模型，到时间序列模型，再到机器学习模型融合，最后到深度学习的尝试这一系列优化，最终在我们的预测线上系统中取得了理想的结果。下面将分别详细介绍每一次技术迭代的细节与其中的思考。

▪ 均值模型 & 时间序列模型

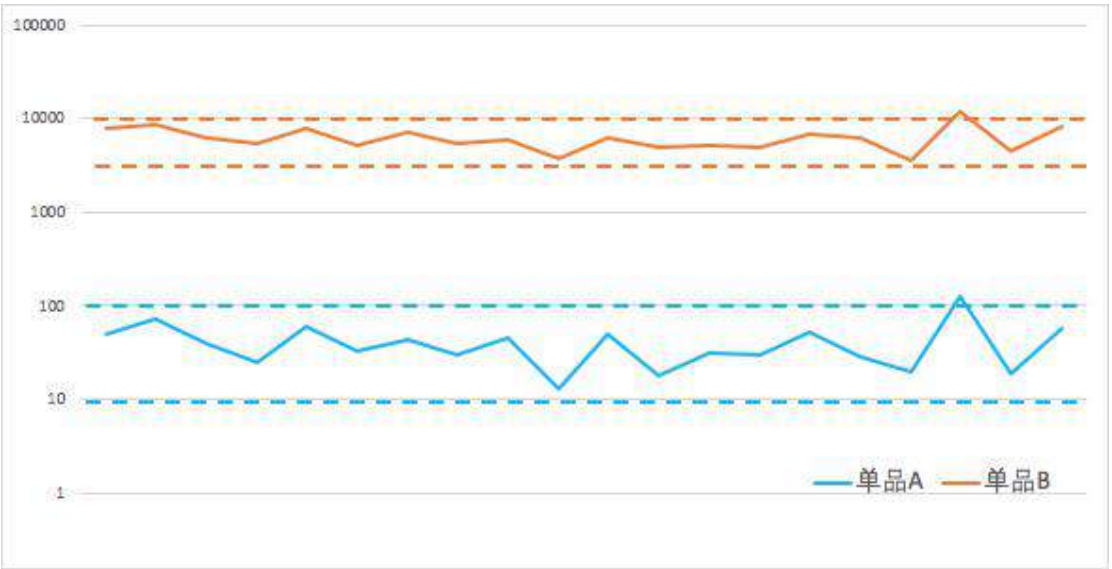
顾名思义，均值模型对历史一段时间的销量求均值，并作为未来每个时刻的需求预测：

$$X_t = \frac{1}{T}\sum_{i=1}^{T} X_i \quad \forall\, t > T$$

显然，均值模型过于简单，并且在实际应用中往往无法达到较好的效果。但是，构建均值模型具有以下两个重要意义：

1. 均值模型为需求预测提供了准确率的基线。如果要从 0 到 1 构建一个预测模型，第一步不是去构建一个复杂的模型，而是先看看简单的预测能达到什么样的准确率，作为后续模型迭代评估的标准。比如对于两个时间序列，机器学习模型预测的准确率分别是 70%和 80%，但是均值模型给出的的基线准确率分别是 50%和 75%，说明前者使用机器学习模型得到的提升更大。

2. 均值模型为预测值提供了合理的尺度。一般来说，某个单品的需求都在某个范围之内波动(促销和缺货除外)。需求不可能短时间激增，也不可能短时间骤减。因此这个波动的范围就是该单品需求的“尺度”，它可以通过均值模型简单得出。在机器学习回归模型中，某些单品销量的的预测值和真实值有时候会差一个量级。(比如真实销量 1000 个，但模型预测 100 个。）如果在模型中加入均值模型作为校准，就可以避免这样的偏差发生。



两个单品的销量时间序列，虚线给出的需求带就是该单品的“尺度”。注意纵坐标是对数坐标

时间序列模型是分析时间序列并进行预测的一种最通用的方法。在金融领域中，该方法得到了最广泛的应用。它主要通过对历史序列的分析和建模，将序列中的“趋势”和“白噪声”分离出来，最后通过对趋势的预测得到未来一段时间需求的变化。一些典型的时间序列模型有自回归滑动平均模型(ARMA)[1]，差分整合移动平均自回归模型(ARIMA)[2]，自回归条件异方差模型(GARCH)[3]；还有一系列指数平滑模型，比如三次指数平滑模型(Holt-Winters)等。

- 均值模型

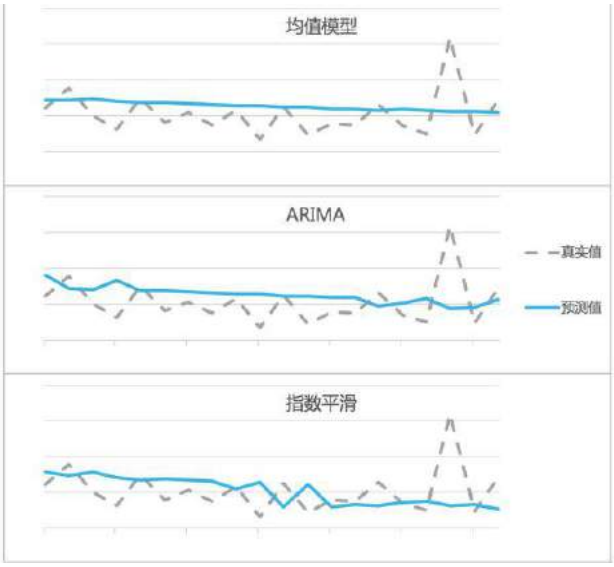
$$X_t = \frac{1}{t-1}\sum_{i=1}^{t-1} X_i$$

- ARIMA

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

- 指数平滑模型(Holt-Winters)

$$S_t = \alpha X_t + (1-\alpha)S_{t-1} \text{ (order 1)}$$



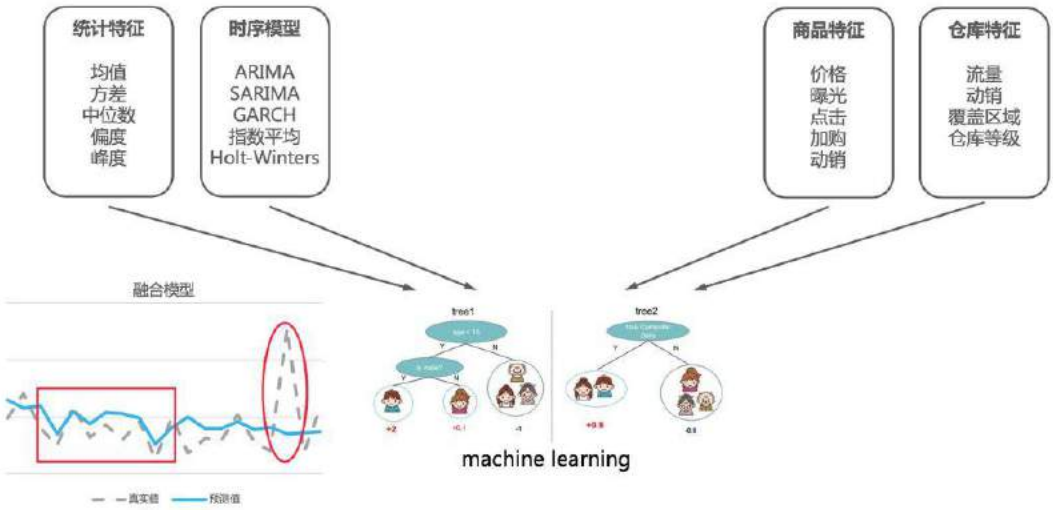
我们在零售通平台上对不同的时间序列模型都做了深入的试验和构建。总体来看，每个模型都在部分单品上表现优秀，但在其余品上无法做到准确的预测。究其原因，还是因为时间序列模型仅仅使用“历史的销量序列”来做预测。显然，我们的数据还包含了比销量丰富得多的数据，比如品仓特征，流量，动销用户数，运营计划等。这些都会对未来的需求产生不同程度的影响，而使用好这类数据只能通过机器学习模型进行预测。

▪ 机器学习模型

前面讲到，需求预测是一种典型的回归问题。因此我们可以用机器学习的标准范式和流程进行解决，这里就不再具体详述，而主要向大家分享两点在实际处理中的经验。

1. 特征工程。不同于其它机器学习模型，需求预测所构建的特征大多都是从历史的时间序列得到的。因此，如何从历史的时间序列中抽取特征就成为决定模型预测是否准确的关键。通常的时间序列抽取方法有滑动窗口平均值(比如过去 7 天平均、过去 30 天平均等)，序列统计特征(最大最小值、方差、偏度、峰度等)。除此之外，我们在实践中还

构建了很多高阶的特征，比如利用时间序列模型得到的预测值，还有通过多项式拟合时间序列得到的多项式系数等。除了历史的时间序列特征，我们还抽取了品仓的特征，作为该单品的固定属性。



2. 模型的个数。我们需要预测的品仓个数多达上万个，每个品仓的销量尺度都大小不一。有的品仓每天的销量多达上万个，有的却不足十个。在这样大尺度的品仓销量下，我们应该构建多少个机器学习模型对数据拟合并预测呢？有两种极端的方法。一种是对每个品仓进行建模。这种方法在实际操作中这是不可取的，一方面它会构建上万个机器学习模型，消耗大量的计算资源，导致模型难以计算和维护；另一方面模型没有用到品与品之间的相似性，无法得到准确的预测结果。另外一种方法是对所有的品仓构建一个统一的模型。这种方法虽然技术上最容易实现，但是它对模型本身的要求变得极高，需要模型在处理不同尺度上的预测都具有很好的表现。我们在实践过程中采取了折衷的方法。首先对品仓按照销量进行分层，然后对不同分层的品仓集合单独建模。一方面，同一层

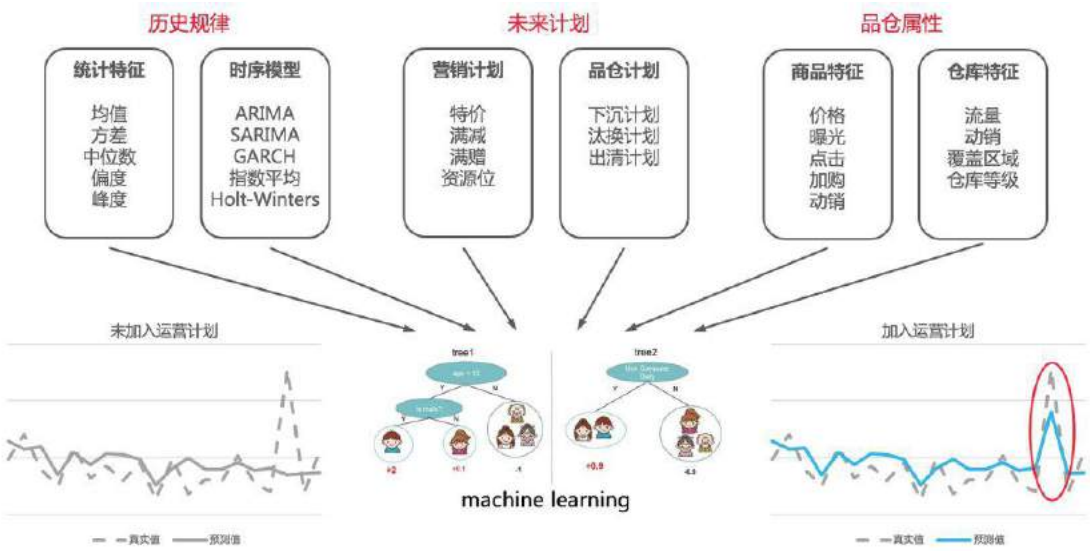
的品仓在销量尺度上大致一直，降低了对模型本身的要求；另一方面又可以用到品仓之间的相似性，提升了模型的泛化能力。同时，少量的模型也易于计算和维护。

▪ 运营计划的引入

上面讲到的无论时间序列模型还是机器学习模型，都只使用了历史的数据进行需求的预测。这些数据在通常的时间序列预测问题中已足够。未来所发生的某些数据波动都可以当作模型的“噪声”。但在新零售的场景中，未来的某些数据波动有可能是人为造成的(比如双 11 大促等)。如果我们提前知道了未来的运营计划，就可以对这些“噪声”进行较为准确的预测。

营销对未来的需求产生作用的手段无非就是“引流”和“降价”两种。前者增加了某个商品的曝光量，而后者主要提升了单品的曝光转化率。这两种手段都会在不同程度上刺激了用户的需求，带来销量的提升。从数据和算法的角度，如果我们知道历史上已经发生了哪些营销活动(比如去年的双 11 活动)，并分析出它对营销期间销量的影响，就可以大致估计出未来的营销计划对需求的拉动作用，作出营销期间较准确的需求预测。

除了营销的影响，供应链运营的某些行动也会导致销量的波动。比如单品从上级仓转换到下级仓(下沉)，品仓的汰换，出清等等。如果我们提前知道了这些运营计划，就能对未来的品仓进行准确的预测。在零售通的需求预测实践中，我们主要通过将营销计划转化为特征的方式来引入到机器学习模型中。

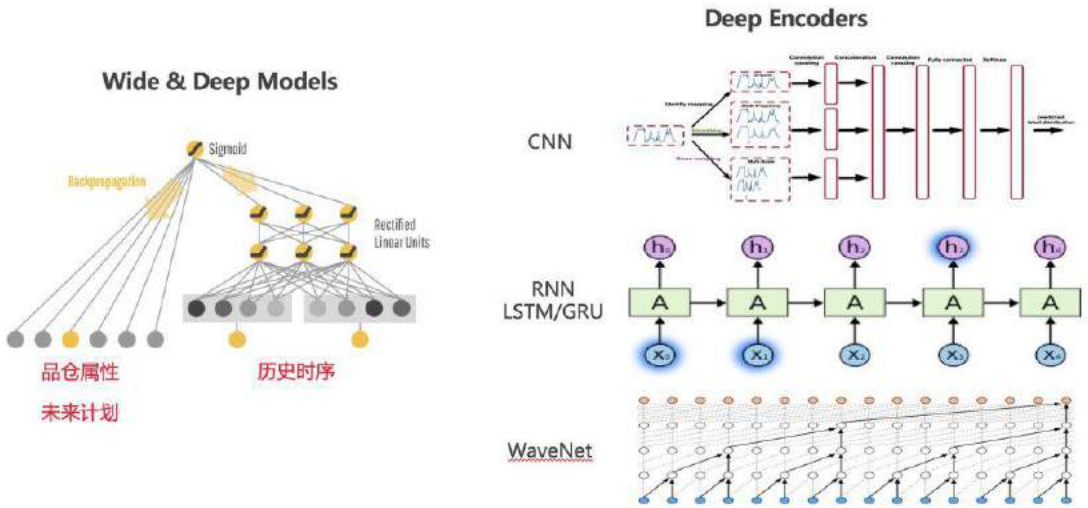


从实际效果来看，在加入了运营计划等特征后，模型在促销期间的需求预测准确率得到了明显的提升。

▪ 深度学习的尝试

传统机器学习的一个弊端就是需要人工构建模型的特征，而在某些场景下深度学习可以自动生成显著的特征，端到端的解决模型预测问题。在零售通的需求预测里，我们也做了一部分深度学习的尝试。

我们可以对获取到的数据分为两类。一类是历史的销量，流量，转化率等时间序列连续特征，它们可以通过深度学习模型进行特征编码。另一类是营销计划，品仓属性等离散特征，比如活动玩法，仓库所覆盖的城市等。它们由于特征的稀疏性，很难直接通过深度学习模型进行特征编码。（embedding 方法除外。）因此，我们借鉴了推荐搜索中的 Wide & Deepmodel[5]，将时间序列特征放在 Deep 部分，强化它们的特征编码能力，同时将离散特征放在 Wide 部分，引入到模型的最后一层中进行预测。



为了更加适应历史时序这种特殊的数据结构，我们修改了 Deep 部分的网络结构，从全连接层变换到 CNN、RNN 等 Deepencoders。这样，处理时序模型的各种深度学习技术(比如注意力机制等)都可以引入到该框架中。在某些品仓中，我们的深度学习模型得到了比传统机器学习更加稳定和优异的效果。

▪ 需求预测的误差

前面讲的需求预测总体方法都聚焦在如何得到需求的某一个值，而库存模型需要的输入是一个需求的分布。在需求正态分布的假设下，我们还至少需要给出需求的标准差，才能输入到库存模型中得到最终的补货量。在我们的实践中，估计需求的标准差主要用到了以下两种方法：

1. 如果预测模型使用的是 ARIMA 等时间序列模型或者线性回归模型，我们可以通过模型拟合的误差计算出参数的误差，进而给出预测目标的标准差。事实上，这种处理预测误差的方式是贝叶斯模型(Beyesian model)的一个特例。在线性回归模型中(ARIMA 时间序列模型可以当成一种特殊的线性回归模型)，参数的误差可以通过解析表达式计算

得到。如果我们使用了较复杂的机器学习模型 那么要得到预测的分布需要 Gibbs 抽样，这就增加了模型计算的复杂度。

2. 第二种方法是通过对比历史回测中的真实值与预测值，得到预测值偏离真实值的标准差。这种方式相比与贝叶斯模型计算更加简单直接，在一般情况下也能估计出模型真实的预测误差。更重要的是，这种方法不依赖具体的模型，无论我们最终用 GBDT 还是 Deep & Wide Model 进行预测，它都能估计出一个较为准确的误差。我们在实际运用中也是以这种方法为主来得到预测的标准差并作为库存模型的输入。

以上就是新零售场景下我们在需求预测中的部分实践。从总体技术框架上，它与传统的机器学习回归问题的技术演进路程差异不大，这里仅仅是提纲挈领地分享了我们在每一步迭代中的方法和经验供大家参考。对于具体的场景和具体的数据，我们也会通过不断优化数据处理和模型构造的各种细节来得到最大幅度的准确率提升，这里就不再一一详述。希望我们的分享能够抛砖引玉，以文会友，让大家对我们在新零售场景下的补货和需求预测算法有所了解，并提出建议和共同提高。

参考文献：

1. https://en.wikipedia.org/wiki/Autoregressive–moving-average_model

2. https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

3. https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity

4. PrajaktaS. Kalekar. "Time seriesForecasting using Holt-Winters Exponential Smoothing"

5. Cheng H T, Koc L, Harmsen J, et al. Wide& Deep Learning for Recommender Systems[J]. 2016:7-10

绿湾机器学习平台（AutoML）打造与落地实践

嘉宾：于建岗　编辑：孙佩霞

今天给大家介绍绿湾科技的机器学习平台。

1、为什么打造平台——释放数据更深层次的价值

大数据与 AI 技术是当今社会创新发展的主题，已经成为各行业提高发展质量和效益的动力引擎。绿湾科技紧跟大数据技术发展趋势，积极探索和引领大数据技术在特定行业应用中的落地实践。随着研究探索的深入，对于海量多源异构数据蕴含的巨大价值，需要新的更有效的技术理念来撬动和助推，绿湾科技的实践证明，我们的大数据分析系统（智子）+ 绿湾机器学习平台（AI 平台）就是很好的选择。首先，依托于绿湾科技现有系统在各领域内的部署拓展，机器学习平台作为公司的 AI 引擎，可以对数据做更深层次的分

析挖掘，解决更复杂的业务场景问题。其次，经过数据治理和数据整合后，为释放数据更深层次的价值，机器学习平台成为必然。

1.1 业务路径——基于行业应用场景的 AI

现实中存在海量多源异构数据，数据有非常多的形式，而且不同来源，不同格式。如何把数据处理得更好，需要进行数据治理。绿湾科技的数据治理运用了大量机器学习平台的模型，包括 ETL，清洗，存储等，利用机器学习做了很多这方面智能化的工作。

绿湾科技提供 AI 行业解决方案。我们所做的 AI 一定是基于行业应用场景的 AI，深入行业，提供行业解决方案，为行业赋能，我们的目标不是仅提供通用 AI 的技术服务。

比如在司法行政领域，我们支撑司法行政大脑；在公共安全领域，我们打造我们的公安大脑；在金融领域，我们同样和金融业务特点相结合，打造我们的金融大脑。

在智慧城市中，人口大数据包含人口怎样迁徙；上下班出行路线；公交车如何更好地调度；产业配置是否合理。举个例子，朝阳区建国门地铁站每天排队长达一公里，这样是否合理，为何出现这种情况？我们从人口迁徙，产业配置，上班族商务楼分布等维度分析其是否合理，给相关部门提供城市监测、反馈、分析，进而提供城市治理的优化方案。

在金融保险行业，大数据进入有相应壁垒，金融是一个专有特性的行业，业务比较敏感，数据比较保密，跨行跨部门的数据统一有一定难度，这时候只是简单的提供 AI 技术服务大多数情况下并不能满足不同金融从业者的需求。比如反欺诈，内审等，我们也以此切入打造金融大脑。

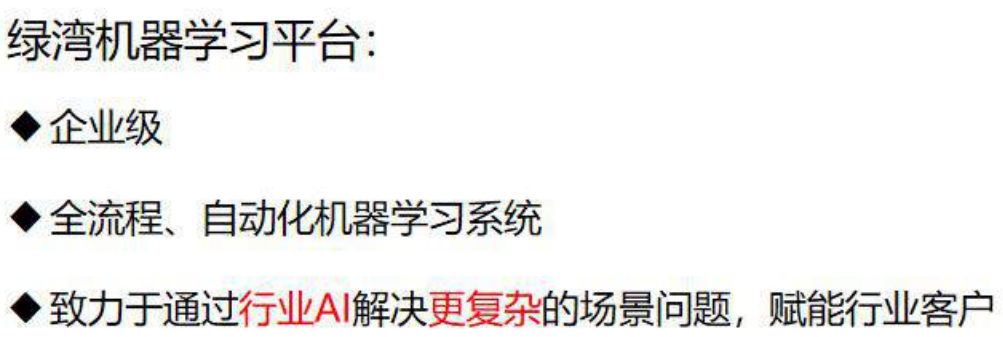
总体来看，绿湾打造的行业应用场景中的很多任务的底层都是机器学习平台来支撑。在机器学习平台之上需要很多专业的能力如自然语言处理，光学字符识别，图像识别等来做特定的工作。

1.2 技术路径——数据技术的真正闭环

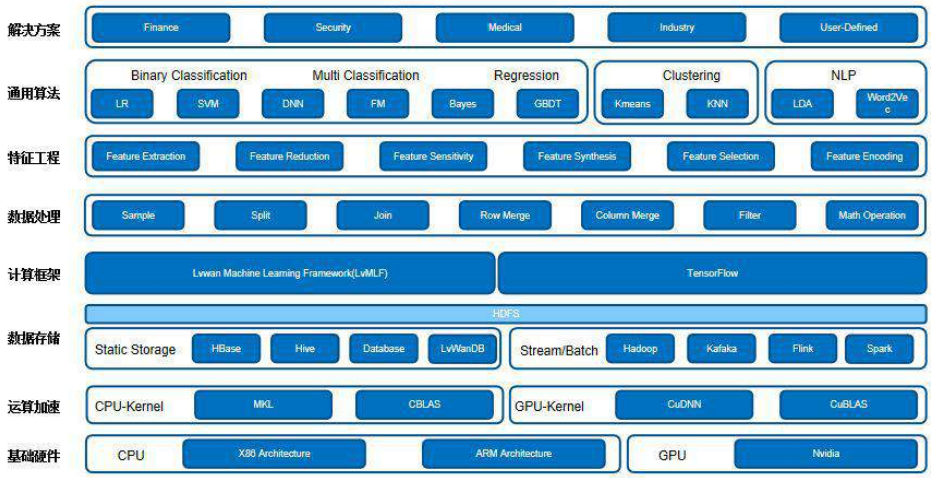
社会是以人组成的，所有社会上存在的一草一木，任何一个事件都是离不开这个“人”的，人是这个社会的有机体。所以绿湾科技以人为本体，构建了基于知识图谱的网络。底层是全源数据，如之前介绍，这些数据都是多源异构的。然后将各个数据融合成一套人、车、物、案的逻辑。通过数据的组织分析清洗和数据挖掘，上层形成了主题库，包括实体库、标签库、知识库。在这个基础上需要做应用，所以上层又做了知识发现，包括关联分析、图搜索、图像识别等。最上层又做了很多应用。这便是我们的一套打法。

这整套的技术方案，形成了一个闭环。以技术驱动从底层的数据清洗、数据治理、数据入库、图数据库、上层应用形成了这样的闭环。我们希望在机器学习平台的基础上做整个行业的解决方案，而不仅仅提供 AI 技术服务。所以当把机器学习平台做起来后，可以完完整整地将闭环打通，这就是绿湾科技为什么要做机器学习平台的出发点。

2. 平台特点——架构领先，支持超大规模数据训练



绿湾机器学习平台是一个企业级的平台，支持千亿数据，万亿参数的同时训练，既支持同步又支持异步训练。同时我们自建了自己的参数服务器。同时也借鉴了此前研发的大型系统数据挖掘平台经验。它是全流程、自动化机器学习系统，致力于通过行业 AI 解决更复杂的场景问题。我们建设的是一个行业大脑而不是通用大脑，提供行业解决方案。



针对现在各行业应用的多样性，我们的平台除了自有研发了 200 多种常用算法之外，我们同时也提供 API 服务支持第三方应用，比如我们也深度集成 TensorFlow 以更好的满足不同客户的需要。

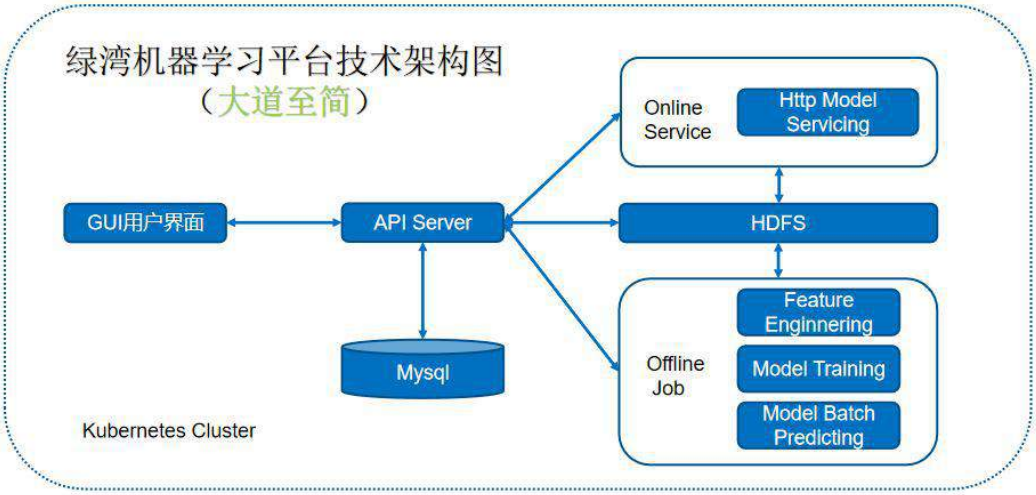
绿湾机器学习平台打造了自己的 MLF（Machine Learning Framework），所有任务的调度和运行，参数服务器参数的更新，以及模型的分片和数据的分片，全都基于 MLF 平台。MLF 平台可以部署在一台机器上，同时可以横向扩展到几百上千台服务器的集群规模。

绿湾机器学习平台有大规模 MLF 平台部署的经验，支持高并发的在线预测和离线训练业务。

架构领先，支持超大规模数据训练

- ◆ 自研的基于参数服务器的开发框架，支持对千亿数据，万亿参数的训练
- ◆ 深度集成TensorFlow，并做定制化优化
- ◆ 系统所有组件都以docker提供服务，通过kubernets作为统一的集群资源管理和调度管理
- ◆ 既支持在单台PC机运行，同时可以横向扩展到几百上千台服务器的集群规模

绿湾机器学习平台所有的系统、组件都是基于 docker 容器的方式提供服务，通过 kubernets 做统一的集群资源管理和调度管理 ,这样在部署上比较适合用户应用。



绿湾机器学习平台的技术架构非常简单，大道至简。首先提供一个 GUI 的用户界面，用户通过拖拉拽方式就可实现自建方案。同时以图的方式存储下来

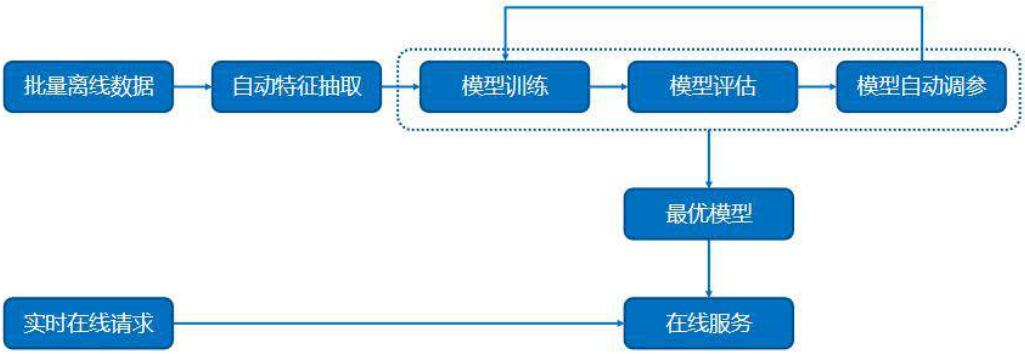
供所有人搜索已做完的一些行业解决方案，这些方案以算子的形式提供。搜索到成功的算子后，也可以去调用这个方案。在下一层我们提供了 API Server，可以调度一些任务，做在线预测或者离线训练等。使用 MySQL 的数据库去做容错处理。比如任务在中间失败，会在失败的算子继续执行而不是从头开始。一些出错的信息会以界面的形式及时地展现出来，可以去看运行的 Log，也可以去看运行的结果，只需右键就可呈现。

- 既提供通用的算法，又提供针对具体领域的解决方案
- 行业内用户只需灵活配置相应算法即可实现定制化的效果

绿湾机器学习平台在行业解决方案上，提供了几百种通用的算法，支持聚类、有监督学习、无监督学习、迁移学习等。同时我们还提供了具体领域的解决方案，如司法行政领域各种人群分析，金融反欺诈。用户可以用这些现有的解决方案，更改数据输入便可做用户想做的工作。

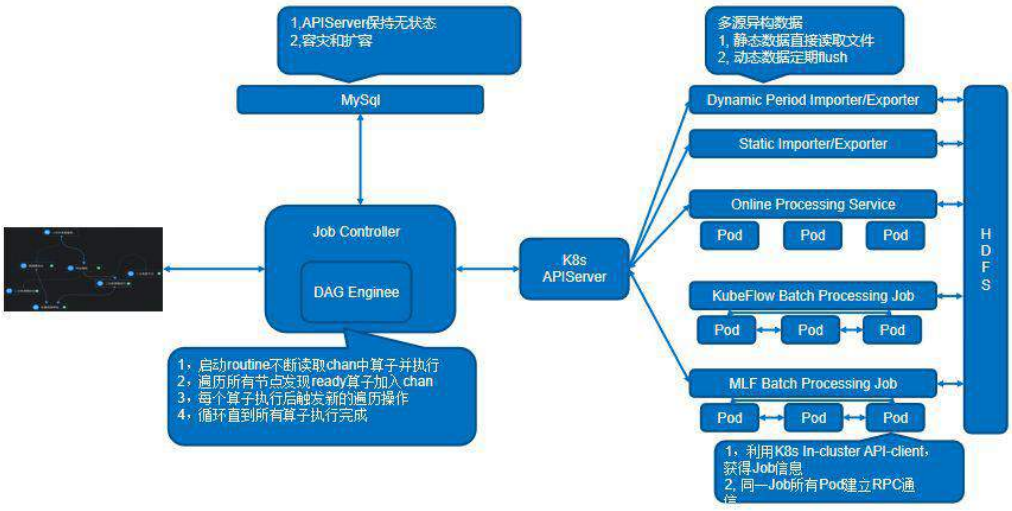
绿湾机器学习平台提供简单易用一站式的体验：

- 基于领先的 AutoML 自动建模，自动构造网络结构等技术，可实现自动特征抽取、筛选、模型训练及结果展示；
- 支持超参数的自动化学习和迁移学习；
- 提供可视化的操作界面；
- 覆盖全流程的数据挖掘链路，集数据清洗、特征工程、模型训练、模型评估及预测功能为一体。



上图是简单的建模逻辑图，从训练到预估等与其他方案没有很大区别。后面会以绿湾科技的 API Server 的实现方式，去介绍一些相关工作。

绿湾的 API Server，包含前端交互，DAG 调度，状态管理，集群调度。

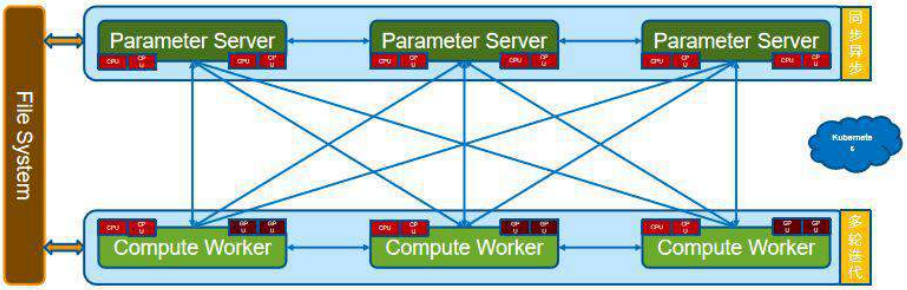


最左侧是 GUI 的建模的算子，算子通过边相连。下面用一些 Job Controller 去控制 job 。这部分由 MLF 来做，具体细节不再阐述。Job Controller 主

要是启动 routine 不断读取 chan 中算子并执行；遍历所有节点发现 ready 算子加入 chan（ 中间会存储很多状态的细节 ）；每个算子执行后触发新的遍历操作 ;循环直到所有算子执行完成。K8s 调度后会产生 Pod ,Pod 建立 RPC 通信，去做 online prediction，或者 offline 基于 MLF Batch 的训练，加上 HDFS 或者其他数据库的存储。

离线训练

1. 模型分布式（SparseTable & DenseTable）
2. 数据分布式（Batch & Streaming）
3. 执行范式（Lazy Execution or Eager Execution）
4. 参数服务器模式（Server-client or AllReduce or Both）

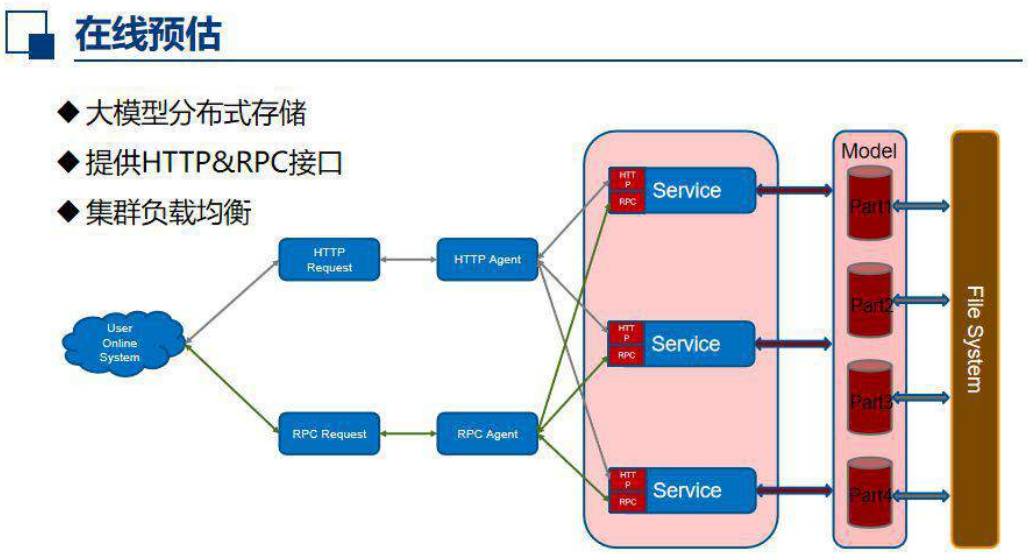


在训练方面，分布式从两个角度去考虑。第一个是模型的分布式，我们基于不同的参数类型提供两种参数的实现方法 ,一个是 spark 的 table（ 基于哈希表的离散 table ），另外一个是基于 vector 稠密的 table 。

第二个是数据的分布式，为了处理的效率，数据的每个节点，每个线程以 block 为单位处理。为了控制数据处理带来的内存开销，采用了基于管道的方法多线程传输数据，数据传输、处理结束后及时销毁，达到流式效果。

执行范式方面， Eager Execution 对用户来讲很简单，但执行效率偏低，因此为了执行效率选择 Lazy Execution 。

参数的服务器模式，在众多的参数服务器模式中，AllReduce 网络开销比较小，Server-client 模式比较大，我们采用了同进程的 Server Worker 的模式，避免了 Server-client 网络开销大的问题。AllReduce 环状问题更多，因此没有采用。



在线预估方面，采用大模型分布式存储，提供 HTTP&RPC 接口，集群负载均衡（由 MLF 提供）。用户发出请求，通过 HTTP 或者 RPC 去调用 Service，去训练 model 。Model 是分布式训练，分布式存储。

3. 落地实践——绿湾社交人群网络分析解决方案

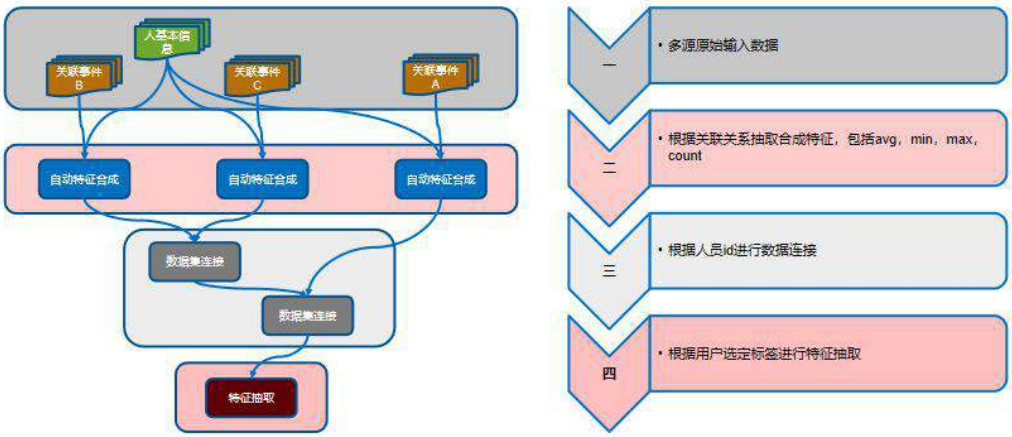
社交网络分析非常复杂，比如只知道网络中的正样本不知道负样本，网络中谁是坏人，谁是好人无法确定，剩余人中既有好人又有坏人，需要在剩余人中找出未知的坏人。正样本可知，负样本未知，模型如何来做可解释性？接触行业数据时发现数据非常稀疏，且不完整，很多缺失的数据。针对数据完整的应用，做了有监督的学习。针对数据非常稀疏，且不完整，负样本缺失等，做了无监督的学习。

机器学习对特征做 Embedding 是非常重要的一点，将很多信息 Embedding 在一起。由于我们是以图的方式处理数据，利用 DeepWalk 去做人的类聚。DeepWalk 的处理方法是把网络处理成序列去算 Word2Vector，得到的 Embedding 信息和人的信息比较类似，因此作用更趋向于聚类。TransE 更多的是 nodeA + link = nodeB 形式，更侧重于关系在网络中人的关联性。利用 Graph Neural Network 做了 Embedding，充分利用 label 的信息来建模，得到人在网络中受到关联人的影响。将这三种模型产生的数据信息做了 Embedding，将 Embedding 之后的特征再放入模型中进行训练，得到最终结果。即做了 Bagging of Graph Embedding 的工作，对人社交进行分析，得到很好的效果。具体总结就是：

一个是在特征构造阶段，引入了（Bagging of Graph Embedding），因为不同的 Graph Embedding 算法侧重与对网络中的某些属性建模，我们进行 bagging 可以取得更加完整的信息。

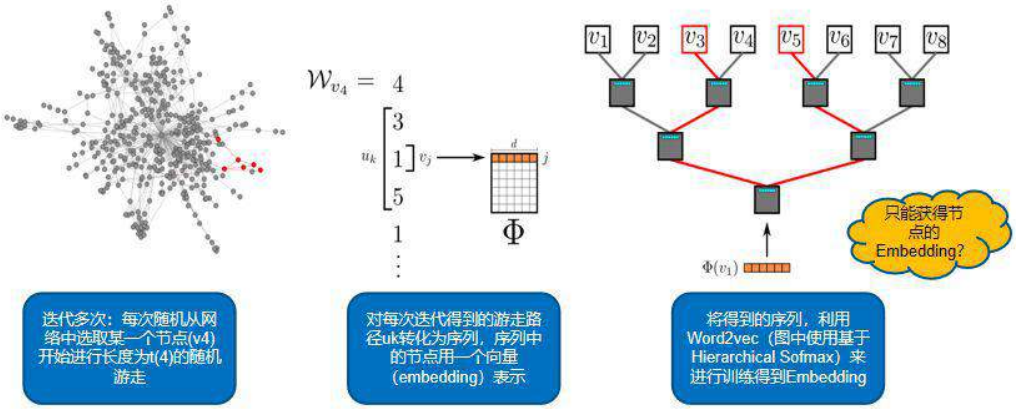
另外一个创新的地方是在模型选择阶段，针对不同的业务场景有的数据较为完整，特征较多我们采用有监督的方法（有监督方法同样有些创新，针对只有正样本的情况我们利用业务规则拆分样本为正样本，负样本和待定样本。同时为了模型更好的解释性对模型的预测结果按照业务规则排序）;针对数据稀疏，特征较少的场景我们采用无监督的方法（通过 ANN 聚类，根据对 Topk 的相似度来对样本进行排序判断）

基本属性特征：自动特征合成

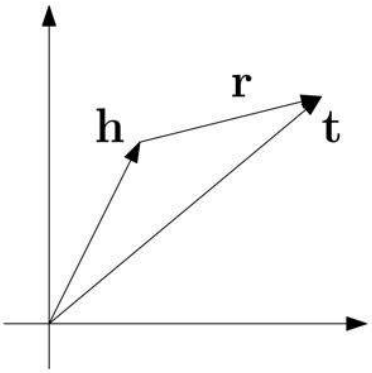


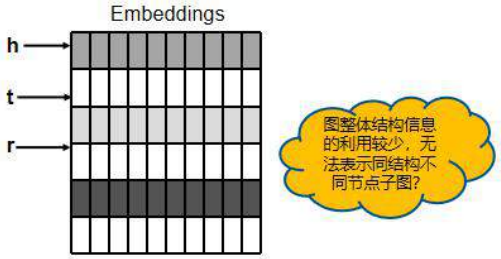
上图是我们的具体过程，不再细说。

Embedding特征 I：DeepWalk



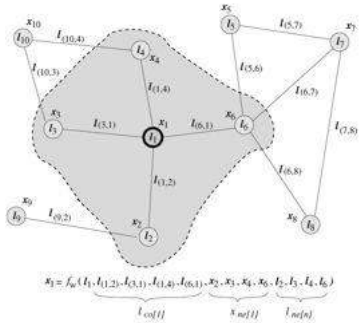
Embedding特征 II：TransE





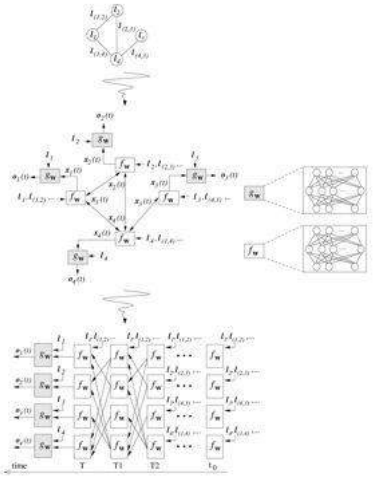
- 算法将图中的每条边都看做是一个三元组 **h+r=t**
- 从边的起始节点**h**出发，通过**r**之后可以到达结束节点**t**，从而进行有监督学习
- 采用mini-batch-sgd算法直接进行训练

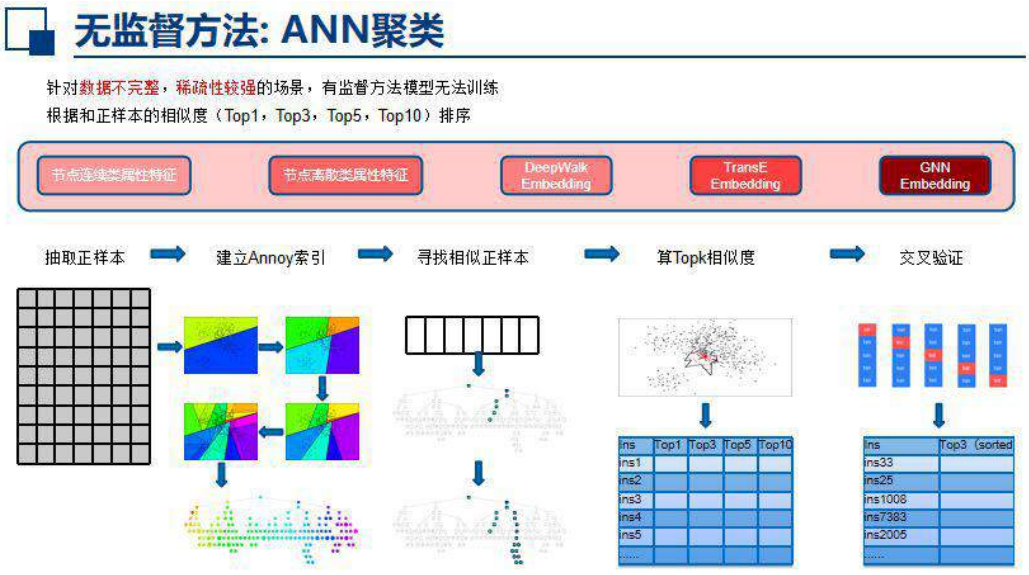
Embedding特征 III：Graph Neural Network



- 图中的每个顶点代表一个对象，边代表顶点之间的关联关系。
- 每个顶点的表示是由其本身属性和其关联的顶点的表示和边来决定的
- 利用巴拿赫不动点理论转换为递归求解公式，最终转换为RNN

$$\boldsymbol{x}_n(t+1) = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}_{\mathrm{ne}[n]})$$
$$\boldsymbol{o}_n(t) = g_{\boldsymbol{w}}(\boldsymbol{x}_n(t), \boldsymbol{l}_n), \qquad n \in \boldsymbol{N}.$$



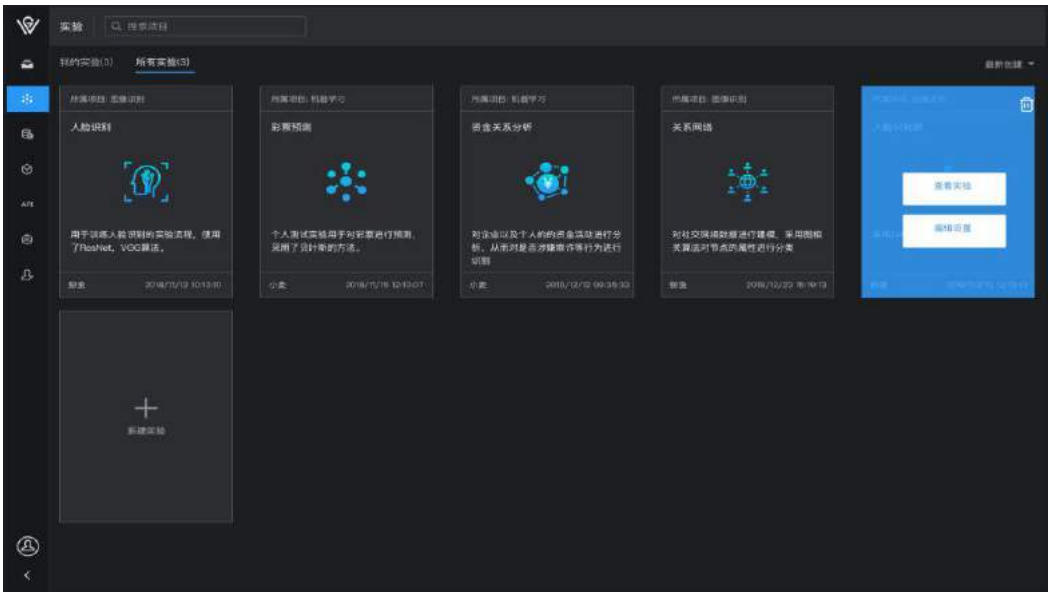


针对之前所述的数据不完整，稀疏性较强的情况，有监督的方法无法进行训练，即使训练出来结果也很差，还不如人工处理。在这里根据正样本的相似度，利用业务规则对最后的结果进行排序。DeepWalk Embedding，TransE Embedding，Graph Neural Network Embedding，加上离散类特征和连续类特征，通过 MLF 的架构去抽取一些正样本，建立索引，根据业务规则去寻找相似的正样本，训练后得到 TopK 的相似度，然后交叉验证。以上便是我们摸索出来针对样本稀疏情况下进行聚类的方案。

4. 平台 look alike？—— 与绿湾智子大数据智能应用系统无缝连接

绿湾科技的平台会提供 MLF 的框架，自建的参数服务器，去支撑上层建模的应用和各种业务场景的应用。下面展示的就是我们的机器学习平台，该平台与绿湾智子大数据智能应用系统已经无缝连接。我们在该平台上做了很多行业解决方案，点进去以后可以复制该方案，改变输入后便可自动调度任务

运行。用户也可以搜索别人运行成功后的方案，也可以自建方案。用户可通过 GUI 的方式调动后端的 API Server 调度任务去训练、预估等。



另外展示一下我们基于人群关系网络做的分析工作，基于算子的一套简单流程。左侧有大约几百种的算法，可以手动拖拽，可以更改算子的参数，也可以通过自动学习获取参数。当任务运行到某一块的时候，该块会变绿，如绿色圆圈所示，出问题时会变红，右键点击时会出现错误信息和运行后的结果。

以上便是绿湾科技的机器学习平台，我们是提供基于机器学习平台的行业解决方案。

新一代联邦学习技术及应用实战

嘉宾：范涛　编辑：Hoh Xil

导读：随着大数据的进一步发展，重视数据隐私和安全已经成为了世界性的趋势， 同时，大多数行业数据呈现数据孤岛现象，如何在满足用户隐私保护、数据安全和政府法规的前提下，进行跨组织的数据合作是困扰人工智能从业者的一大难题。而“联邦学习”将成为解决这一行业性难题的关键技术。今天会和大家分享下微众银行主导的新一代联邦学习技术及应用，而 FATE 则是联邦学习落地的一个工业级开源平台。

今天的介绍会围绕下面五点展开：

- 联邦学习背景介绍
- 纵向联邦学习
- 横向联邦学习
- 应用案例
- FATE：联邦学习开源平台

联邦学习背景介绍

首先和大家分享下联邦学习的背景。

1. AI 落地的理想与现实

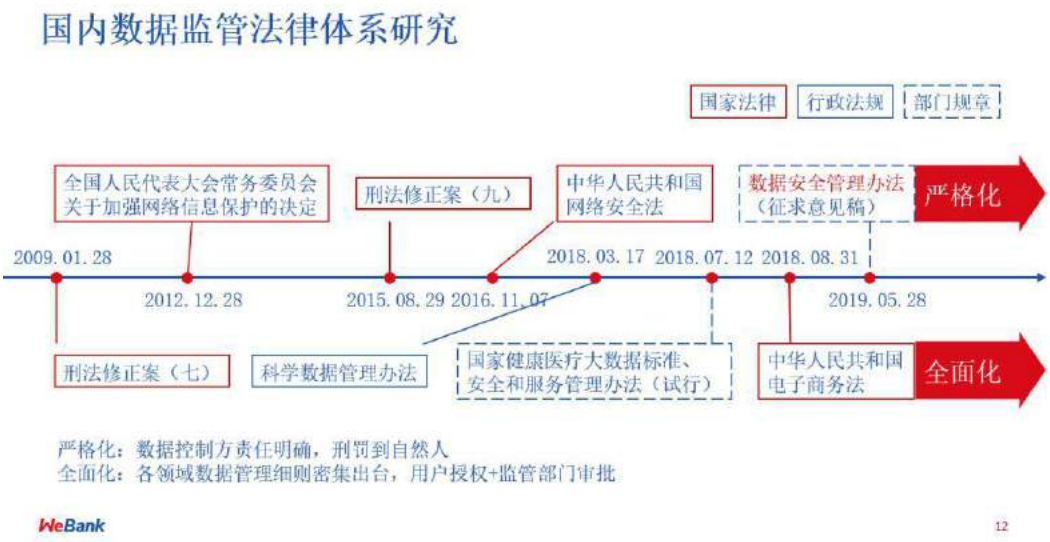
我们知道 AI 落地的时候，会遇到很多现实的问题：

现实中，我们的数据质量是非常差的，比如聊天数据中有很多噪音；

数据标签，收集是比较困难的，很多场景中的数据是没有标签的；

数据是分散的，（这也是最重要的一点）每家应用的数据不一样，比如腾讯用的是社交属性数据，阿里用的是电商交易数据，微众用的是信用数据，都是分散来应用的。现实中，如何进行跨组织间的数据合作，会有很大的挑战。

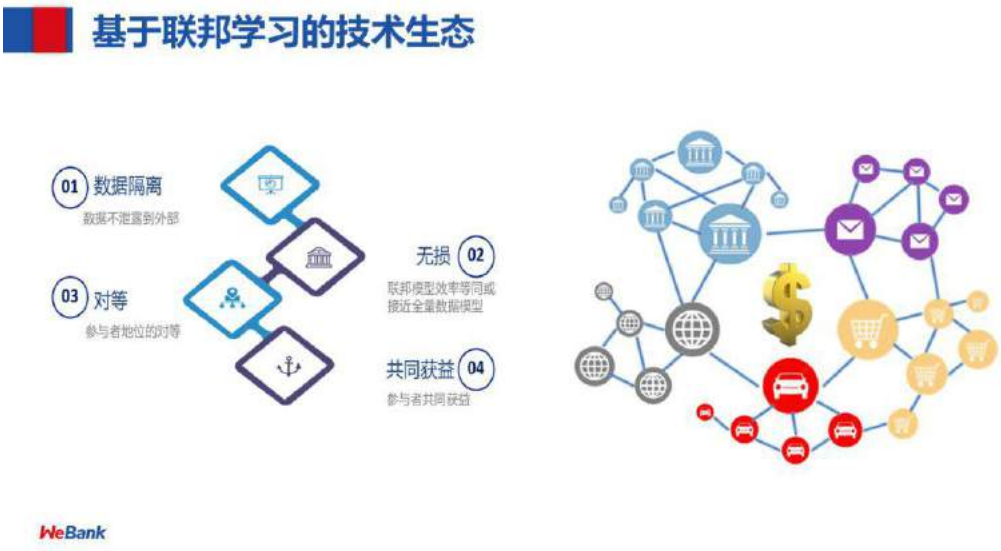
2. 国内数据



监管法律体系研究

从 09 年到现在的 10 年时间内，国家关于数据的法律条例是趋向于严格化的，同时趋向于全面化，每个细分领域都纷纷出台了相应的条例和条款。相对来讲，让之前可行的一些数据合作方案变得不太可行。

3. 基于联邦学习的技术生态



针对上述问题，微众银行提出了基于联邦学习的技术生态，特点如下：

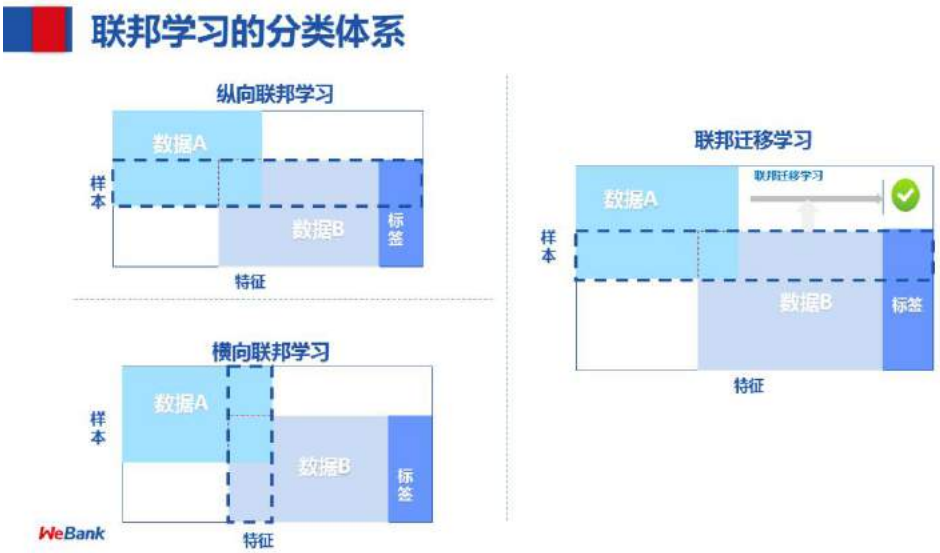
数据隔离：联邦学习的整套机制在合作过程中，数据不会传递到外部。

无损：通过联邦学习分散建模的效果和把数据合在一起建模的效果对比，几乎是无损的。

对等：合作过程中，合作双方是对等的，不存在一方主导另外一方。

共同获益：无论数据源方，还是数据应用方，都能获取相应的价值。

4. 联邦学习的分类体系



联邦学习的分类体系，包括：

- 纵向联邦学习，两个数据集的用户（U1, U2, ...）重叠部分较大，而用户特征（X1, X2, ...）重叠部分较小；
- 横向联邦学习，两个数据集的用户特征（X1, X2, ...）重叠部分较大，而用户（U1, U2, ...）重叠部分较小；
- 联邦迁移学习，通过联邦学习和迁移学习，解决两个数据集的用户（U1, U2, ...）与用户特征重叠（X1, X2, ...）部分都比较小的问题。

下面，重点分享下纵向联邦学习和横向联邦学习。

纵向联邦学习

1. 联合建模需求场景

举个例子 :微众与合作企业进行联合建模 ,比如做信贷逾期模型 ,微众有 Y 数据，包括标签数据，逾期记录，用这样的数据可能会建一个很好的模型，但我们希望用更多的数据，比如合作方的标签数据和画像数据来更大的提升风控模型的效果和稳定性。

传统模式的问题是：

合作企业缺乏 Y 无法独立建立模型，需要微众把 Y 数据，带入到合作方的生产环境建模，但是由于国家的数据保护条款和各企业自身对数据的严格规定，得到的 X 数据不能全量的传输到微众。

针对这个问题 ,通过纵向联邦学习来解决。如右图 ,两边的数据都有共同的 ID，特征是完全不一样的，可以通过一方特征来弥补另一方特征的不足。

2. 同态加密技术保护隐私

纵向联邦学习的技术实现，首先应做好两点，来保护数据隐私：

- 建模样本 ID 差集不向对方泄露，在合作之初需要进行用户匹配，需要找出用户的交集，但是不能泄露差集，因为这是企业最核心的资产。
- 任何底层（X，Y）数据不向对方泄露，建模过程中如何保证数据不被泄露。

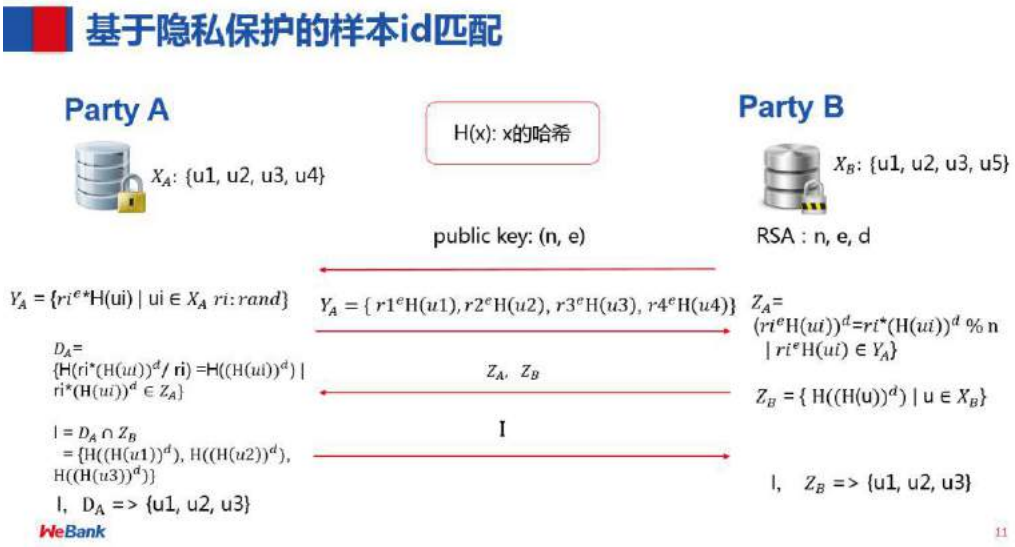
解决方案：

- 通过 RSA 和 Hash 的机制，保证双方最终只用到交集部分，且差集部分不向对方泄露。
- 采用同态加密技术，这个过程中，各方的原始数据，以及数据加密态都没有被传输。交互部分，双方通过损失中间结果，用同态加密的机制进

行交互，模型训练完之后，会各自得到一个模型，各自的模型会部署在各自的一方，就是如果我只提供了 3 个特征，那么我只有 3 个特征的模型，只提供 2 个特征，就只有 2 个特征的模型，任何一方的模型都没法单独去应用，只有共同应用的时候，才能进行决策。

3. 基于隐私保护的样本 id 匹配

刚才提到基于隐私保护的样本 id 匹配，和大家分享下具体的技术方案。比如，A 方有 [u1，u2，u3，u4] 四个用户，B 方有 [u1，u2，u3，u5]，那么整个过程中，如何保证双方知道 [u1，u2，u3]，而 A 方不知道 B 方有 [u5]，B 方不知道 A 有 [u4]？



这里是通过 RSA 和 Hash 的机制做到的，B 方会作为公钥的生成方，会把公钥给到 A 方，A 方基于 Hash 引用一个随机数，再交互传给 B 方，B 方同时做 Hash 然后传给 A 方，A 方会最后做一个结果的交集。整个过程中，你可以看到没有任何一个明文数据传递过来，即使采用暴力或者碰撞的方式，依然解析不出原始的 id。通过这套机制，我们很好的保护了双方的差集部分。

4. 同态加密

同态加密

- 全同态或者半同态 Full Homomorphic Encryption and Partial Homomorphic Encryption
- 数据层面的信息保护 Data-level information protection

Paillier 半同态加密 Partially homomorphic encryption

Addition :	[[u]] + [[v]] = [[u+v]]
Scalar multiplication:	n[[u]] = [[nu]]

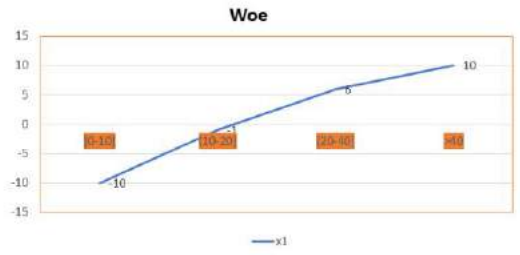
Rivest, R. L.; Adleman, L.; and Dertouzos, M. L. 1978. On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press, 169 - 179.

WeBank 12

分享了匹配过程中隐私保护的问题，接下来分享一个通用的技术，同态加密。

刚才提到建模过程中，引用了同态加密技术，比如对两个数字进行加密，加密后两个数字的密文可以进行数学运算，比如加法，其结果依然是密文，对密文解密后得到的结果和它们明文的加法结果是一样的。

联邦特征工程



特征IV值

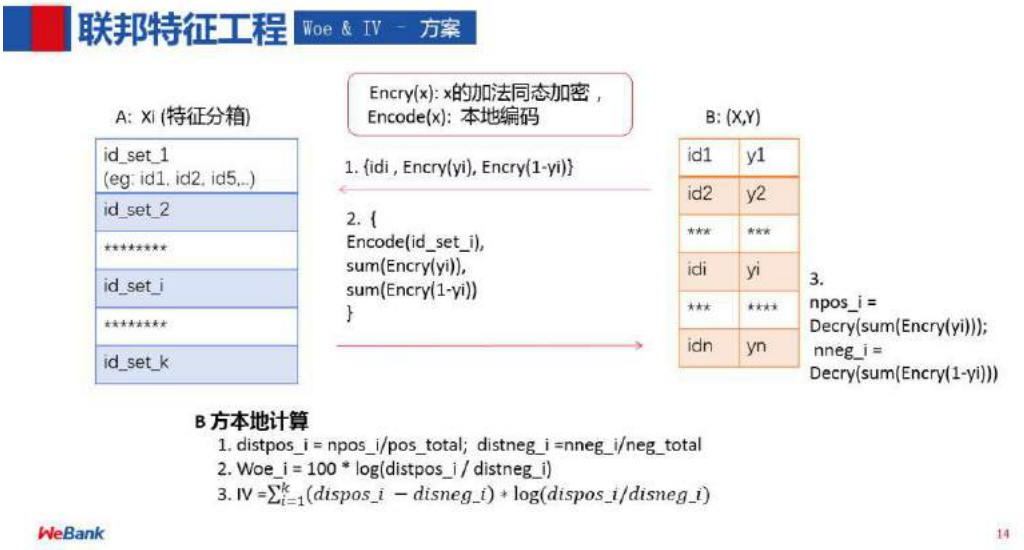
特征	IV
x1	0.3
x2	0.2
x3	0.4
x4	0.01
x5	0.05

问题：在保护双方隐私下，A侧(含X)和B侧(含X, Y)特征如何计算WOE和IV？
难点：
- A 侧只有特征x，没有y; 计算Woe和IV得同时依赖x,y (B侧特征Woe &IV 可以本地计算)
- A侧不能对B侧暴露x，B侧不能对A侧暴露y
- 最终只能让B侧获得所有特征Woe & IV

WeBank 13

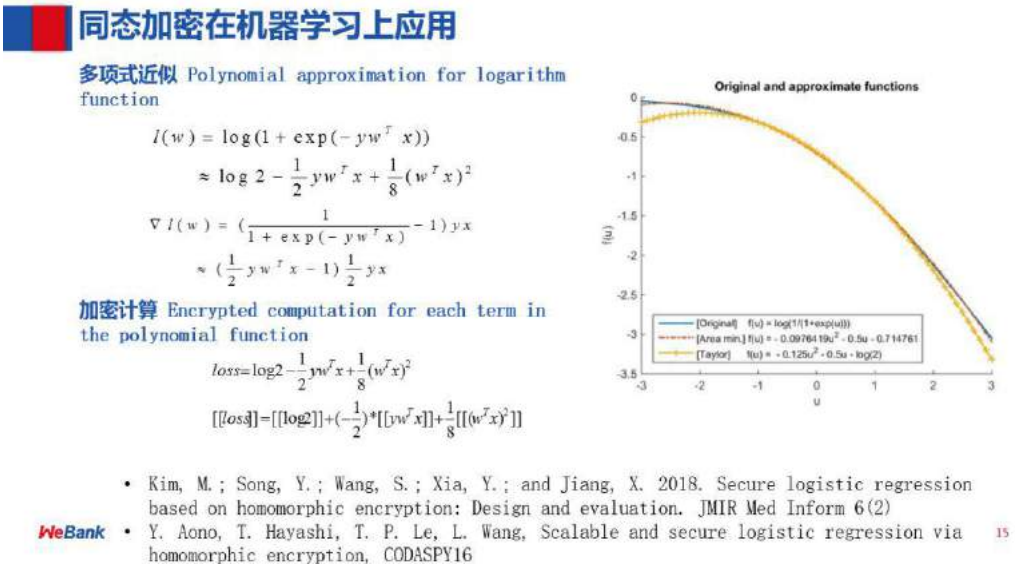
通过这样的同态加密技术，我们把它应用到机器学习，包括特征工程中。接下来会重点介绍，联邦机制下特征工程和机器学习建模的一些细节。

我们知道特征工程是机器学习建模中非常重要的一环，在联邦机制下，如何完成联邦特征工程？尤其 A 方只有 X 没有 Y，如果想做一个 WOE 或者 IV 值的计算是非常困难的。那么如何在联邦学习的机制下，A 方利用 B 方有 Y 的数据计算 WOE 和 IV 值，且在这个过程中 B 方没有泄漏任何数据？



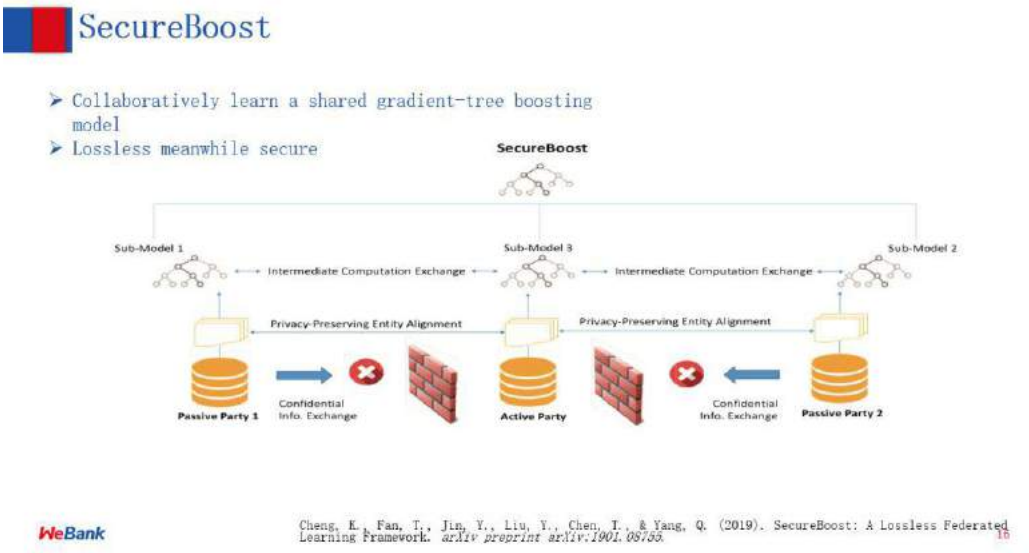
首先，B 方对 y 以及 1-y 进行同态加密，然后给到 A 方，A 方会对自己的特征进行分箱处理，进而 A 方在分箱中进行密文求和的操作，再把结果给到 B 方进行解密，然后算出 A 方每个特征分箱的 WOE 值和 IV 值。在这个过程中，没有明文数据传输，A 方不知道 B 方的 y 值，同时 B 方也不

知道 A 方每个特征的值是什么，从而在安全隐私保护的情况下，完成了特征工程的计算。

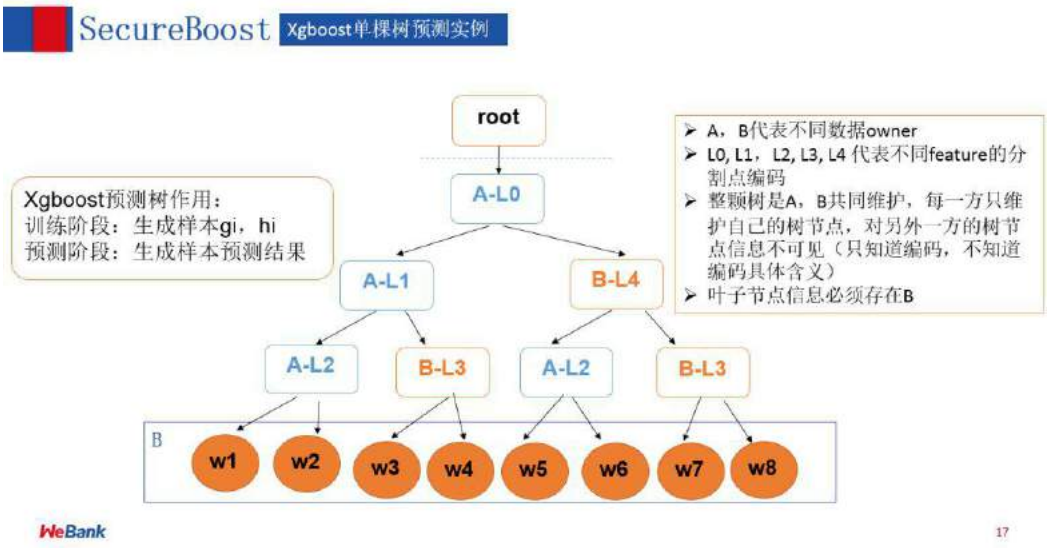


说完特征工程，再讲下最核心的机器学习，比如常见的逻辑回归，这是经典的 loss function 和梯度，刚才说的同态加密的特性，目前用到的是半同态的技术。所以，需要对 loss function 和梯度进行多项式展开，来满足加法操作。这样就可以把同态加密的技术应用在 loss function 和梯度中。

5. SecureBoost



在很多现实的业务应用中，树模型是非常重要的，尤其是 XGBoost，对很多应用来说，提升非常明显，因而被业界广泛使用。在联邦机制下，如何构建这样的树？这里我们提出了 SecureBoost 技术方案，双方协同共建一个 boosting 树，我们证明了整个过程中是无损的。图中为一些 paper 和链接，感兴趣的小伙伴可以查找下。



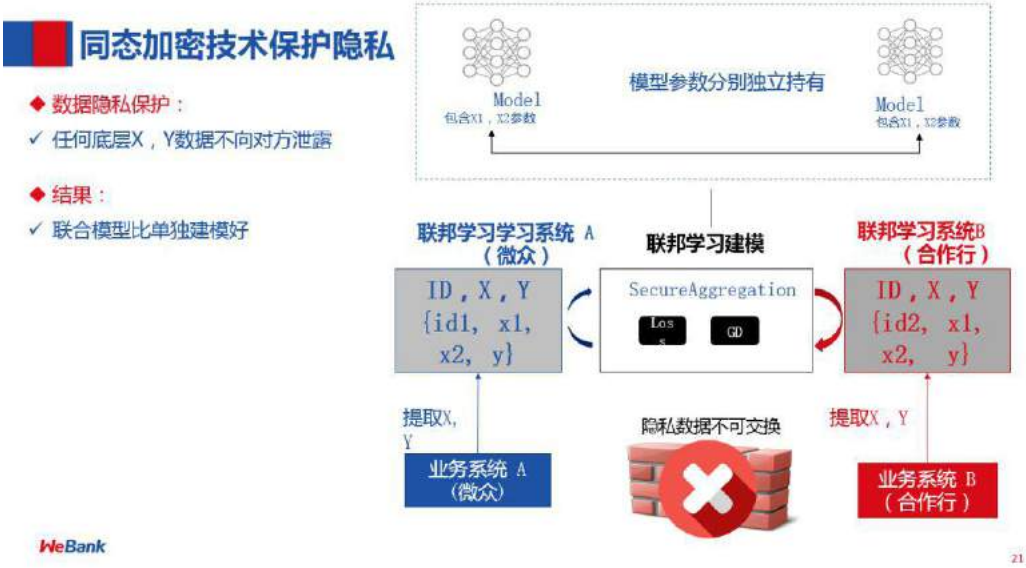
SecureBoost 的核心技术点。上图为基于 SecureBoost 构建的树，A 和 B 代表不同数据的 owner，L0，L1，L2，L3，L4 代表不同 feature 的分割点编码，整颗树由 A 和 B 共同维护，每一方只维护自己的树节点，对另外一方的树节点信息不可见（只知道编码，不知道编码具体含义），保证整个训练和预测过程都是安全的。

构建 SecureBoost 核心的关注点是如何构建分裂节点，尤其一方只有 X，另一方有（X，Y）的情况。

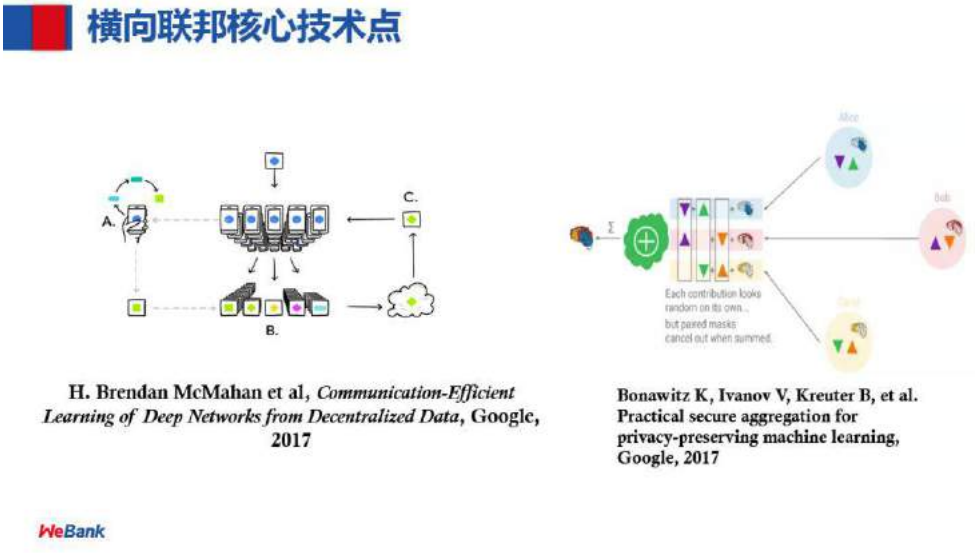
基于同态加密的机制，B 方会把 1 阶梯度和 2 阶梯度传递给 A 方，A 方基于分箱之后的结果算出每个分箱中的 1 阶梯度和 2 阶梯度的求和值，然后传递给 B 方，B 方会解密这个求和值，算出信息增益，然后给到 A 方。求分裂节点的核心就是如何算信息增益，通过这样的机制，就可以算出每个分裂节点，同时没有泄露任何隐私信息。

横向联邦学习

举个例子：微众和合作行共建反洗钱模型，期望优化反洗钱模型。因为，各自利用自家样本建立的反洗钱模型的效果和稳定性都不能满足现实需求。我们可以利用联邦学习的机制，充分利用多家的反洗钱样本，同时在不泄露样本的条件下，构建一个非常大的模型，可以看到横向联邦学习中，微众银行和合作行，都是有（X，Y）的。



技术层面上，采用了同态加密、Secret-Sharing 技术，整个过程中，双方交互的是模型和梯度，同时引入了 SecureAggregation 机制，让交互过程中的梯度也是很难被反解的。最终，大家都会得到一个相同的模型。横向联邦学习，综合多家样本，可以让模型更加稳健，效果更好。



下面分享下横向联邦学习的核心技术点，这是 Google 的两篇 paper ，第一个是每个端自己训练模型，然后发给云端进行综合各个模型的效果，但是这里面会有个问题，就是模型本身也可能会泄露隐私信息，所以引入了 SecureAggregation，各方在传输模型的时候会加一些噪音，通过云端来消除这些噪音，使数据和模型得到保护。

应用案例

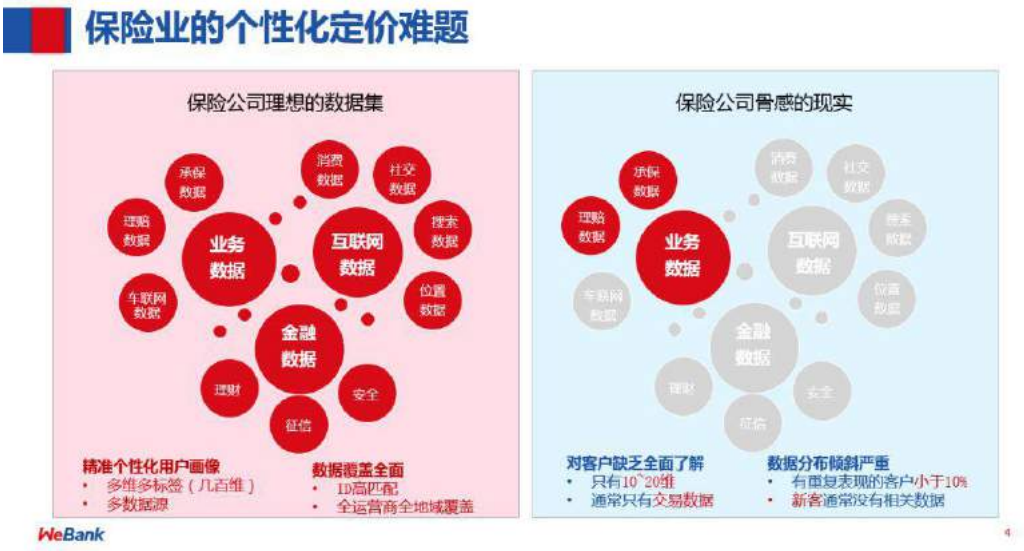
联邦学习目前已经赋能众多关键领域，取得了不错的效果：

- 银行+监管，联合反洗钱建模
- 互联网+银行，联合信贷风控建模
- 互联网+保险，联合权益定价建模

❖ 互联网+零售，联合客户价值建模

以其中的两个场景以及在视觉领域的应用来说明：

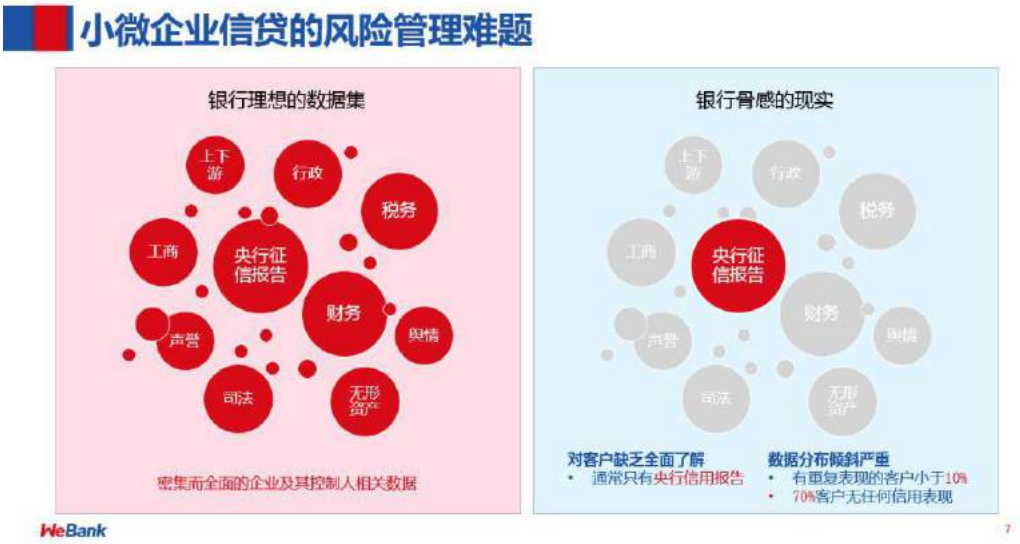
1. 保险业个性化定价



对于保险公司如果想做个性化的定价，是一件非常困难的事情，保险公司只有一些业务数据、承保数据和理赔数据，通过这样的数据，对用户做千人千面的定价是非常难的。通过联邦学习机制，可以融合多个数据源，来构建这样一个千人千面的定价。

这是我们目前在做的案例，当你购买权益产品之后，如果违章了，可以帮你免赔，在这里每个人看到的价格是不一样的，这就是我们通过联邦学习帮它们做的。我们综合了出险数据和互联网数据，如标签、用户画像，构建了一个基于保险定价的联邦学习产品。

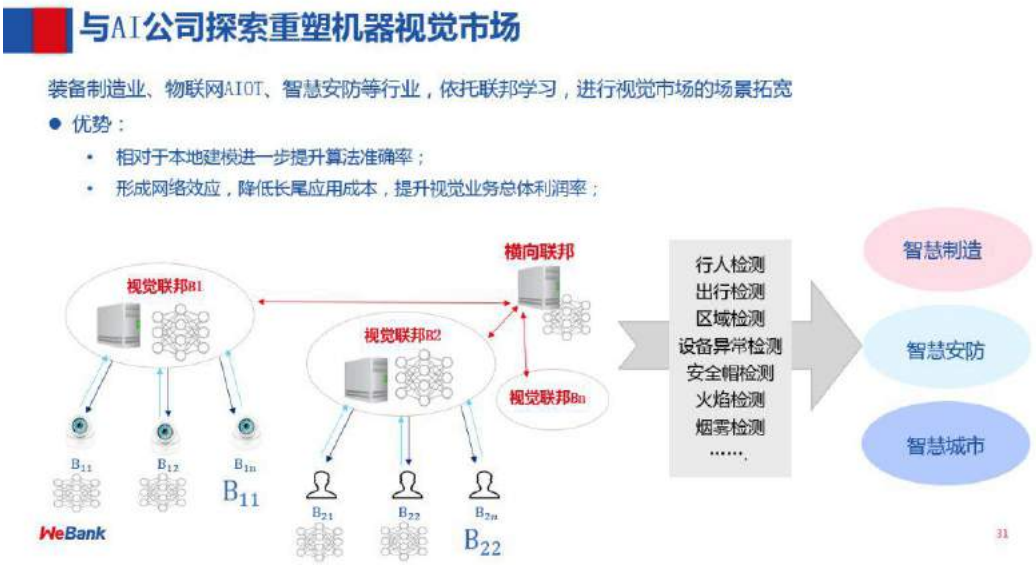
2. 小微企业信贷风险管理



很多时候对小微企业我们只有央行的征信报告，想要更精准的刻画企业的信用状况，需要更多的数据，比如发票、工商、税务的数据。

这是我们的一个案例，我们利用了开票金额和央行的征信数据共建了一个联邦学习模型，来预估每个企业的风险。

3. 联邦学习在视觉领域的应用



与 AI 公司探索重塑机器视觉市场，利用联邦学习的机制，相对于本地建模，进一步提升算法准确率，并且形成网络效应，降低长尾应用成本，提升视觉业务总体利润率。

这是联邦视觉在城市管理上的应用。

联邦学习开源平台：FATE

通过刚刚的介绍，相信大家对联邦学习能够做什么，有了大体的认识。而联邦学习想要落地，不可避免的就是开源，接下来为大家介绍下微众银行开源的联邦学习平台——FATE：

FATE 定位于工业级联邦学习系统，能够有效帮助多个机构在符合数据安全和政府法规前提下，进行数据使用和联合建模。

设计原则：

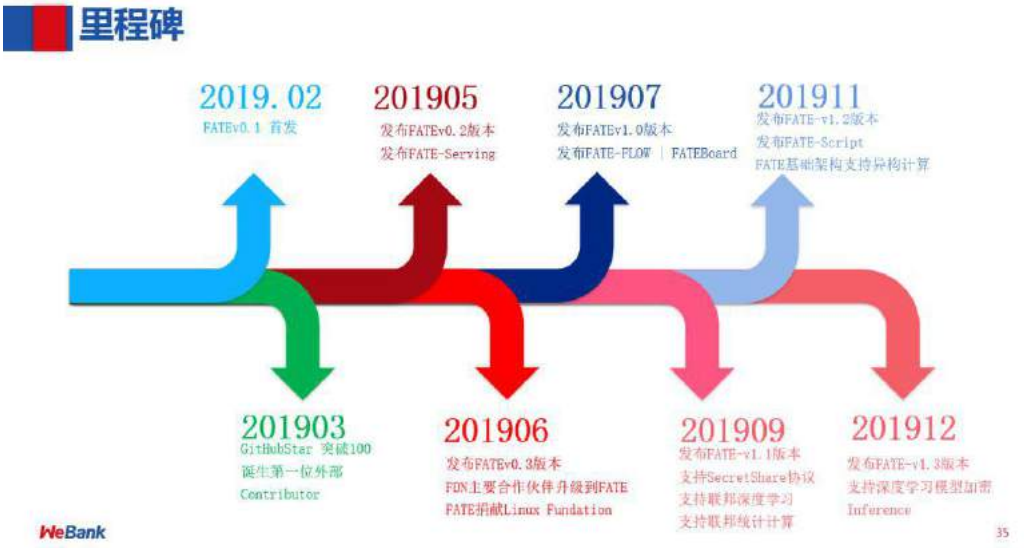
- ❖ 支持多种主流算法：为机器学习、深度学习、迁移学习提供高性能联邦学习机制。

- 支持多种多方安全计算协议：同态加密、秘密共享、哈希散列等。
- 友好的跨域交互信息管理方案，解决了联邦学习信息安全审计难的问题。

FATE 的 Github 地址：

https://github.com/FederatedAI/FATE

1. 里程碑



介绍下 FATE 的里程碑，FATE 在今年 2 月份首发了 0.1 版本，3 月份的时候有了第一位外部 Contributor，同时 GitHubStar 突破 100，5 月份发布了 0.2 版本，支持了联邦特征工程和在线推理，6 月份发布了 0.3 版本，把主要的合作伙伴迁移到了 FATE，并把 FATE 捐献给了

Linux Fundation，8月份发布了1.0版本，支持 FATE-Flow 和 FATE-Board，还有些后续的计划，大家可以了解下。

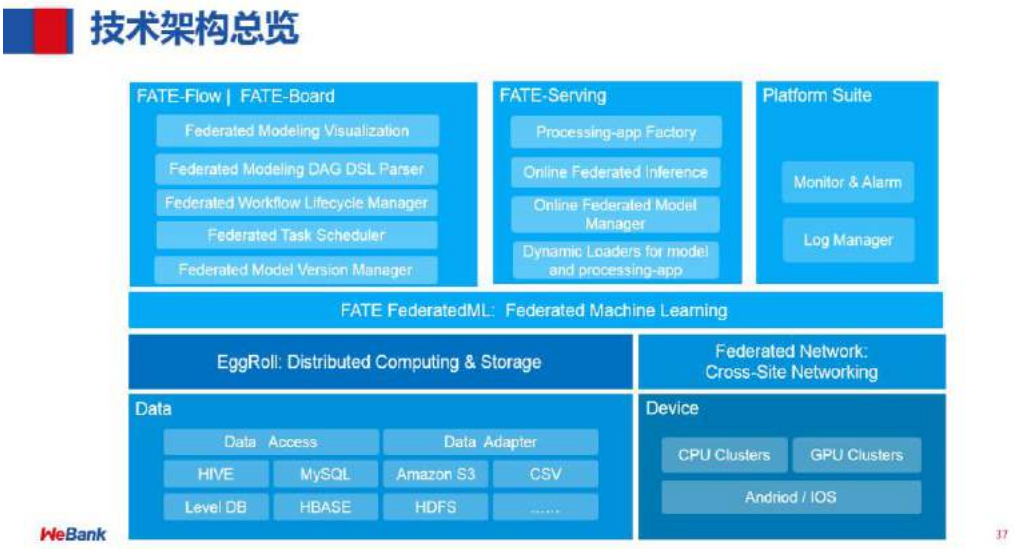
2. 挑战

联邦学习从一项技术真正成为一个关键系统和产品方案时，我们遇到了下述挑战：

- 一站式建模过程的联邦化
- MPC 协议下分布式算法（on WAN）易理解和易维护
- 跨站点数据传输安全性和可管理性，如何让交互部分是可以被管理和被审计的。

- 异构基础架构自适应，联邦学习可能会运行在 CPU、GPU 和端上，如何让上层不受底层的变化而变化。

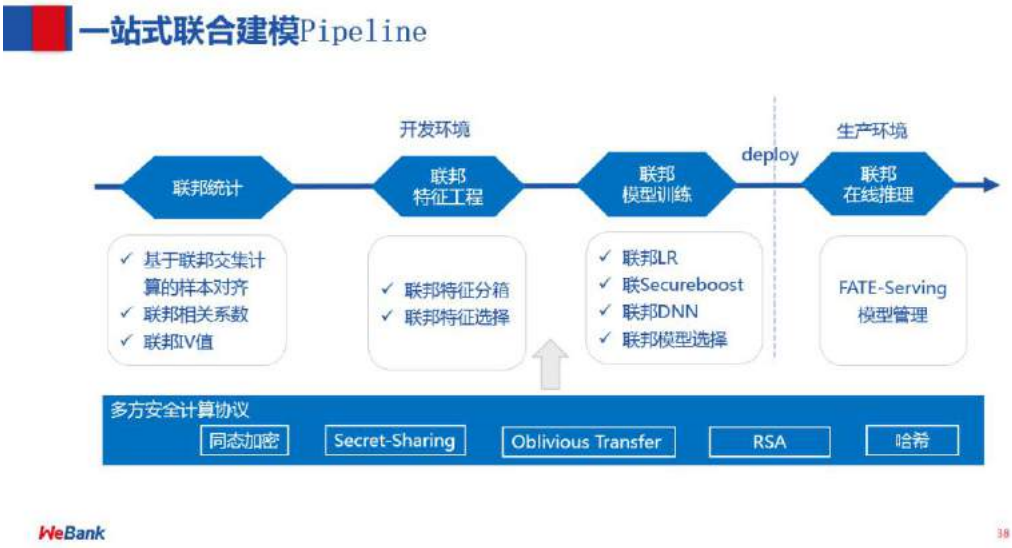
3. 技术架构



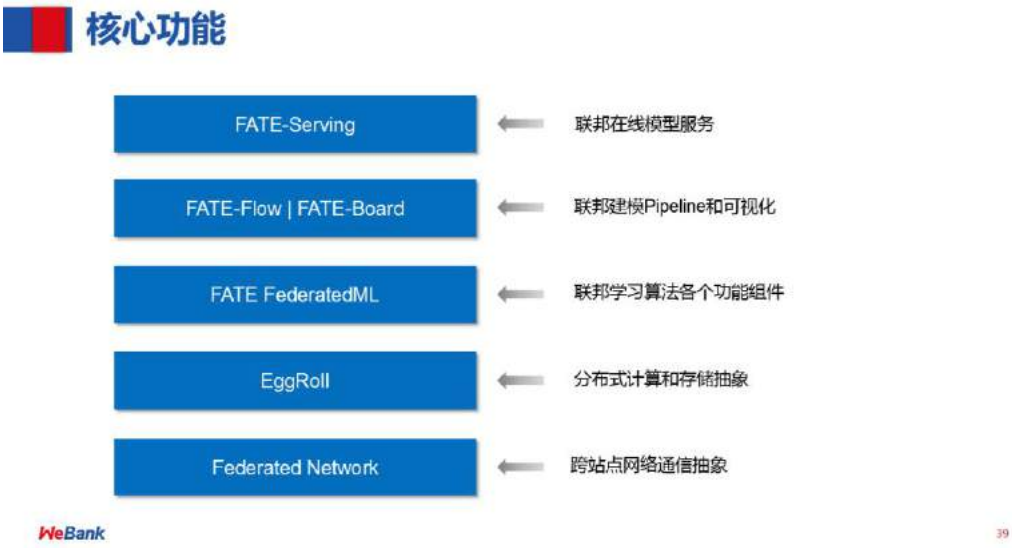
接下讲下 FATE 的整体架构：

- EggRoll：分布式计算和存储的抽象；
- Federated Network：跨域跨站点通信的抽象；
- FATE FederatedML：联邦学习算法模块，包含了目前联邦学习所有的算法功能；
- FATE-Flow | FATE-Board：完成一站式联邦建模的管理和调度以及整个过程的可视化；

❖ FATE-Serving：联邦学习在线推理模块。



一站式联合建模 Pipeline，其流程：在开发环境下，其流程是从联邦统计->联邦特征工程->联邦模型训练，当上线部署的时候会有联邦在线推理模块，底层则会采用多方安全计算协议去支持上层各种联邦算法。



这是 FATE 的五大核心功能，这里不再复述。

① FATE FederatedML

FATE FederatedML：

- EggRoll & Federation API：底层是 EggRoll 的算子，比如 Map 和 MapValues，Remote 和 Get 可以完成整个分布式计算的抽象；
- MPC Protocol：包括同态加密、秘密共享等多种多方安全协议，
- Numeric Operator：会抽象出数学算子，比如加法或者乘法；
- ML Operator：用建好的数学算子构建机器学习算子，而不用管底层的安全协议是什么；

❖ Algorithms：有了 ML 算子之后就构建各种算法模型。

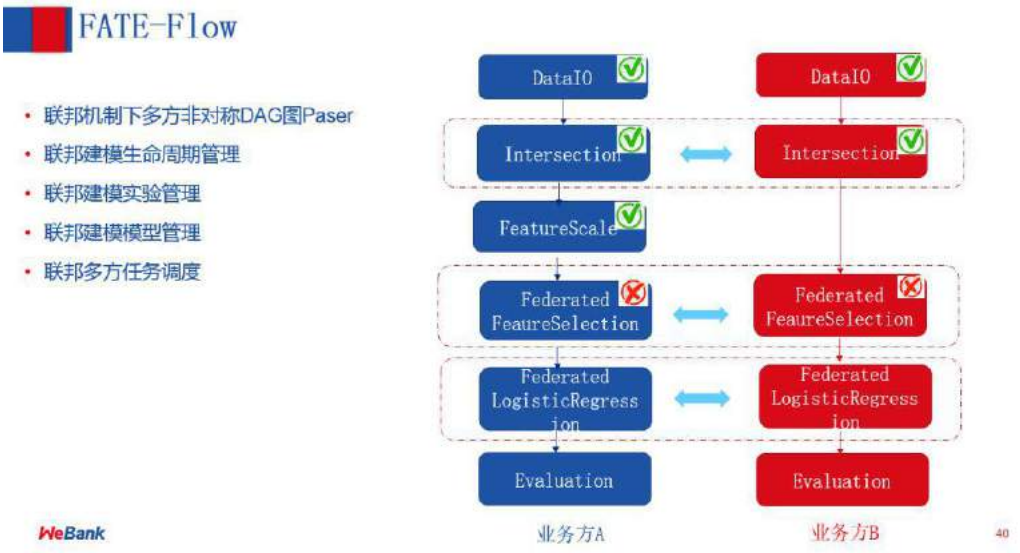
② EggRoll

EggRoll，是整个分布式计算和存储的抽象。面向算法开发者，通过 API 实现分布式计算和存储。上面为 EggRoll 的整体架构图。

③ Federated Network

Federated Network：联邦学习不仅需要分布式计算，还需要跨站点通信和交互，上层会提供一个 API 给到开发者，通过 Remote 和 Get 就可以完成数据点的收发。具体模块，如上图。

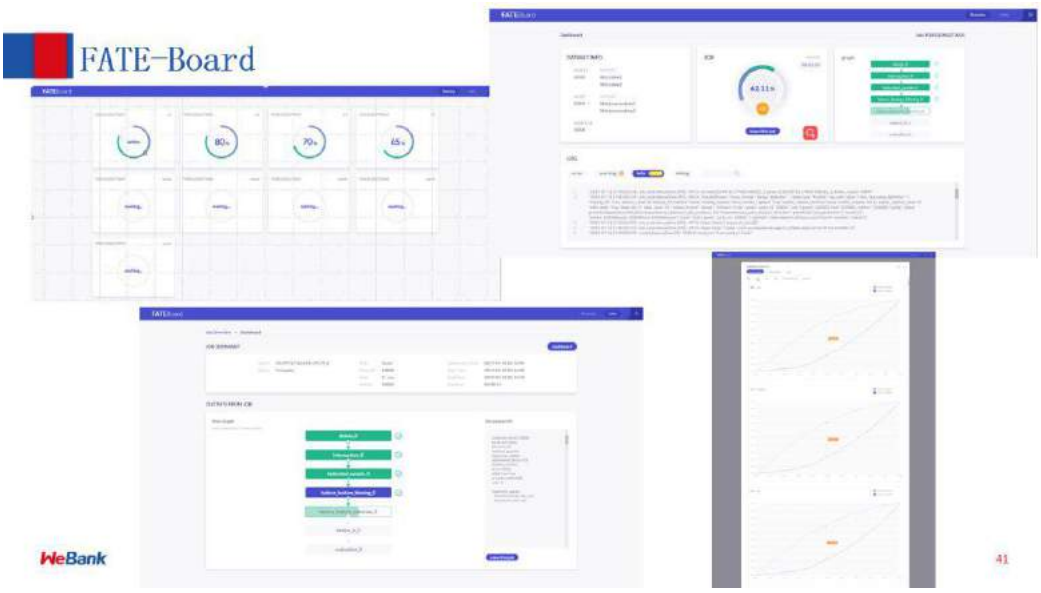
④ FATE-Flow



FATE-Flow：整个一站式联合建模 Pipeline 需要统一的调度管理。右边为 A、B 双方的建模流程，某些步骤是 A、B 双方共有的，某些步骤可能只有一方有，所以 FATE-Flow 完成了下述管理：

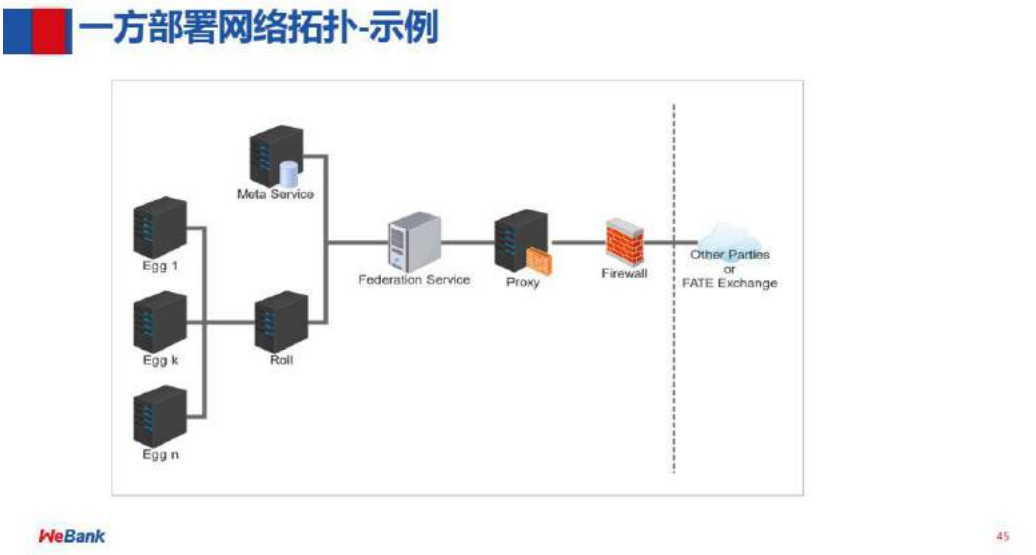
- 联邦机制下多方非对称 DAG 图 Paser
- 联邦建模生命周期管理
- 联邦建模实验管理
- 联邦建模模型管理
- 联邦多方任务调度

⑤ FATE-Board



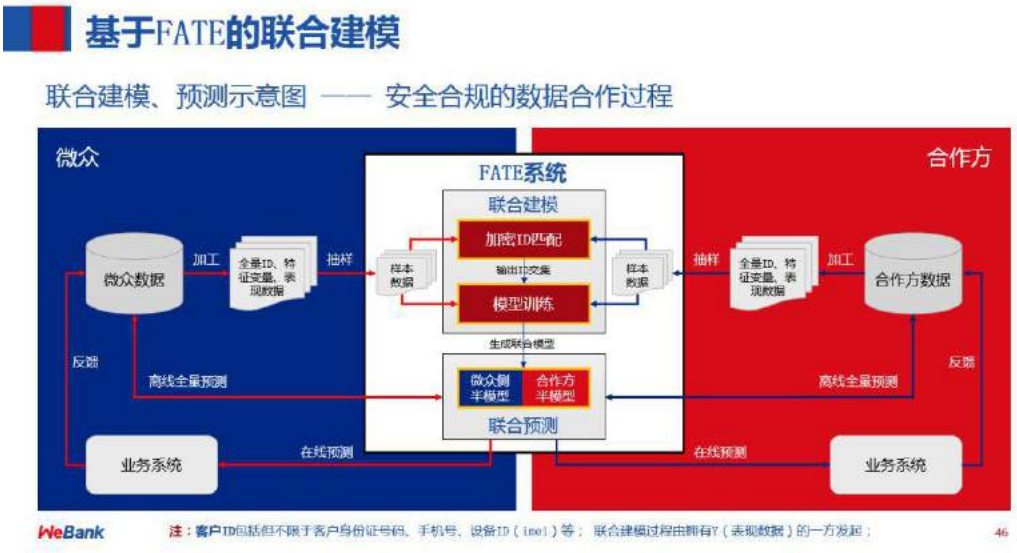
FATE-Board 的目的是实现整个联合建模 Pipeline 可视化追踪，记录联邦学习的全过程，可以监控建模进行到哪个步骤，以及模型效果如何。

4. FATE 部署架构



这是 FATE 的部署架构，每一方都是差不多的，是一个对称的结构，通过 EggRoll 实现分布式计算和存储，通过 Federation Service 和外部交互。

5. FATE 应用



现实中，FATE 是如何应用的呢？这里和大家分享一个示意图，会在每一方部署一套 FATE 系统，双方都是在各自部署的 FATE 系统中进行交互。

6. 开发流程

如果大家对 FATE 感兴趣，作为开发者利用联邦学习框架实现算法，只需要四步：

① 选择一个机器学习算法，设计多方安全计算协议

② 定义多方交互的数据变量

③ 构建算法执行工作流

④ 基于 EggRoll & Federation Api 实现算法工作流中各个功能组件

7. 目前 FATE 项目中的算法和案例

这是目前 FATE 项目中的算法和案例，会根据需求，不断增加各种各样的算法。最后分享下我们联邦学习的官网 https://www.fedai.org.cn/cn/

今天的分享就到这里，谢谢大家。

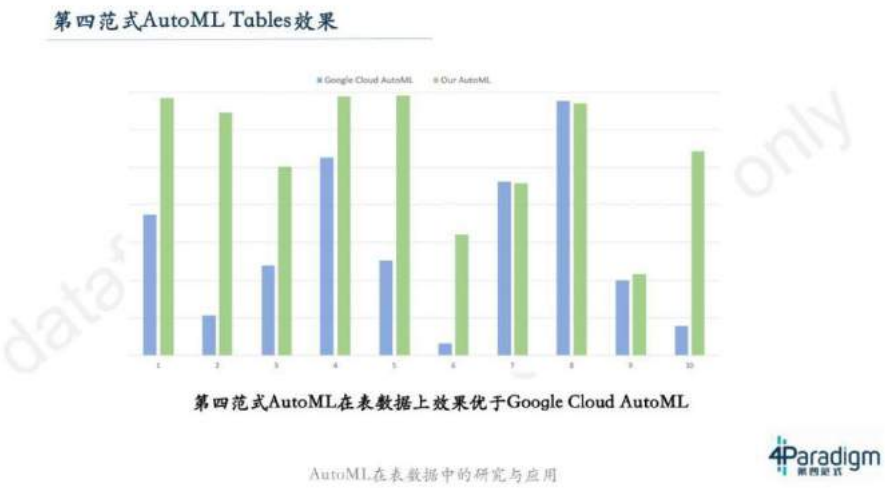
AutoML 在表数据中的研究与应用

嘉宾：罗远飞　编辑：Flora

导读：大家好，今天分享的题目是 AutoML 在表数据中的研究与应用。对于 AutoML，大家听到比较多的可能是神经网络结构搜索（NAS，Neural Architecture Search），NAS 主要应用于图像，而我们的工作主要应用于解决表数据（Tabular Data）中的建模问题。目前 NAS 在表数据中的研究较少，有兴趣的小伙伴可以尝试。

第四范式 AutoML Tables 的效果

我们选取了10个 Kaggle 比赛数据，分别通过第四范式 AutoML Tables 和 Google Cloud AutoML Tables 产生结果并提交，然后分别计算每种方法的在整个排行榜中的相对排名，如图所示，第四范式的 AutoML 在表数据上的效果大部分要优于 Google Cloud AutoML，其中图中青色代表第四范式，蓝色代表 Google，柱状图越高表示效果越好。



下面我主要从五个方面进行介绍：

- AutoML Tables 的背景
- AutoML Tables 的自动特征工程
- AutoML Tables 的自动参数寻优
- 第四范式 AutoML Tables 的落地案例
- 对未来工作的展望

AutoML Tables 的背景

1. 什么是机器学习？

Tom Mitchell 在 1997 年《机器学习》的教材中讲到，定义如下：A computer program is said to learn from experience E with respect to some class

of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

2. 什么是 AutoML？

机器学习步骤

以现实中的场景“反欺诈”为例，介绍下机器学习的步骤。

① 形式化的定义问题

② 收集数据

③ 特征工程

④ 模型的训练

⑤ 模型的评估

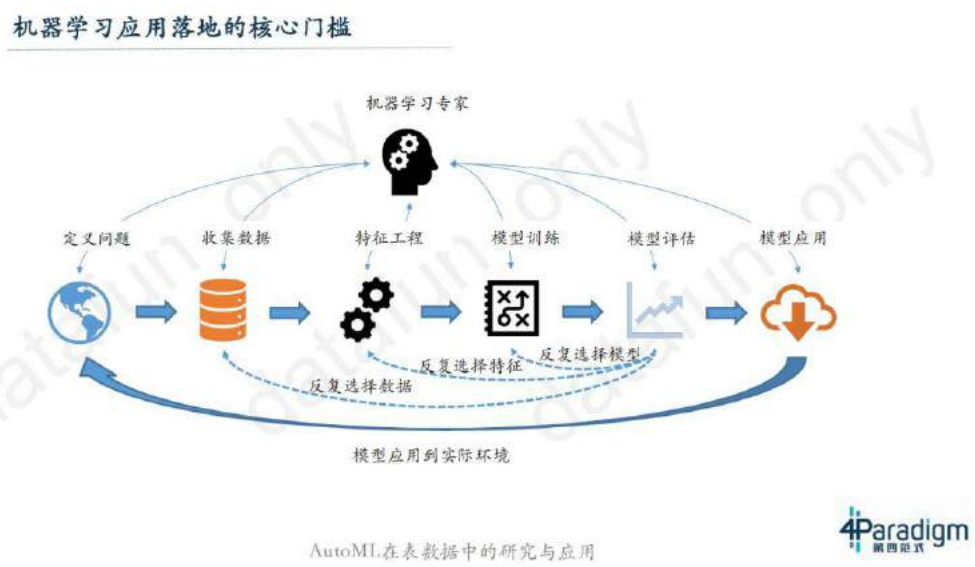
⑥ 部署和应用（如果效果达到目标）

上述所有的工作基本都是由机器学习的专家来完成的。该过程可能是一个迭代的过程，需要根据模型的效果，多次选择数据、特征和调整模型等。

AutoML 做什么？

AutoML（Automated Machine Learning）是利用机器来完成机器学习步骤中的某部分工作，而不是全部由专家来完成。目前，在相当多的领域缺乏有机器学习背景的开发人员。因此，利用 AutoML 可以使得机器学习更好的应用到更多的行业、更快地造福于社会。

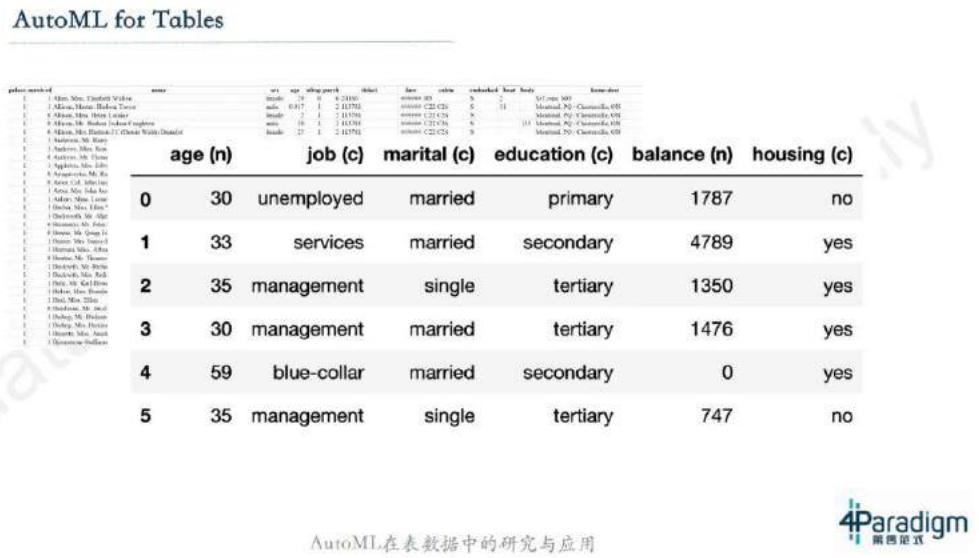
现在 AutoML 主要侧重于特征工程、模型训练和模型评估，对于问题的形式化研究较少。



3. 什么是 AutoML for Tables？

通常大家对于 AutoML 的印象更多来自于 Google 的 NAS 方面的文章，例如自动在 CIFAR10 或者 ImageNet 上搜索网络结构。这些通常是应用于图像数据的。

AutoML for tables 的工作主要是面对通过业务逻辑拼接成的宽表，不涉及图像数据或者 NLP。图中示例的数据集是来自 UCI 的数据，主要预测是否给客户贷款，是否有风险，用到的特征有年龄、工作、教育、资产等。



AutoML Tables 自动特征工程

根据前面提到的机器学习的 pipeline，我们先介绍 AutoML Tables 的自动特征工程。在参加竞赛中，大家会提到一个说法“特征决定了效果的上限，模型只是决定趋近于这个上限的程度”。在我们的实际工作中，特征也是一个提升空间更大的地方。

自动特征工程主要是由下面几个模块组成：

- 自动拼表
- 自动特征生成
- 自动特征选择
- 自动特征增强

1. 自动拼表

现实中完成一个业务场景的任务，是需要很多张表的。例如一张表描述用户信息，一张表描述商品信息，还有一张表描述其他的补充信息（例如上下文，浏览记录等）。

2. 自动特征生成

在自动特征生成中，我们主要是有下面四类的算子：

① 一元算子

基于特征做线性和非线性的变换，例如归一化、log 变换等。n 个特征，复杂度是 O(n)。

② 二元操作算子

例如加减乘除，笛卡尔积等。n 个特征，进行二元操作，则复杂度为 $O(n^2)$。

③ group-by 算子

Group-by operator 是比较特殊的一种算子，它需要先做 partition，然后做 aggregation。此外，如果牵扯到时序性（ 时序可能放在 group by，或者放在 window function），算子的复杂度会比较高，同时需要小心穿越。

④ 高阶的算子（high-order）

例如有 k 阶，则从 n 个特征中选择 k 个进行操作，数量为 C_n^k，再从这些特征中进行选择的话，就是指数的指数级，指数空间的搜索问题很难。

3. 自动特征选择

特征是不是越多越好？答案是否定的。一方面有的特征是有害的，另一方面特征较多对于系统性能都会增加要求。所以，在自动生成了很多特征后，我们需要考虑如何从自动生成的特征中选取有效的特征。

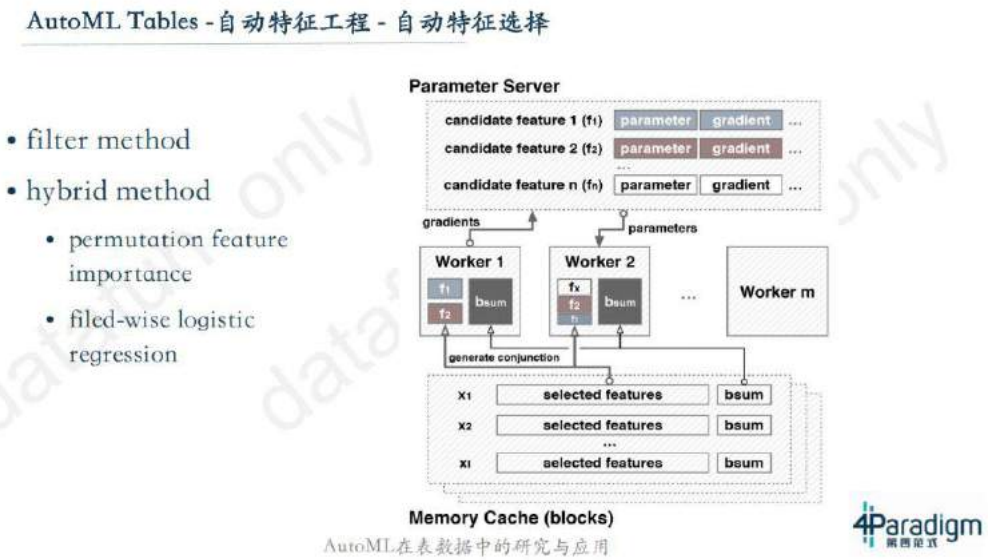
首先，最直观的做法是将特征放入模型中，如果模型效果好就保留，效果不好就舍弃。这个方法有两个问题，一是如果应用到全量数据，进行计算会比较贵；另一个是一个特征本身没有用，和其他的特征组合有用。

经典的特征选择主要有 Filter、Wrapper、Embedded 三种方式，可以去查看相关的资料。我们做的工作的目标是快速地，低代价的进行特征选择。这

次主要介绍一个 PFI（permutation feature importance）特征重要性的方法和（field-wise logistic regression）的方法。

PFI（permutation feature importance）方法

下面举例来介绍 PFI 方法，假设我们有 10 个特征待评估，首先进行一个模型训练，得到了模型的效果评估值（performance）AUC 为 0.8。然后，固定 9 个特征不变， shuffle 第一个特征，再次进行模型训练，得到新的评估值 AUC 为 0.7。判定特征的重要性为两次的差值 0.8-0.7，为 0.1。重复上面的过程，如针对第二个特征，固定除这个特征外的 9 个特征，只是对第二个特征进行 shuffle，这个时候模型为 0.75，则第二个特征的重要性为 0.8-0.75，为 0.05。该方法既可以做特征重要性，亦可做特征可解释性。该方法的动机是，特征越重要，对其扰动后，模型的效果抖动越大。



逐域对数几率回归（field-wise logistic regression，FLR）方法

当我们有 10 个特征，可分别用每个特征单独训练 LR 模型，但是这样代价比较高。我们主要进行了两方面的改进。

① 并行化改进

我们的改进之一是借鉴 boosting 的思想，并行来完成对于特征的建模。部分特征建模后，固定其权重，再进行剩余部分特征的建模。例如，当原始表有 100 个特征，又增加 10 个特征时，我们先利用 100 个特征训练的模型得到的特征权重，固定好这 100 个特征权重后，也即让 LR 有一个起始点，然后在这个起始点再去训练其他 10 个特征的权重。请注意：这 10 个特征的学习过程互相独立，即权重互不影响。

② 工程化改进

如果每次读取数据，只针对一个特征训练相应的 LR 模型，频繁 IO 导致性能下降。所以，我们结合参数服务器，实现扫描一次数据，训练出所有特征对应的 LR 模型，也即扫描一次数据，得到所有的特征重要性。

我们采用这两种方法得到特征重要性，然后进行迭代，最后得到有效的特征组合。

4. 自动特征增强

我们做的另一个工作是自动特征增强，由于里面仍然有很多问题较为困难，仍需要进一步研究。表数据中可能有各种数据，如 NLP 类型（一个 user 的 profile 为文本），image 类型（user 的头像），audio 类型。另外还有 graph 类型（例如考察团伙作案），以及 user 的住址等 Knowledge graph 的信息等。处理的方法一方面是进行直接处理（nlp 直接分词），另一种是 embedding 的方法，可以进行微调（fine tune），也可以不做。

我们最后将其应用到不同的数据集中，如下图。有的数据集上的效果很好，有的数据集上的提升效果较小，但也可以提升建模效果。

AutoML Tables - 自动特征工程 - 自动特征增强

• 实验结果

+ nlp / image		
Dataset	baseline	enrichment
Data1	0.7681	0.9169
Data2	0.8201	0.8420
Data3	0.6014	0.6185
+ knowledge graph		
Dataset	baseline	enrichment
Data4	0.6290	0.6757
Data5	0.8303	0.8441
Data6	0.6846	0.7004

自动特征增强可显著提升建模效果

AutoML在表数据中的研究与应用

4Paradigm 第四范式

5. AutoCross 介绍

下面介绍下我们在 KDD 2019 上面的一个工作：自动特征组合（AutoCross），详细可以参看文献 AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications，地址：

https://arxiv.org/abs/1904.12857

5.1 AutoCross 的系统设计

该系统包含从右到左三个部分：

① 基础设施

我们基于已有的大规模分布式的机器学习平台，搭建了一个并行处理机器学习算法。主要有参数服务器，cache 的优化， feature 的优化管理等。

② 算法集合

Beam Search 方法

基于 Beam search 解决了如何从原始特征选出 2 阶、5 阶乃至 10 阶的高阶特征生成与选择方法。主要是采用贪婪算法，首先进行 2 阶选择，然后将重要的特征放入下一步，逐次迭代生成多阶特征组合。形象地解释，beam 就是一束光，光照到的就会进行特征验证，没有照到的就不会被衍生下去。

逐域对数几率回归方法（field-wise logistic regression，FLR）

如前文所述，它可以快速地选择特征。

连续批训练梯度下降（successive mini-batch gradient descent）

即使有 50 个原始特征（ 现实较为常见 ）两两组合后特征就上千，在全部数据上扫一遍会代价很高。为了加速迭代，我们借用调参算法的思想，将评估特征逐步砍半，降低评估代价。

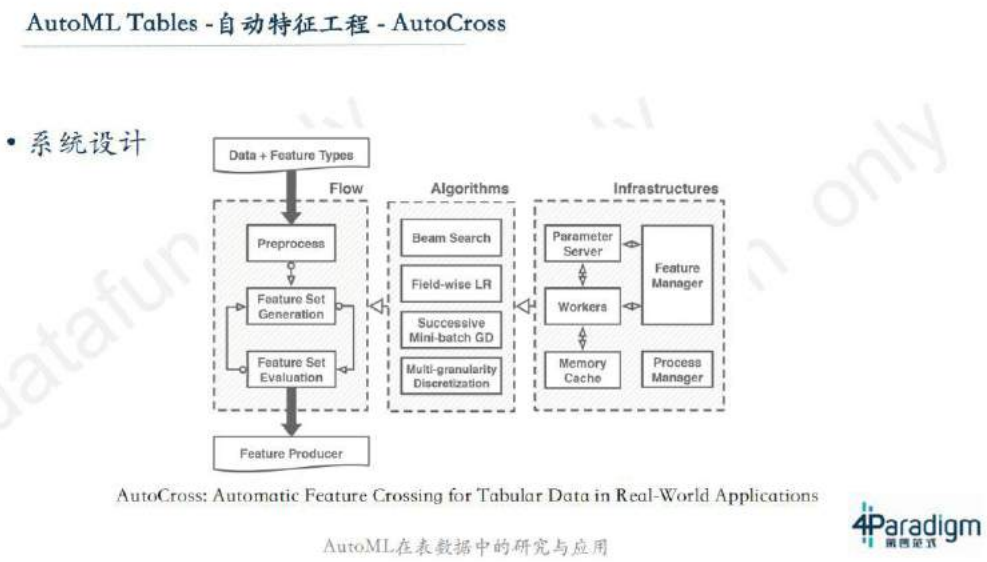
多粒度离散化（multi-granularity discretization）方法

当组合离散特征和连续特征进行建模的时候，需要进行连续特征变换。传统的方法是 log，或者分桶。我们在实际过程中发现，在一些数据集合上，连续特征离散化对于桶的个数很敏感（ 效果有 5%的差异 ）。为此，我们采用多粒度的离散化方法，对数值特征，根据不同粒度做离散化，同时生成多个

不同粒度离散化对应的离散特征，然后采用逐域对数几率回归挑选出最优的离散特征。多个划分粒度既可以由用户指定，也可以由 AutoCross 根据数据大小和计算环境来自适应地选择，使得用户不需要反复调整离散化的粒度。

③ 工作流实现

对数据进行基本的预处理后，我们将特征组合和特征选择迭代，最终选出来 Top-K 个的特征。



5.2 AutoCross 的效果

基于 5 个公开的数据集和 5 个实际业务数据集，我们验证 AutoCross 的效果。数据集范围从几万到几千万。

① 线性模型

首先对比利用生成特征和无该特征的建模效果的差异性，具体如下面的 LR（base）和 AC+LR 的对比，可知 AutoCross 可以提升线性模型的建模效果。

② 非线性模型

此外，可以发现 AC+W&D 结合后，模型效果不比 xDeepFM，因此 AutoCross 对非线性模型（DNN）也可以提升效果。

AutoML Tables - 自动特征工程 - AutoCross

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
LR (base)	0.9400	0.9169	0.8292	0.8655	0.7855
AC+LR	**0.9455**	**0.9280**	**0.8567**	**0.8942**	0.8034
AC+W&D	0.9420	**0.9260**	**0.8623**	**0.9033**	**0.8068**
CMI+LR	**0.9431**	0.9153	0.8336	0.8901	0.7844
Deep	0.9418	0.9130	0.8369	0.8745	0.7985
xDeepFM	0.9419	0.9131	0.8358	0.8746	**0.8059**
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
LR (base)	0.8368	0.8356	0.6960	0.6117	0.5992
AC+LR	**0.8545**	**0.8536**	**0.7065**	**0.6276**	0.6393
AC+W&D	**0.8531**	**0.8552**	**0.7026**	**0.6260**	**0.6547**
Deep	0.8479	0.8463	0.6936	0.6207	0.6509
xDeepFM	0.8504	0.8515	0.6936	0.6241	**0.6514**

AutoCross可显著提升线性模型（LR）和非线性模型（DNN）的建模效果

AutoML在表数据中的研究与应用

4Paradigm 第四范式

AutoML Tables 自动参数寻优

关于超参数优化，常用方法有 Random search（方法较为简单，而且效果够好），论文中常用的 Grid search 方法，以及贝叶斯优化方法。我们这次不再讲解，主要介绍随机坐标收缩、连续减半算法、基于种群的优化和双层优化四种方法。

超参数寻优过程主要面临下面两个问题：

用什么样的模型刻画超参数空间和效果，例如贝叶斯模型为高斯过程的，random search 是 Model free 的。

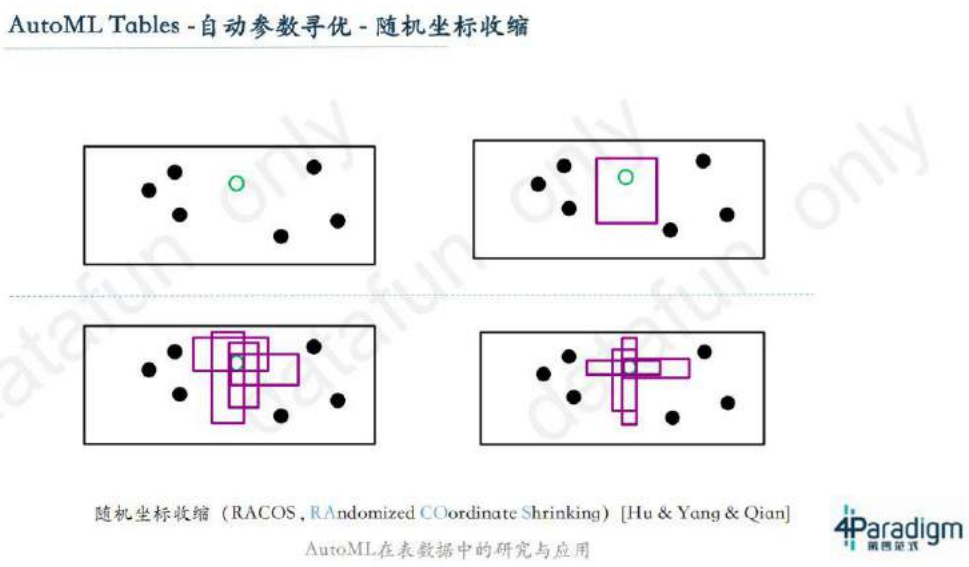
当有了模型效果评估后，如何选取采样策略，生成下一个组采样点和下一组超参数配置。

1. 优化算法

① 随机坐标收缩（RAndom COordinate Shrinking, RACOS）

该方法基于分类的思想，利用分类器将一个较大的超参数空间分成好坏两部分。其将一个超参数的组合定为一个坐标，操作过程中，每一维是独立的，操作显示为图中的矩形方式。其中，离散值是要或者不要该点，连续值为左

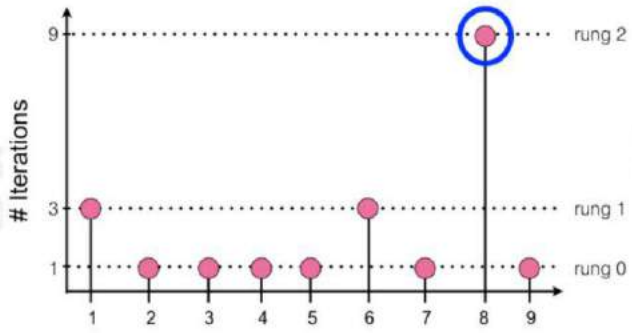
移或者右移。在划分为好的超参数区间以一定的概率进行采样，同时兼顾探索和开发，随后逐渐将这些坐标收缩到某一点。具体为，在最小化的过程中，逐渐缩小分类器的阈值，逐渐缩小好的参数空间的范围。



② 连续减半算法（SHA）

该方法思想朴素，然而非常有效。例如有 30 组参数，先在十分之一的数据上运行，选择出最好的十组，相当于砍掉了三分之二。然后利用这十组的数据，再跑一定的数据。如图，后面的数据只跑了一轮，所以它的代价一般较小。

AutoML Tables - 自动参数寻优 - 连续减半

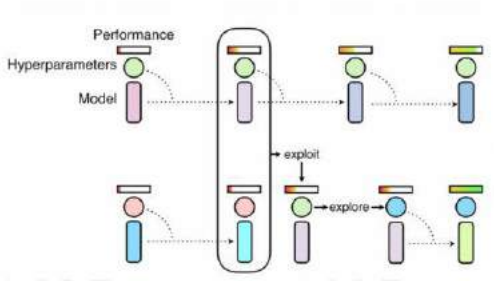


连续减半算法（SHA，Successive Halving Algorithm）[Jamieson & Talwalkar]

③ 基于种群的优化

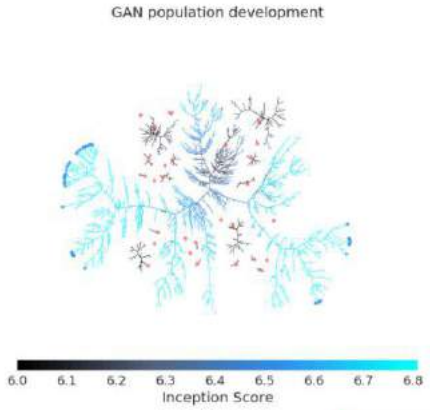
PBT 主要有提前停止、热启动和分支限界三个特征。右边的图中，红色的点为起始点，组成了初始的种群，好的点会进行交叉变异等操作，进行生长直到生成下面蓝色的点（ 效果比较好 ）。

AutoML Tables - 自动参数寻优 - PBT



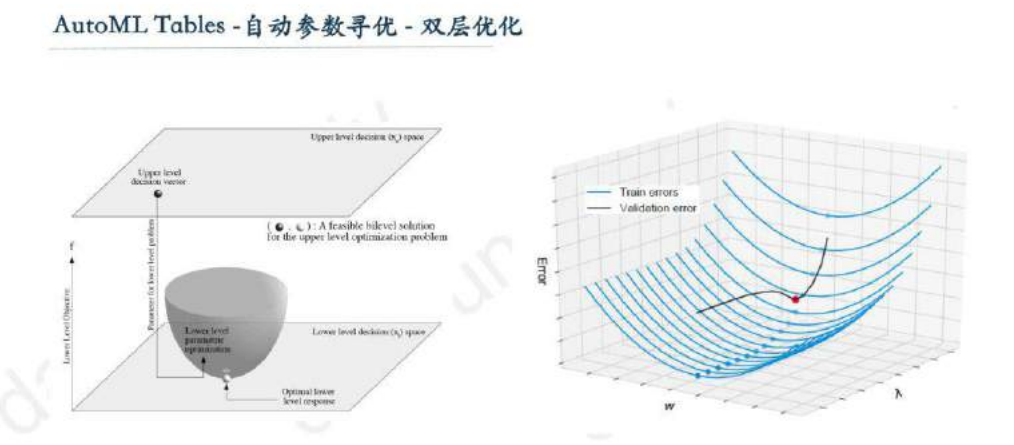
提前停止 + 热启动 + 分支限界

[DeepMind]



④ 双层优化

该方法属于层次优化的范畴。具体来说，机器学习会在训练集（training set）求得模型的参数，在验证集（validation set）上优化超参数。可以将训练集中对模型本身参数的选取作为一层，超参数在验证集上的优化作为一层，两层可以进行交互迭代。但该方法一个问题显著问题是：两层优化需要求二阶导数，会使得计算代价较高，对此也有些近似的方法可以替代。



双层优化（bilevel optimization）[Sinha et al. 2017; Franceschi et al. 2018]

2. 算法实例-AutoDSN

我们针对推荐等大规模稀疏数据，构建深度稀疏神经网络（DSN，Deep Sparse Network）方法。由于神经网络对于参数敏感，我们针对深度稀疏网络利用前面提到的方法进行自动超参数寻优。自动调整神经网络

（AutoDSN）在五个数据集上面的效果都较好。在 3-9 倍代价内，可以达到专家调参效果的 99%以上。

Dataset	Wide & Deep	NFM	Expert-DSN	Auto-DSN	Auto / Expert (%)	cost
Data1	0.8331	0.8290	0.8542	0.8533	99.9	8.85
Data2	/	/	0.8724	0.8733	1.00	5.46
Data3	0.7010	0.7042	0.7099	0.7088	99.8	4.73
Data4	0.8033	0.8005	0.8078	0.8066	99.9	3.61

AutoML Tables 落地案例

1. 学习圈理论

公司基于库伯经典学习圈提炼了机器学习的闭环流程。库伯的学习圈指的是人有理论后可以指导行动，行动有反馈，也即生成了经验，不断反思可以补充完善理论。这个对应机器学习就是收集行为数据，得到反馈数据，不断进行模型训练，然后进行模型的应用，模型应用后又可以收集数据。

我们基于学习圈理论做了一个 AutoML 产品 ,如左边为一个学习圈的全流程闭环设计图。它可以支持一键建模，也支持 AI 模型的可解释和特征可解释性，并可以进行数据自动回流和指标自动计算等。

2. 实际应用场景

我们利用闭环的机器学习的产品给一个大型的互联网公司做了一个项目，目前看项目的效果不错，具体见图中描述。

AutoML Tables 展望

下面主要考虑未来我们还可以做什么，主要包括效率、效果和交互式三个方面。

1. 效果和效率

AutoML 是需要同时追求效果和效率的，一要降低成本，二要提高效果。如果不达到一定建模效果，那就不可以用。但如果为了达到一定的效果代价太高，也是不可行的。效果方面，从问题定义到模型上线都有可以优化的点。效率方面，一是可以应用更好的硬件，软硬一体，另一方面可以优化算法，例如通过特征空间划分等提高效率。

2. 可交互

当前的 AutoML 比较封闭，人的参与度不高。我们希望可以达到人需要的时候 AutoML 帮助我们，不需要 AutoML 时用户自定义，来提高建模效率。当人在建模流程中，不想要做某方面的选择，可以给定目标函数、输入，让

机器来优化（ 机器较为擅长 ）。在优化的过程中，人可以随时介入到学习过程中，提前终止或修改搜索空间和方向。

强化学习：原理与应用

嘉宾：王凡　编辑：王成林

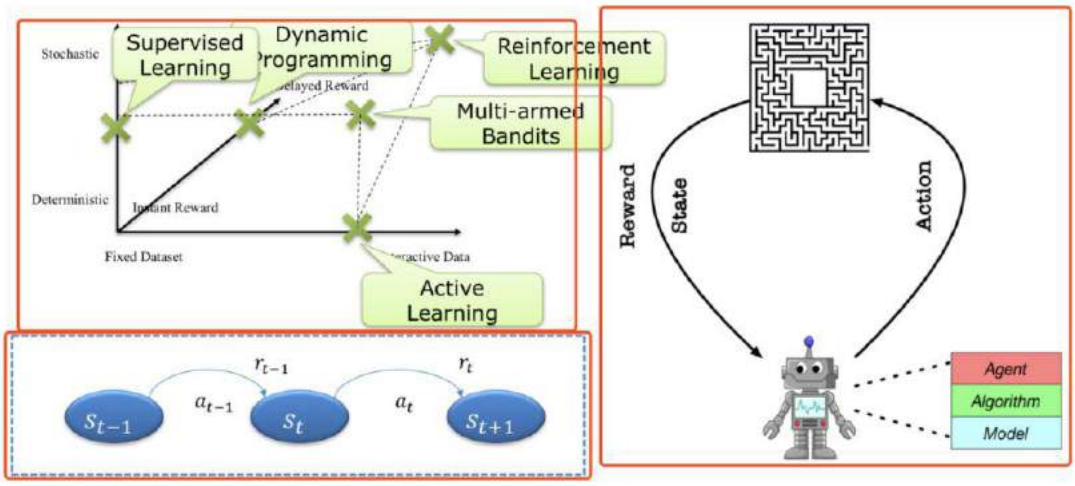
今天分享话题主要是跟强化学习有关，因为强化学习最近在工业界是非常火热。从加入百度开始，我主要就是做这一块。强化学习是一个比较大的话题，分享的内容大概是三个部分：1）强化学习的基本算法；2）部分百度内部的强化学习应用；3）最后是强化学习的工具介绍，最近团队刚发布一款工具叫做 PARL，是 PaddlePaddle 的一个强化学习库。

Algorithms

Tools & Platforms

Applications

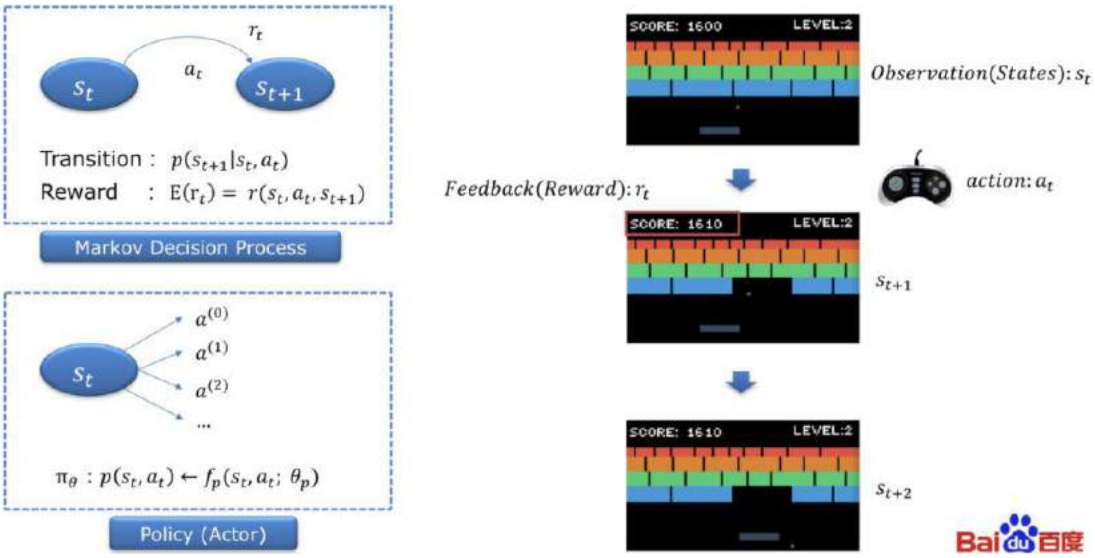
算法部分



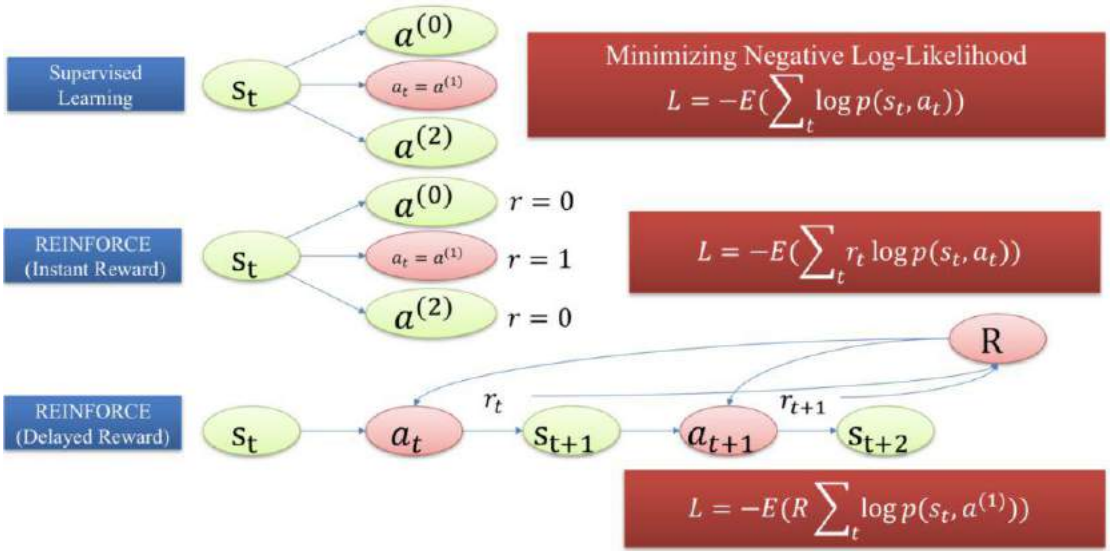
上图右边部分是最能说明强化学习的过程：一个机器人（ agent ），它会跟环境（ environment ）有一个交互，其中 agent 是通过 action 来影响 environment，environment 会返回 reward 和 state，这个交互过程会不断地交互迭代。上图左下角部分描述以上迭代过程，我们称之为马尔可夫决策过程。

如果熟悉有监督学习，但是不熟悉强化学习，可以通过“三轴”图（图中左上角部分）来看强化学习与之联系和区别：第一条轴 deterministic-stochastic 描述转移概率，决策过程和奖励值分布的随机性。第二条轴 fixed dataset - interactive data，表示的是学习数据的来源和分布规律。最后一个轴是 instant reward- delayed reward，该轴表示奖励值是立即的还是有延迟的。 我们熟知的有监督学习，针对的是静态的学习数据，以及近似可以看作无延迟的奖励。动态规划针对的则是确定性的环境，以及静态的数据。主动学习则针对无延迟的奖励以及确定性的数据。多臂老虎机(multi-armed bandits ）则同时处理随机性和交互类型的数据，但依旧是无延迟的奖励问题。只有强化学习处理三个方向（随机，有延时，交互）的问题，因此，他能够解决很多有监督学习无法解决的新问题。

举个例子如雅利达游戏，状态（state）指看到当前电游屏幕上，包括图像还有其他各种信号信息。当人或者机器看到一个 state 可以有相应的 action。这个 action 就是游戏手柄的操作。然后进入下一个过程。整个过程可以用马尔可夫决策过程来描述。马尔可夫决策过程有两个要素，一个是 transition 代表环境转移的概率，另外一个 reward。



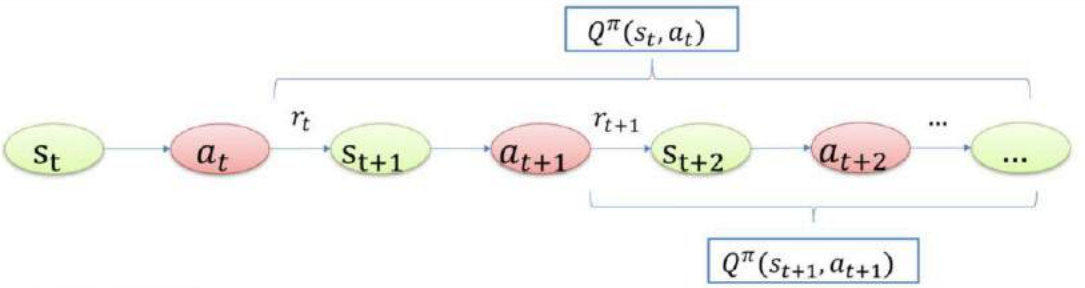
从有监督和强化学习两个学习目标的角度来理解以上过程，在有监督学习过程中，在 t 时刻 action 我们称为 a_t。在有监督的学习过程中需要人来示范动作，我们希望机器去学习人的示范动作，那么这个人的示范就成了 label。我们通过最小 Negative Log-Likelihood (NLL)来使得我们的 policy 函数逼近人的示范。



假设没人来示范，让机器自己想办法去做一个动作。如果只 instant reward，机器人学习到 $a^{(1)}$ 这个动作。但是它不知道自己做的对不对，但是可以拿到一个 reward。如果我们用 $L = -E(\sum_t r_t \log p(s_t, a_t))$ 来迭代它，假设这个动作做对了会拿到 +1 的奖励，做错了奖励是 0，我们可以发现，这个式子跟 $L = -E(\sum_t \log p(s_t, a_t))$ 是一致的，也就是我们回到了有监督学习。

现在，更进一步，在马尔可夫决策过程迭代中，学习目标是一个序列过程，而不是一步结束。此时就把 reward 提取到公式的外面来，即 $L = -E(R \sum_t \log p(s_t, a_t))$。即整个序列完成好的话，就强化整个学习序列，反之弱化整个序列。这个策略便是 REINFORCE，是 92 年提出来的一个策略。这样，我们就从上面单步的 REINFORCE 扩展到了多步。

接下来介绍另外一种学习策略目标是最大化长期收益，序列过程如下图最上面所示。定义一个 Q 值，是从某个 t 开始，一直到最终结束的回报总和。这种序列可能无限长，为了不致于使得 Q 值没有清晰的定义，我们引入 gamma（衰减系数）。



Value Function (Critic)

$$Q^{\pi}(s_t, a_t) = E(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots)$$

$$Q^{\pi}(s_t, a_t) = \mathrm{E}[\gamma\, Q^{\pi}(s_{t+1}, a_{t+1}) + \mathrm{r_t}]$$

Bellman Equation, Learning Target

此时的 Q 值（value funciton）为 $Q^{\pi}(s_t,a_t)=E(r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+...)$。如果写成递归方式，即为 $Q^{\pi}(s_t,a_t)=E(\gamma Q^{\pi}(s_{t+1},a_{t+1})+r_t)$，这其实就是简化的贝尔曼公式。反过来想只要满足这个公式，就是想要的 Q 值。因此，我们要想办法求出这个 Q 值，或者说学习这个 Q 值对应的参数，就可以想办法逼近这个公式，也就是这个公式变成了目标函数。我们正式地写出这个学习公式。这分为两种情况，一种是离散过程，另外一个是连续过程。

Discrete State Space:

$$Q^{\pi}_{k+1}(s_t, a_t) = Q^{\pi}_{k}(s_t, a_t) + \alpha\,(\gamma Q^{\pi}_{k}(s_{t+1}, a_{t+1}) + \mathrm{r_t} - Q^{\pi}_{k}(s_t, a_t))$$

Continuous State Space: $Q_{\theta}(s_t, a_t) \rightarrow Q^{\pi}(s_t, a_t)$ Function Approximator

$$L(\theta) = E[\sum_t \left(\gamma Q_{\theta'}(s_{t+1}, a_{t+1}) + \mathrm{r_t} - Q_{\theta}(s_t, a_t)\right)^2]$$

Temporal Difference Error

在离散过程中，state 和 action 都是有限的，用目标 Q 减去当前 Q 作为误差。让 Q 值逼近想要目标。连续情况会比较复杂，需要一个 model 或者 function 来逼近它，此时 Q 值是一个模型，theta 是参数，state 和 action 是特征，比如深度学习中神经网络模型，此时模型的目标函数是希望贝尔曼误差最小，这就是 temporal difference error。目前用得比较多的 DQN 仍然无法解决 Action 是连续的情况，为了解决 action 连续的情况，我们还是需要引入 policy function。但是，REINFORCE 这类直接求解 Policy Function 的方法，效率通常比 DQN 要低很多。举例来说，比如说下围棋前 99 步都下得很好，最后一步下的不好，导致满盘皆输。但是如果我们用 REINFORCE 方法，我们认为这 100 步都不好，包括其中下得其实不错的前 99 步。虽然从统计意义上来说，如果下的次数足够多，梯度更新最终仍然是会往正确的方向，但毋庸置疑，即这是一种学习效率很低的算法。为了改善这个问题，另一类应用更广的方法是 actor-critic。actor 是 policy，critic 是评价函数。 我们希望评价函数做一件事情，不论结果怎么样，都会给 action 返回一个合适针对它自己对于整个棋局影响的“综合”反馈，这个所谓的综合反馈在学术界又叫做 credit assignment。REINFORCE 是可以认为用整个序列的总体 Reward 作为 credit。 而 actor-critic 赋予每一个动作一个独特的反馈 Ψ_t，叫做 advantage，它能够针对 action 做一些独立的反馈。下面列出的是一些不同的 advantage 的计算方式。

$$L = -E[R\ \Sigma_t \log(p(s_t, a_t))]$$

Advantage

$$L = -E[\Sigma_t \Psi_t \log(p(s_t, a_t))]$$

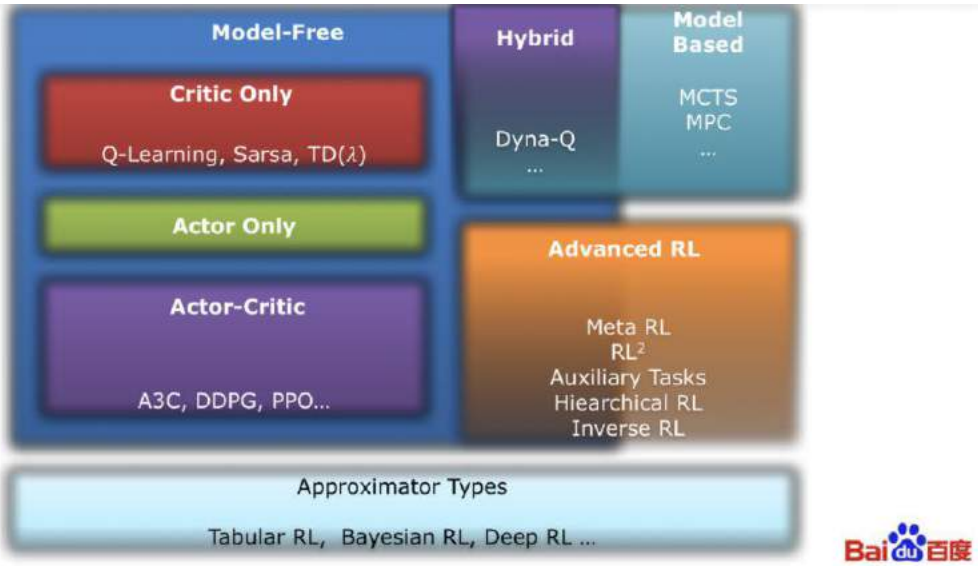
1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^{\pi}(s_t, a_t)$: state-action value function.
5. $A^{\pi}(s_t, a_t)$: advantage function.
6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual.

强化学习涉及的算法范围非常广，包括：

1）model-free 算法， 这类现在应用比较多，它又可以分为 critic only，actor only 和 actor-critic。

2）model based 算法，通过对环境建模来进行规划，这类算法最近逐渐获得越来越多的关注。

3）还有一些和不同算法结合在一起的 advanced RL，如 Auxiliary Tasks 可以学习一些稀疏的任务。还有反向强化学习，元强化学习等等。



应用部分-推荐系统

2011 年之前强化学习是处于一个不温不火的状态。主要用到强化学习的是 Intelligent Robotics，用在控制问题，因为控制问题特别适用于强化学习。2012-2013 年，强化学习是伴随深度学习的高涨逐渐火热起来， 这一年 openAI 发布了 Gym 环境。就百度来说 ,2012 年的时候 ,百度开始用 multi-armed bandit 在排序和搜索上面。2014-2017 年百度在工业界进行探索，有一些成功和失败的经验。百度早在 12 年也成立深度学习研究院，其中一项工作是发布了 XWorld，这个作为一个通用 AI（AGI）环境在学术界引起了不小关注。到 2017 年，百度有不少强化学习应用，例如凤巢的定价系统，还有 feed 流的推荐。

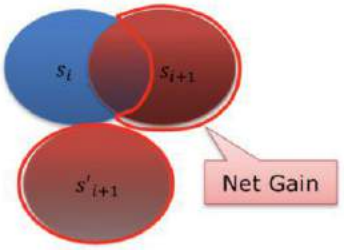
强化学习在百度的一个重要应用是 feed 流推荐系统。工业界的推荐系统最早使用方法是协同过滤，追求用户对一个新闻的偏好估计。现在的推荐系统有两个研究热点：

1）intra-list correlations：考虑的是列表页多样性，即列表里的内容关联。举例来说，在同一个页面一次展现多个相同或者类似内容，哪怕用户喜欢其中单独每一个，也不一定能接受。

2）dynamic evolution 或者 inter-list correlations：考虑的是列表页之间关联，即推出的一个新闻不仅仅于点击与否，还需考虑点击行为对接下来的推荐影响，例如头条或

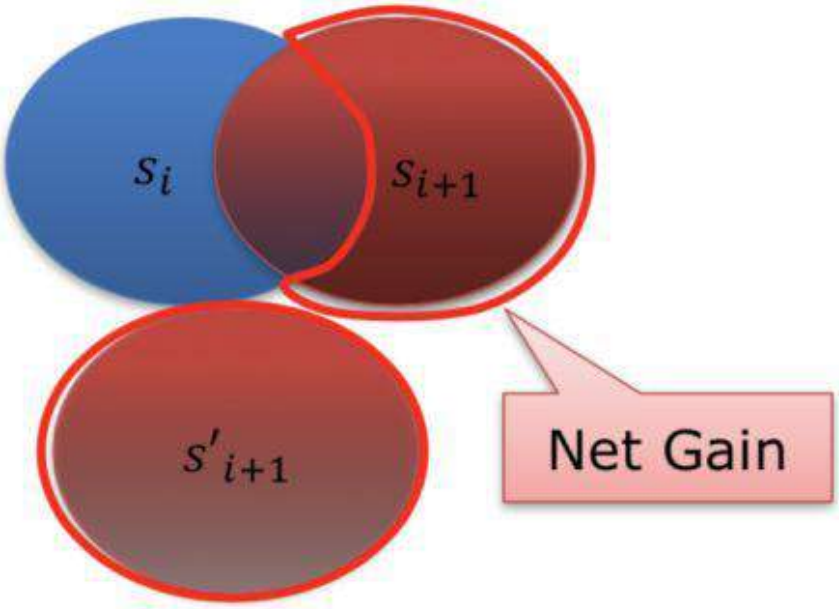
者百度等 App 一次会推荐很多新闻。另外一个层级是内容是不断出现的，例如刷新一次过后，又推荐一次内容。核心点是在于激发用户的兴趣点，例如昨天对某种类型新闻感兴趣，那今天可能就不那么感兴趣。

- K-Items Recommender System
  - Problem setting
    - Exposure of a list of K items: $s_1, s_2, \dots s_K$, Receive clicks: $r_1, \dots r_K$
  - **Item Independent Prediction**
    - Click of item is independent
    - $P(r_i|u, s_i) \propto f(u, s_i)$
  - Diversified Ranking
    - Submodular Ranking[1]
      - $P(r_i|u, s_i) \propto f(u, \Delta(s_i||s_1, \dots, s_{i-1}))$
      - $\Delta(s_i||s_1, \dots, s_{i-1})$: Net Gain
    - Deterministic Point Process[2]
      - Represent the item correlation with kernels



我们一般的推荐，可以认为叫做 k-items recommender system，问题定义为一次性给用户 k 个 item ，然后收集用户点击行为。以往经典做法是基于 item independent 的假设，来预估点击 ctr 之类的，我们把这个叫做 item-independent prediction. 不同用户的 item 点击是独立的，这种假设存在的问题是没有考虑多样性的情况。

一般解决多样性的方法有 1）submodular ranking（次模函数排序），submodular 实际上就对“边际效用递减”这个说法的形式化，通俗的说就是把所有商品看成一个集合，随着所拥有的物品数量的增加，那么获得剩下物品所得到的满足程度越来越小。在推荐系统中，在上文推荐的基础上进行重新估计，例如下面的图中第 i 次新闻推荐内容项为 s_i ，第 i+1 次推荐的内容需要考虑去掉已推荐过的相似内容，考虑用户感兴趣的 topic 空间 ，给用户推荐的应该是月牙形区域的内容 ，这便是 submodular 中的 net gain 的定义。



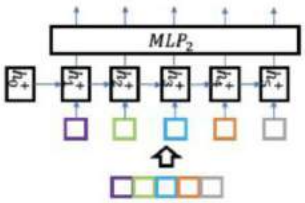
这种假设存在问题有：

1）模型任务多样性不是一个简单的统一的公式能总结的，实际上是跟用户和其 topic 是相关的。例如有的用户就是喜欢多看一样或类似的东西，另一些用户不喜欢。

2)多样性之外的 correlation 没有考虑 ,这些 correlation 可能包括正向的 correlation, 例如推荐两个有关联的 topic，比单独推荐一个 topic 更感兴趣。

3）还有一个问题是局部与全局最优的问题。

- **Preceding Items Aware Prediction**
  - Neural network models all possible correlations
  - $P(r_i|u, s_i) \propto f(u, s_i, S_i^- = \{s_1, \ldots, s_{i-1}\})$
  - Still feasible to be used for ranking items



首先考虑的建模方式是：

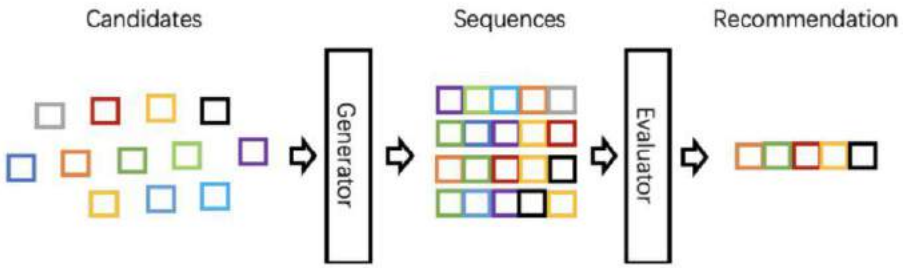
1）将已经排列的前文建模到模型中。这种建模方式也存在缺点，如实际中用户可能会被有些东西迷住，然后回过头看东西，这种情况是无法用前文的内容来描述。

2）采用上文和下文来同时建模。我们采用 rnn/gru/lstm 建模上文或者下文，上下文同时建模就会变成双向的 rnn/gru/lstm 模型。

- **Contextual Items Aware Prediction**
  - $P(r_i = 1|u, s_i) \propto f(u, d_i,\ S_i^- = \{s_1, \ldots, s_{i-1}\},\ S_i^+ = \{s_{i+1}, s_{i+2}, \ldots, s_K\})$
  - Following Items matters, too
  - Bi-Directional Correlation Modeling
  - Can not be used directly for ranking

但是双向模型是无法直接用来排序的。为此我们提出一种方法 Evaluator-generator framework，该系统特点是把高精度的评估模型作为一个衰减序列的模型。系统中有一系列的 generator 结构，generator 的目标是生成能够在 evaluator 上面拿到最高的分数的序列。

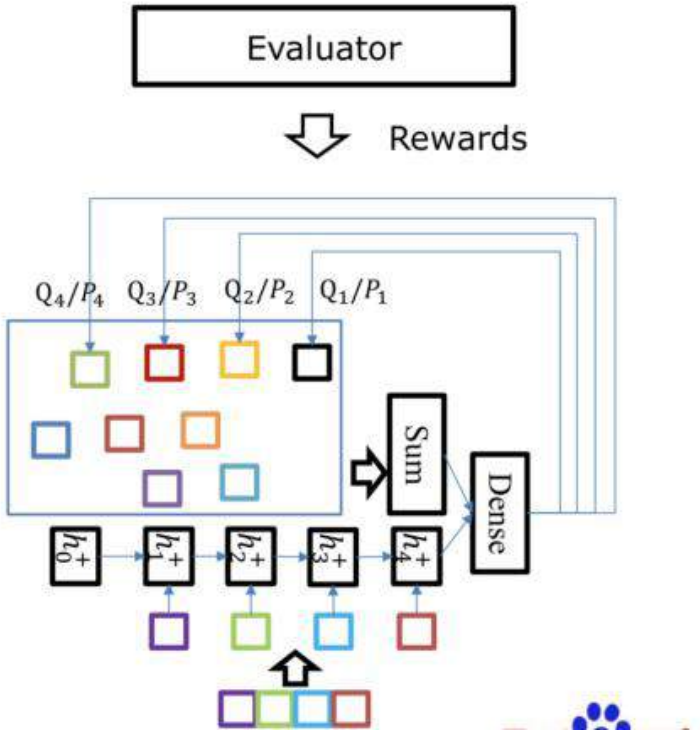
- Combining Bi-Directional Evaluation & Single Directional Generation



对于 Generator 来说，因为建模没有考虑到下文和全局（整个序列）的整体收益，所以最佳方式是考虑强化学习，为此我们提出了一种网络结构，revised pointer network(RPN)适合这种情况，RPN 的特点是：

1）是一种 set2seq 的建模方法；

2）能够将候选未排序的结合变成推荐序列。



以下是实验部分结果，已在百度的内容分发取得不错的结果，获得过百度最高奖。

- Offline
  - Sequence Optimization greatly increases the click & duration compared with DNN only
- Online
  - Ever since we launched Evaluator-Generator system
    - Average Clicks rise > 10%
    - Average Duration rise > 5%

Evaluator	Correlation (Click)	Correlation (Duration)
DNN	0.228	0.802
Bi-Directional GRNN	**0.968**	**0.947**

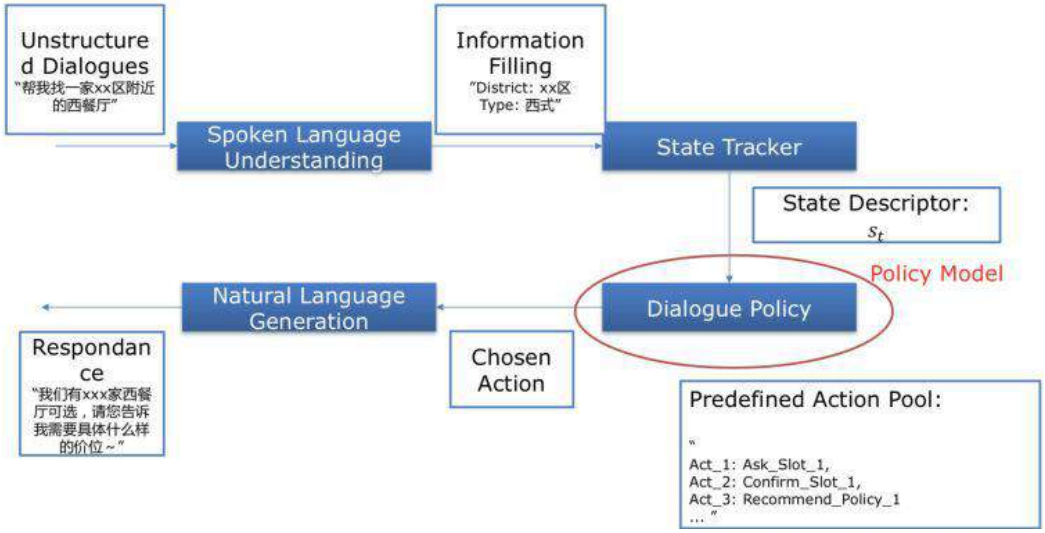
Generator	Evaluation Score
DNN Greedy	1.564
DNN + Sampling (20)	1.701
RPN + Q-Learning + Greedy	**1.941**

Evaluator	Pointwise AUC (Gain)	Correlations with the Total Clicks
DNN	+0.00%	0.468
GRNN	+0.82%	0.483
Bi-Directional GRNN	**+1.14%**	**0.492**

应用部分-对话系统

NLU 里面最重要的应用就是对话系统，小组从 2012 年就开始探索对话系统，做了不少原型系统。小组还参与部分度秘（duer os 前身）的工作，还有一个就是 open-domain 对话系统的工作。

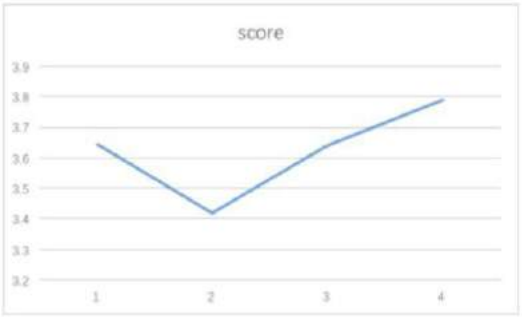
对话系统主要包括 NLU、state tracker、dialogue policy 和 NLG 四个部分。

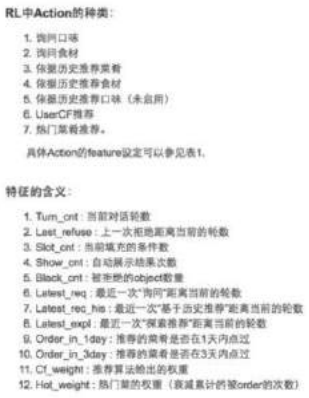


图中 state tracker 把状态流变成 state descriptor，dialogue policy 根据 state descriptor 来获取 action，最终生成结果。目前基于任务的对话系统大概还是这个结构。

最早强化学习在对话系统中的应用是 2012-2013 年做的点餐系统，这个项目拿到一些结论，但不是很确信。

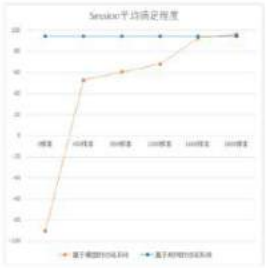
- Automatic Ordering System
  - Algorithms: Least Square Temporal Difference (LSTD), 12 feature, 7 actions
  - Feedbacks: Human Labeling, 1000 + sessions

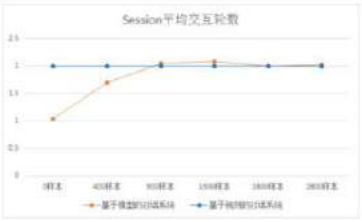


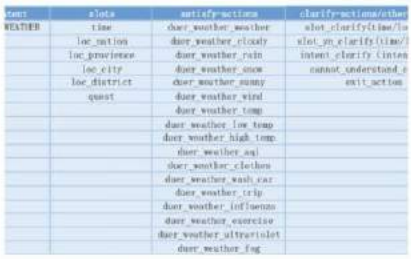


另外一个原型是天气预报系统，特征相对点餐系统多一些，这个原型能得出些重要结论，如发现有些指标能够超越人精心设计的规则策略，说明即通过学习还是有可能超过工程师精心设计的规则对话管理。

- Weather chat system
  - Algorithms: Least Square Temporal Difference (LSTD), 100+ features
  - Feedbacks: Human Labeling, 3000 sessions
  - Baseline: Elaborated Rules

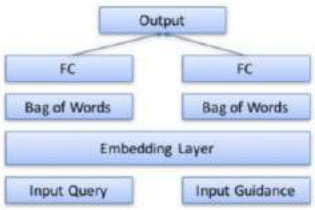


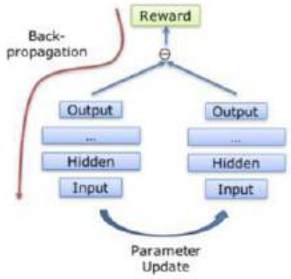




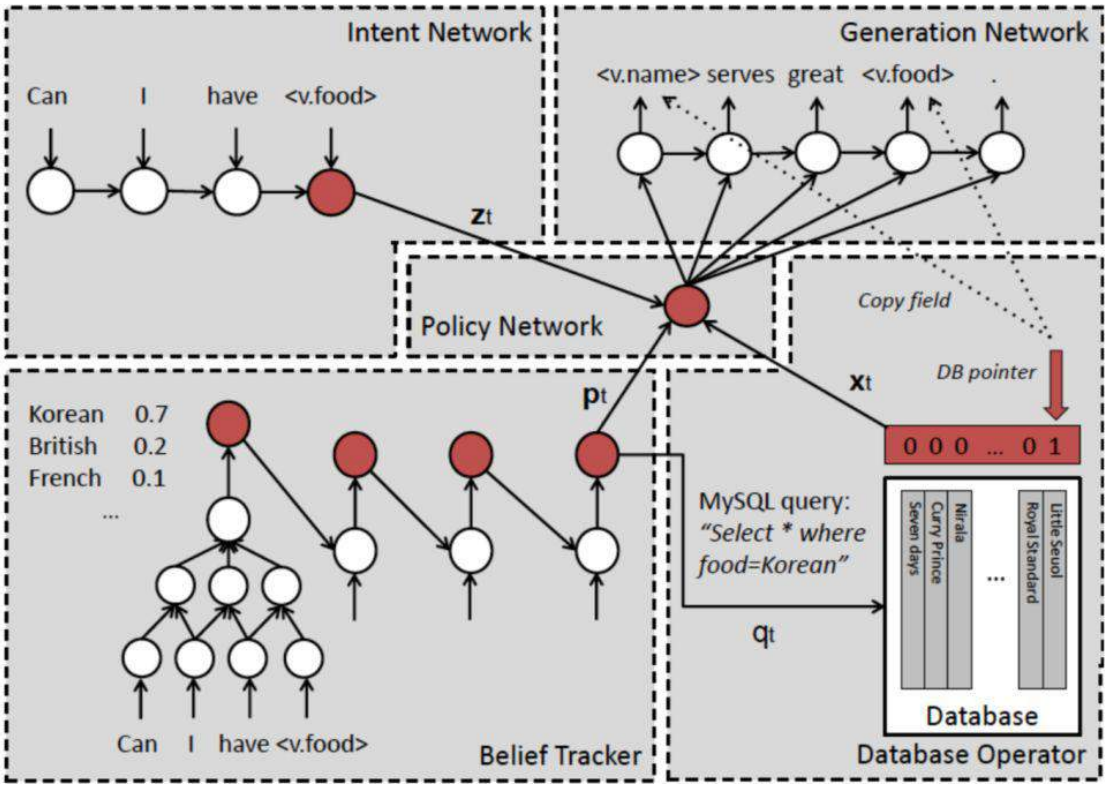
后来度秘里面也做了一个这样类似的功能，如对话引导，引导用户下一句说话，这过程可以通过强化学习来进行，核心部分是语义匹配 DQN。

- Conversation guidance (Dumi-Chatbot)
  - Algorithm: Semantic Matching DQN
  - Feedbacks: Off-Policy data, over 8 million
  - Evaluation vs Baseline (Semantic Matching DNN): GSB = 16:75:9



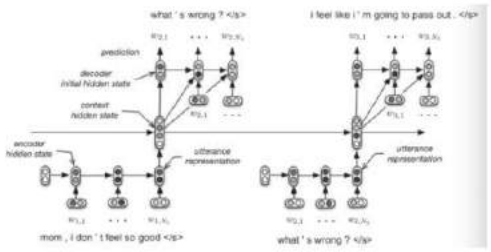


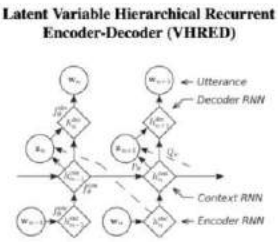
2016 年 Wen Tsung-Hsien 提出一种端到端的面向任务对话系统，使用神经网络把四个模块综合起来。所有模块中的状态都是可训练，这相对以前工作是进步一大步。



在 open-domain 方向最近的进展情况分为基于检索、自动生成两种方式。

- Retrieval Based
  - Semantic Matching (DSSM, ...)
- Generation Based
  - LSTM (Lowe, Ryan, et al 2015)
  - HRED (Serban, Iulian Vlad, et al. 2016)
  - VHRED (Serban, Iulian Vlad, et al. 2017)
  - ...
- Limitations
  - Generalized Response (Lol, that' s great, that' s good ....)
  - Lack of Consistency (I have a daughter, I am 12...)
  - Lack of Coherence (Where are you from? I like to play football...)
  - Lack of Goal





而对于开放域的对话系统，则尚没有标准的解决方案。一般认为，端到端神经网络是一个非常有潜力的方向。而端到端的对话系统存在的问题有：

1）生成式回答中没有内容，如对话会变成“哈哈” ，“呵呵”

2）缺乏逻辑性，如“我有一个女儿，我 12 岁”

3）答非所问，例如问：“你从哪里来”，回答“我喜欢踢足球”

4）最后一种是对话没有目标。

强化可以是解决以上问题的有效方法，但是也存在问题：

1）强化学习需要明确的学习目标。

2）强化学习需要用户的反馈，而这些反馈太昂贵，因为需要大量的数据，而且可能比有监督学习需要更多的数据集。

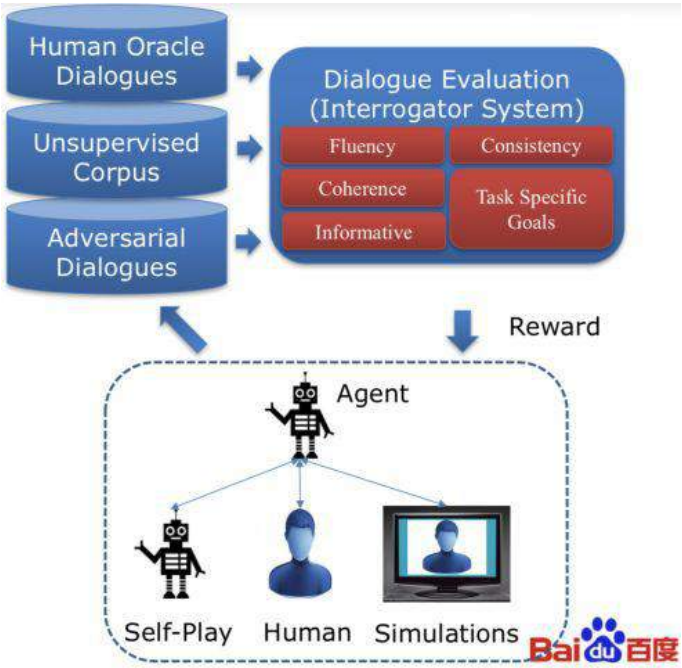
目前方法是尝试去解决 reward 如：

1）用户模拟器；

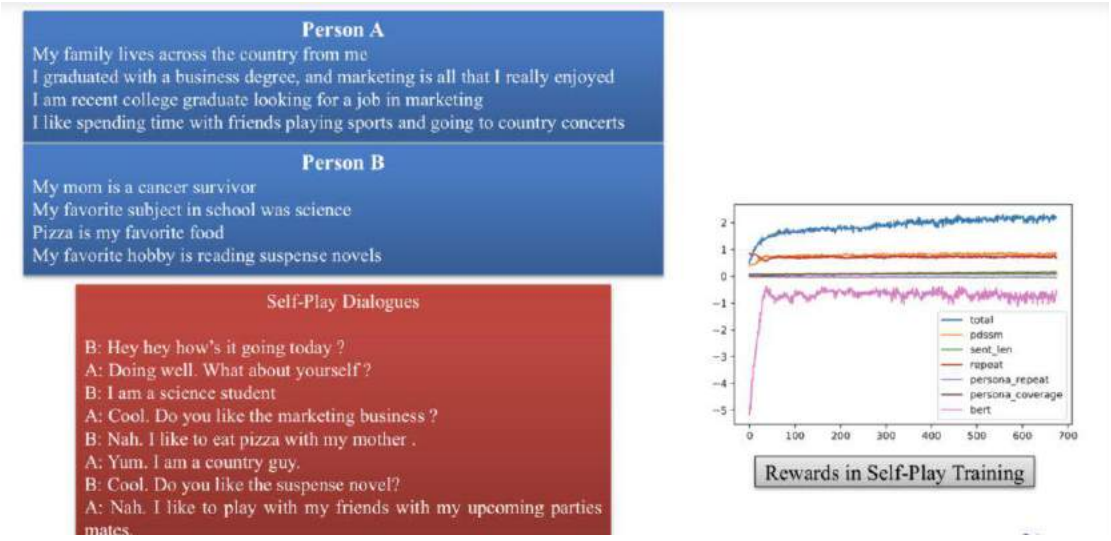
2）自我对话（self-play），两个机器人对话，然后评估对话质量；

3）adversarial self-play，即在评估的时候，加入对抗训练，把机器对话定义为假性对话，人与人之间对话是真性对话，评估时需要分辨出真假性对话，那就能给出正确的 reward。

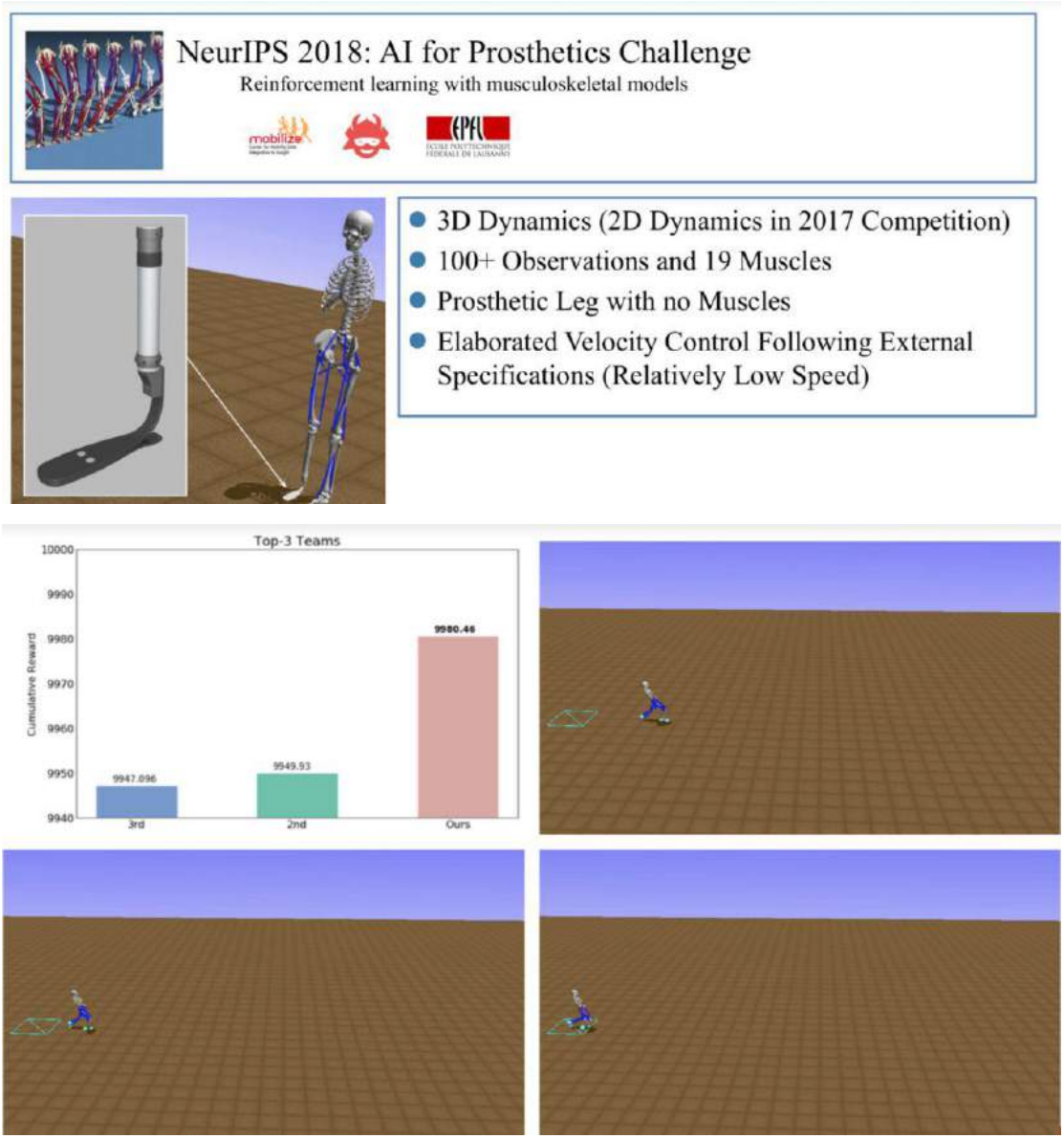
小组目前在做的是把以上的方法集成起来，让对话自动进行，具体如下。



基于以上考虑，我们做一些尝试性工作，图中是 FB 发布的数据集 persona，基于两个 persona（个性数据），产生两个人之间的对话. 这两个机器人相互聊，聊完之后用 evalution-system 来进行评估获得 reward，然后用来强化这两个机器人的对话。



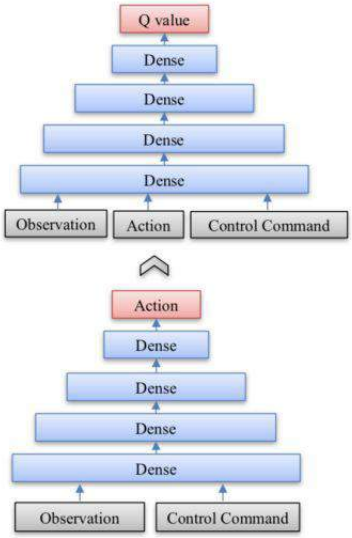
应用部分-移行控制



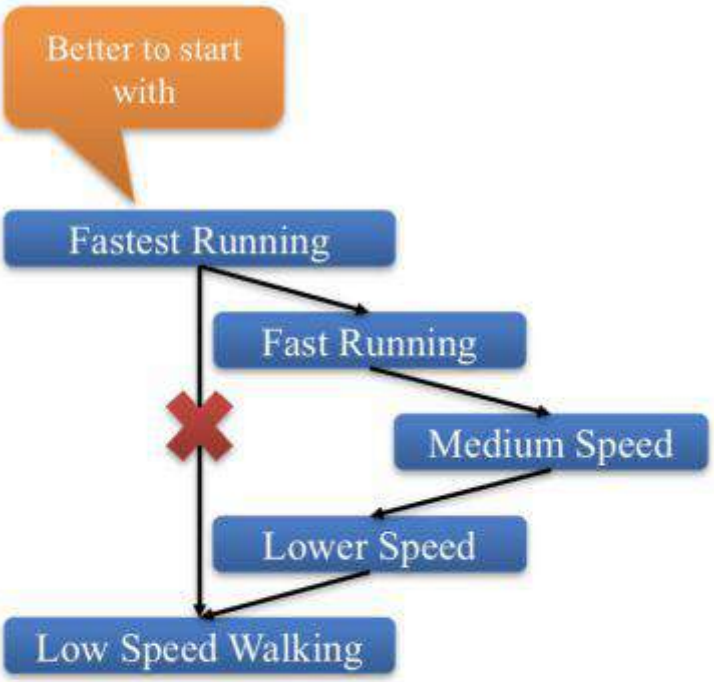
另外一个工作 ,是 NeurIPS 2018 中的一个比赛 ,叫 AI for Prosthetics. 在这个比赛中，需要控制一个人体骨骼肌肉模型的肌肉（19 维），需要学习的目标是按照外部不断变换的给定的速度行走。对没有按照速度行走或者跌落扣分处理。这个任务需要将强化学习和迁移学习相结合才能解决。

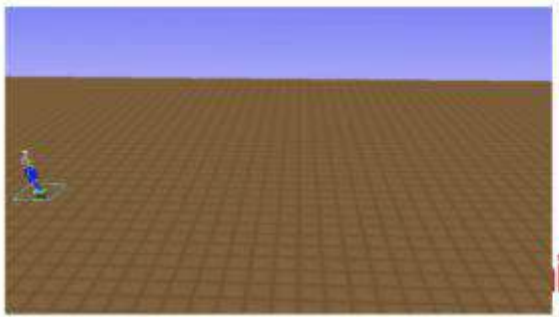
这次比赛中用到 4 个 trick：

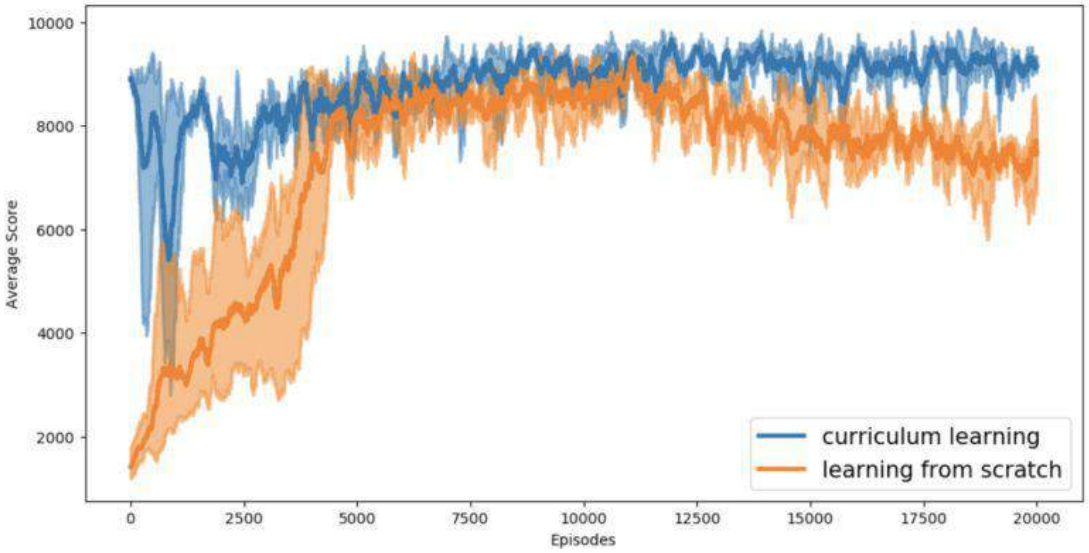
1）模型结构上：使用 deep deterministic policy gradient，该网络结构由 4 层 MLP 组成，以速度为学习目标，具体网络结构与强化学习结合情况如下。



2）Curriculum Learning ：课程学习并不是一开始就使用的。开始的学习指定速度低速行走的效果并不是很好，所以尝试换个思路改变问题的目标，让其跑的越快越好。这样训练我们发现能够得到非常高效的姿态。我们猜测低速行走是一个多解的问题。而在高速时，机器的姿态必须特别高效，其解相对更少。但比赛中的目标是更为低速的行走空间，因此需要进行迁移学习。我们将高速奔跑但是姿势较好的 policy 用于热启动，然后训练其逐步往低速的空间前一。通过这种方式我们得到了更好的奔跑姿势。在其中我们利用了课程学习，设计一系列从易到难的课程，让其逐渐地降低速度。通过这种方式，会得到一种特别稳定的姿态用于热启动。

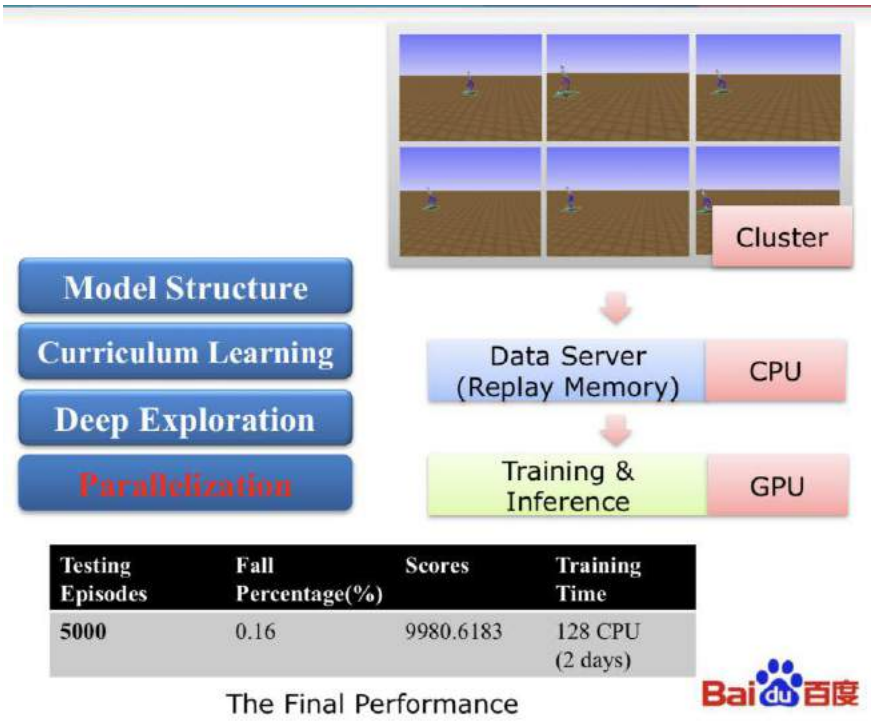

Better to start with
Fastest Running
Fast Running
Medium Speed
Lower Speed
Low Speed Walking




Average Score
Episodes
curriculum learning
learning from scratch

3）Deep Exploration 我们利用多个 header 的相似预测，得到其预测的波动，这个波动代表了当前策略和值的波动范围。这个能够帮助策略有效探索到探索不足的空间。

4）Parallelization：我们基于 PaddlePaddle 的高效的计算框架，能将性能提高几百上千倍。基于多个 CPU 的模拟器，以及一个 data server、一个高性能 GPU 作为单机多卡的训练。



Testing Episodes	Fall Percentage(%)	Scores	Training Time
5000	0.16	9980.6183	128 CPU (2 days)

The Final Performance

另外一个项目跟自动驾驶方面相关，我们利用原始的传感器信号直接决策控制信号（端到端控制）。我们在一个飞行器上进行这个实验。而训练控制模型是典型的延迟反馈问题，强化学习非常适合解决这类问题。

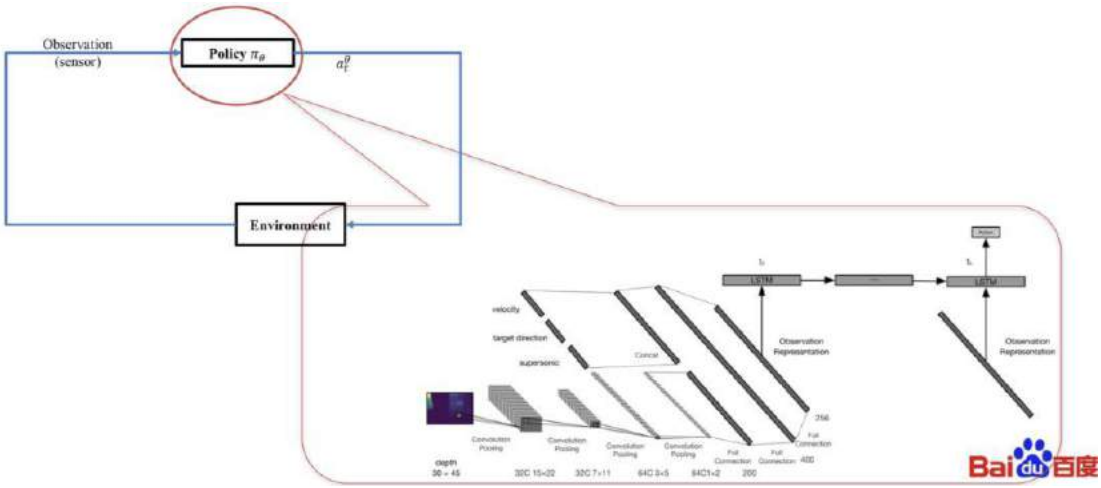
- Deploying RL to high risk real world control system faces challenges in terms of safety and cost
- We use human intervention to ensure safety during the **training process**
- We propose the IARL framework for policy optimization based on the **intervened trajectory**

然而强化学习用在 autonomous driving 中存在的问题有：

1）需要大量的数据；

2）强化学习需要一些冒进的动作，并且需要经历一些风险和失败（比如碰撞）。然而在真实的世界这个成本过大，我们不可能让无人车真的撞毁无数次来学会如何躲避障碍。

我们设想的解决方案是利用安全员，当发现紧急情况时，安全员会做一些紧急动作。假设我们 policy 如下图所示 ,是一个比较复杂的端到端网络结构 ,融合 CNN 与 LSTM(多帧信息)。但是我们不能把无人车或者飞行器真的完全交给这个模型，而是通过引入安全员来保障安全。

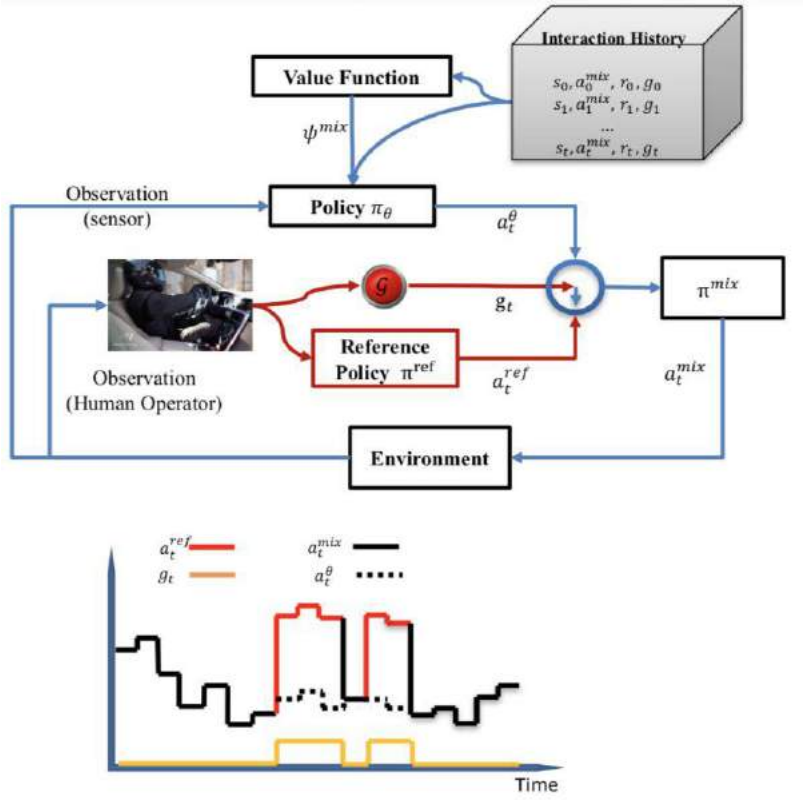


当我们系统做出错误判断或者出现高危动作是，安全员会及时介入。在以往的无人车中，安全员都只作为安全保障或者测试用。然而，安全员的每次干预，说明机器的工作不对。这其实是一个非常珍贵的反馈信息，应该被机器反过来学习。这也是设计 IARL 的初衷。

我们设计的 IARL 算法包括两个方面：

1）惩罚任何干预；

2）学习安全员干预时所采用的操作。



IARL 在 policy 和 loss function 中的体现如下：

1）我们需要修改 behvior policy,因为现在的 policy 不再是机器完全自主，而是变成了机器和干预的混合策略。

2）我们在 reward 上，对干预的过程做出惩罚。

3）我们对于 policy 的目标函数进行修改，增加一项用于学习干预过程的安全员操作。

- Behavior Policy Formulation

$$\pi_\theta^{mix}(a|s_t) = (1 - \mathcal{G}(s_t))\pi_\theta(a|s_t) + \mathcal{G}(s_t)\pi^{ref}(a|s_t)$$

- Loss Function

$$L^{IARL} \propto -E_{\pi_\theta^{mix}}[\sum_{g_t=0} \psi_t^{mix} \log \pi_\theta(a_t^{mix}|s_t) + \alpha \sum_{g_t=1} \log \pi_\theta(a_t^{mix}|s_t)]$$

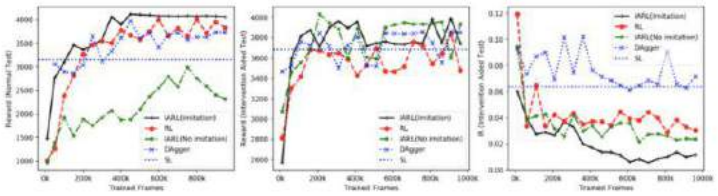
Intervention Avoidance　　Reference Imitating

实验结果如下：

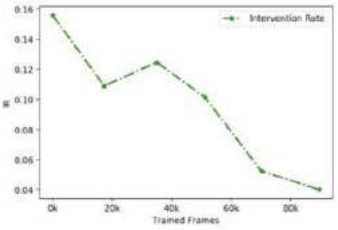
- IARL learns to avoid collision **without** any collision during the training process
- IARL outperforms the baselines in collision rate, rewards, and Intervention Rate.
- Practical for various high risk real world tasks.

Table 1: Average collision in *Normal Test* in different groups

Method	SL	DAgger	RL	IARL(No Imitation)	IARL(Imitation)
Average Collision	40%	24%	22.5%	42%	0%



Simulations



Reality

工具-PARL

PARL 是 PaddlePaddle 上的一个强化学习平台，特点是：

1）灵活性高，支持不同框架的强化学习；

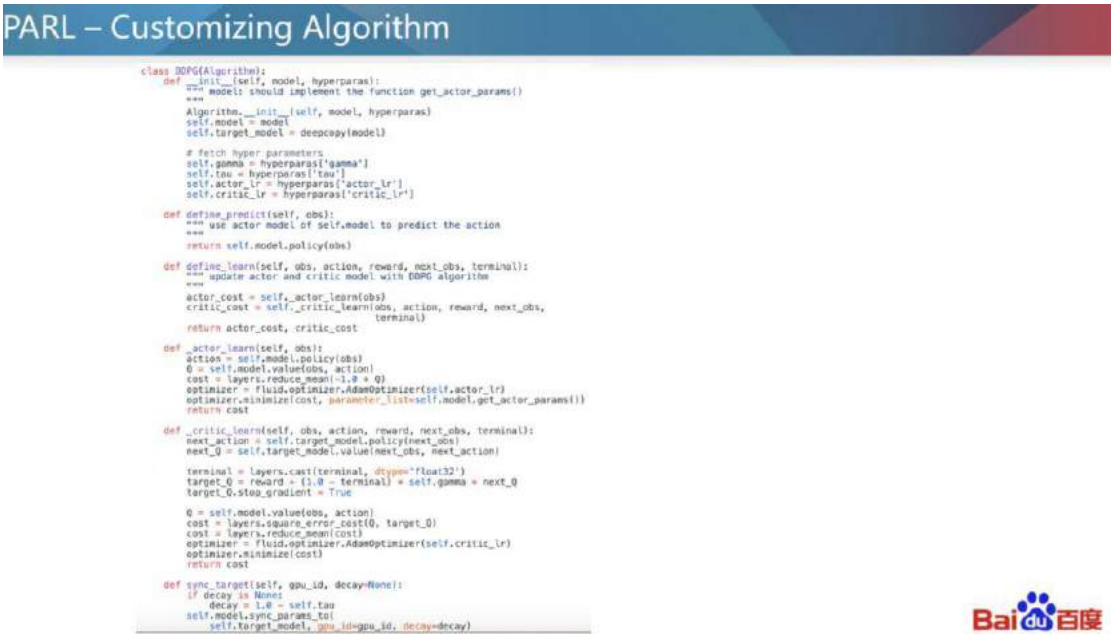
2)为数不多的开源其大规模分布式能力的库。而如谷歌的 tensorflow 里面有很多限制，并没有完全开放出来，尤其是在工业级的应用层面；

3）同时支持工业界应用和科研任务。

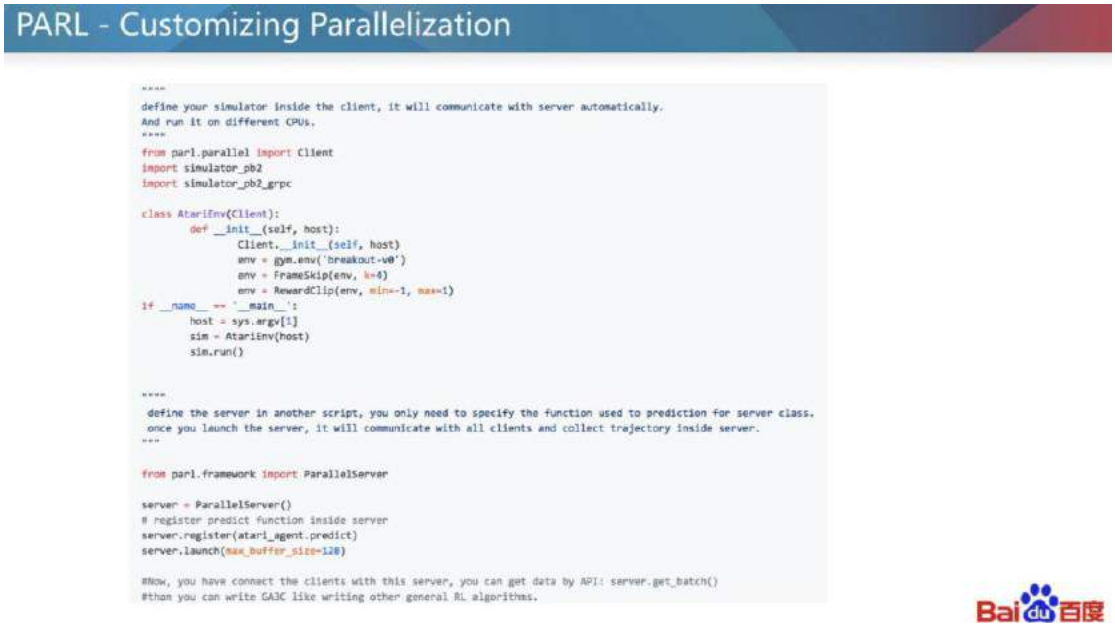
这套强化学习框架基于 3 个基础类，分别是 agent、algorithm、model。 通过不同层级的定制，PARL 能够实现方便，灵活，可服用，定制方便的强化学习算法。 同时，对于大规模并行能力有很好的支持。用户可以很方便写出自己独特的定制算法，而不必去管具体通信接口等逻辑。 以 DQN 为例，我们提供了 algorithm 的现成算法，用户只需要定制自己的模型。

```
import parl
from parl.algorithms import DQN, DDQN

class AtariModel(parl.Model):
        """AtariModel
        This class defines the forward part for an algorithm,
        its input is state observed on environment.
        """
        def __init__(self, img_shape, action_dim):
                # define your layers
                self.cnn1 = layers.conv_2d(num_filters=32, filter_size=5,
                                                stride=[1, 1], padding=[2, 2], act='relu')
                ...
                self.fc1 = layers.fc(action_dim)
        def value(self, img):
                # define how to estimate the Q value based on the image of atari games.
                img = img / 255.0
                l = self.cnn1(img)
                ...
                Q = self.fc1(l)
                return Q
"""
three steps to build an agent
   1.  define a forward model which is critic_model is this example
   2.  a. to build a DQN algorithm, just pass the critic_model to `DQN`
       b. to build a DDQN algorithm, just replace DQN in following line with DDQN
   3.  define the I/O part in AtariAgent so that it could update the algorithm based on the interactive data
"""

model = AtariModel(img_shape=(32, 32), action_dim=4)
algorithm = DQN(model)
agent = AtariAgent(aglrotihm)
```

而如果用户需要定制全新的 RL 算法 ,而且库里面本来没有 ,也可以童工定制 Algorithm 简单地实现。



最后，我们也展示一个例子，用户能够通过定义一些通信逻辑，实现一些并行化的复杂操作。



强化学习的存在问题及应对方法

最后讲讲强化学习当前存在的一些问题和最有潜力的方向。

1）在很多应用中，往往是目标不明确，例如对话最终目的是什么，像人？像人这个目标又该如何定义？解决方法是 reward modeling，最近谷歌也在做这个方向。例如百

度在推荐排序的时候，使用 evalution-generator 的框架，即首先对 reward 的建模而不是一上来就学习。

2）强化学习需要海量的样本，甚至比有监督学习还需要更多的样本。解决方法有使用 world model 或 planning。

3）奖励函数过于稀疏，极其难以探索到优质解。解决方法有分层训练、课程学习和引入辅助任务。

4）泛化能力非常差，基本可以说很多是处在过拟合的区域。可以使用元学习，迁移学习，以及近期大火的攻防理论来改善。

5）实验难以复现：很多实验，甚至随机种子不一样都会得到完全不一致的效果。我们提倡开源，也是要解决这个问题的第一步。

参考文献：

[1] Schulman, John, et al. "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438 (2015).
[2] Yue, Yisong, and Carlos Guestrin. "Linear submodular bandits and their application to diversified retrieval." Advances in Neural Information Processing Systems. 2011.
[3] Wilhelm, Mark, et al. "Practical Diversified Recommendations on YouTube with Determinantal Point Processes." Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 2018.
[4] From item-wise recommendation to combinatorial recommendation: Sequence Optimization Approach

[5] Wen, Tsung-Hsien, et al. "A network-based end-to-end trainable task-oriented dialogue system." arXiv preprint arXiv:1604.04562 (2016).
[6] Zhang, Saizheng, et al. "Personalizing Dialogue Agents: I have a dog, do you have pets too?." arXiv preprint arXiv:1801.07243 (2018). “ Self-Evolving Dialogue System with Adversarial Safe Automatic Evaluation”
[7] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971(2015).
[8] Bengio, Yoshua, et al. "Curriculum learning." Proceedings of the 26th annual international conference on machine learning. ACM, 2009.
[9] Ian Osband, Charles Blundell, Alexander Pritzel, Benjamin Van Roy, Deep Exploration via Bootstrapped DQN, In NIPS 2016.
[10] Wang, F., Zhou, B., Chen, K., Fan, T., Zhang, X., Li, J., ... & Pan, J. (2018, October). Intervention Aided Reinforcement Learning for Safe and Practical Policy Optimization in Navigation. In Conference on Robot Learning.

强化学习在滴滴车服用户运营中的实践

嘉宾:刘凯　编辑：田小娣

导读：本次分享的主题为强化学习在滴滴车服用户运营中的实践。在互联网场景下，面向用户的营销是每个业务所对的共同问题，而如何帮助滴滴车服运营提升营销的 ROI 是我们算法同学的工作重点之一。在本文，我们将向大家介绍如何将车服的营销问题建模为一个强化学习过程，并且如何精细化地对每个人做出个性化的营销动作，在实际业务中又取得了怎样的效果。本次分享主要包括：

- 滴滴车服用户运营算法体系
- 智能营销建模方法
- 效果展示

——滴滴车服用户运营算法体系——

1. 滴滴用户运营算法房子图

为了支持用户增长的需求，滴滴车服构建了一个平台，一个系统以及四个算法策略模块：即流量分发平台、标签系统、画像模块、目标人群定向模块、营销策略模块、触达优化模块。

流量分发平台是对用户进行场景化广告的弹出，和用户进行一些交互，包括发 push 消息、播报及发短信消息等各种渠道。

标签系统一方面在线上服务中判别用户的身份属性、状态以及身份，另一方面是帮助各业务的运营人员去细粒度的圈选目标人群。

针对用户运营的四个阶段，也有相应的算法策略模块：

第一个模块是画像，包括用户画像，商家画像以及车辆画像，目标是把生态内的各个实体细粒度的刻画出来。

第二个模块是目标人群的定向，用户运营首先根据任务和用户生命周期去找到合适的用户群体，比如对用户长期价值的定向、对短期的转化率的预估、用户流失预警、体系外用户的精准定向等。

第三个模块是营销策略，分两部分，一个是滴滴体系内线上的一些算法策略模块，如强化学习、组合策略、个性化推荐，滴滴体系内有大量的拉单司机、专车司机、快车司机、顺风车司机以及出租车司机，他们和滴滴平台有非常紧密的联系，每天都在平台内拉单，用户行为非常稠密，为智能营销提供了丰富的源数据。另一部分是滴滴体系外的用户增长，如社交营销、DSP 外投广告。

第四个模块是触达优化，算法根据用户的状态和行为发放给用户一张一定面额的优惠券或一个活动提醒，为了增强用户的转化意愿，在触达优化阶段对用户进行个性化的消息提醒。因为平台业务众多，各业务下的运营活动众多，为了避免无节制地把所有的活动消息在很短的时间内推送给同一个人，需要做一些取舍，这就是全局流量优化解决的问题：在尽量减少打扰用户的限制下，提升平台的收益。

2. 人工运营的痛点和解决方案

人工运营的痛点	解决方案 =强化学习+个性化消息提醒
■粗放式运营：很难对用户进行精细分层	■精细化运营：细粒度地对用户进行精细分层
■经验驱动：未充分利用用户的在平台内线上线下各种细粒度的行为数据	■数据驱动：可充分利用用户的在平台内线上线下各种细粒度的行为数据
■任务划分：不同生命周期阶段分任务运营，忽略用户状态的变化	■过程运营：平台与用户的交互建模为一个过程，不同生命周期统一成一个大任务
■触达方式：非个性化，不能满足用户需求的痛点，易疲劳	■触达方式：可匹配用户的个性化兴趣，满足用户潜在需求渴望，提高打开率和转化率
■效果天花板：抓手较少，天花板较低	■效果天花板：抓手较多，可不断抬高

人工运营的痛点和解决方案

人工运营和监督学习方法可以圈选目标人群进行运营活动，但是比较粗放，依赖运营人员的经验，将用户生命周期强制分成拉新、提频、沉默召回等阶段，由不同运营人员负责，但是这样将连续过程割裂开不利于运营效果的提升；同时没有利用平台用户线上线下丰富的行为特征，很难对用户进行精细

化的分析和运营；消息提醒千篇一律，用户会感到疲劳从而影响转化率。针对这些痛点，滴滴车服采用强化学习方法对用户和平台之间的交互过程进行建模，并且在消息提醒阶段，使用 graph embedding 方法根据用户的兴趣进行个性化的消息提醒，真正的做到精准触达。

——智能营销建模方法——

1. 强化学习

强化学习是一种与监督学习不一样的学习范式，通过一个智能体与环境进行交互从而达到学习目标。其最重要的概念包括状态 State、动作 Action、回报 Reward，智能体接收到环境的状态后，对该状态做出一个动作，环境根据该动作做出一个回报，然后进行一轮一轮的过程学习。强化学习的典型应用有游戏、个性化推荐、效果广告和网约车调度，如 AlphaGo Zero 在围棋领域战胜世界顶级选手、多款游戏中的 OpenAI 基本战胜人类，滴滴的网约车调度也采用强化学习去预测司机和乘客在时空中的匹配价值，进而在提升平台 GMV 的同时也提升了乘客和用户的产品体验。

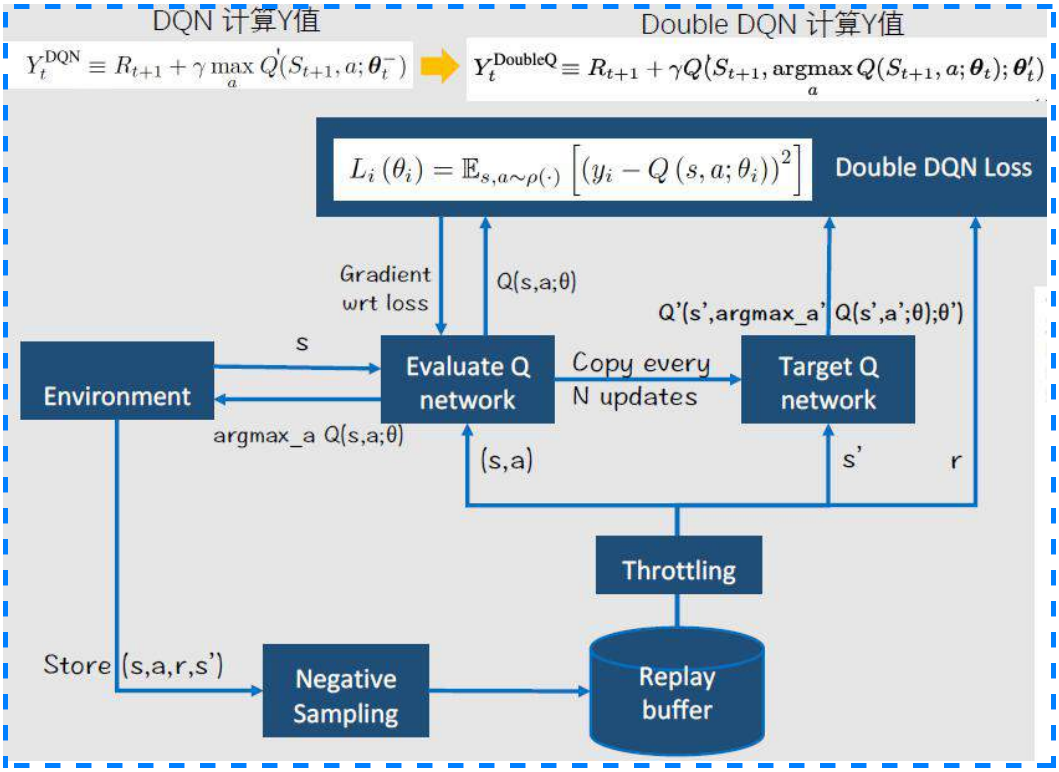
强化学习分类	■ Model based VS model free ■ Model free: policy opt, q-learning, policy opt与q-learning融合 ■ policy opt: A3C, PPO ■ q-learning: Deep Q-Learning ■ policy opt与q-learning融合: DDPG, SAC

强化学习分类

大的方面强化学习可分为 Model based 和 Model free，两者的区别是 Model based 可以完整的了解并学习 Agent 所处的环境，Model free 却不能。Model free 分为三方面：策略梯度优化、Q-learning 及两者的结合，策略梯度优化适用的场景是动作连续的或者动作空间非常大，比较适合推荐或广告领域，典型的方法有 A3C 和 PPO；Q-learning 适合的场景是动作空间有限的几个或几十个，典型方法有 deep Q-learning；两者结合的典型方法有 DDPG 和 SAC。针对车服用户运营的问题，首先两个实体中 Agent 是车服平台，Environment 是用户，或者其他上下文。在每一轮迭代中 Agent 会向环境发送一张一定面额一定周期的优惠券，或者一定周期的消息或者空动作，Environment 收到动作后经过一定周期会做出一个正向或负向反馈，该反馈被量化后发送给 Agent，环境状态的变化 State 也会返回给 Agent。

强化学习图解 + Action 周期

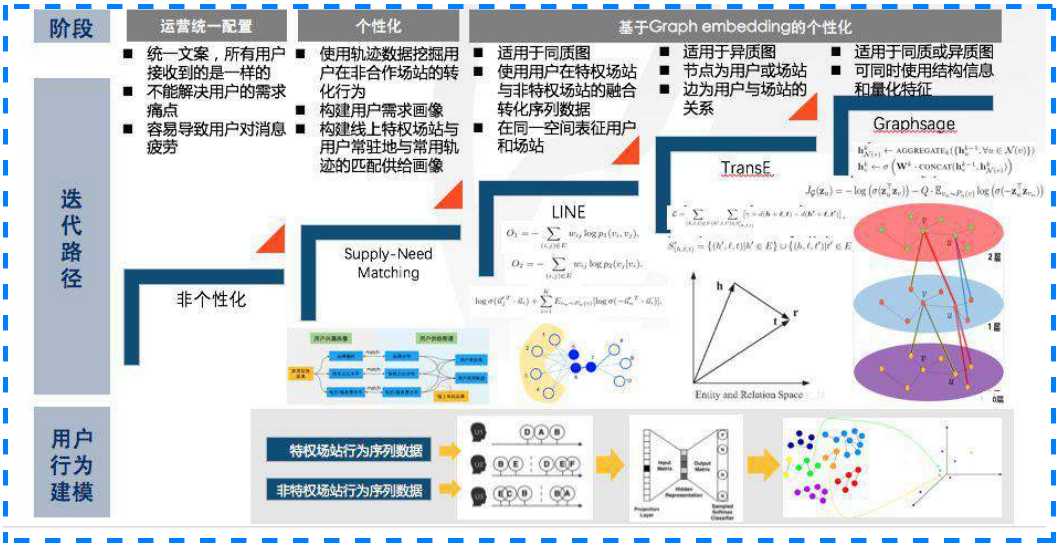
在这个场景里，强化学习的第一个要素 Action 包括不同面额优惠券的推送、不同周期的消息推送、空动作；第二个要素 State 是观察和抽取出来的一些特征的表征，包括用户线下行为、用户线上行为、静态行为、模型学习预估分；第三个要素 Reward 是用户对 Action 的反馈，如空动作加油、消息推送加油、用优惠券加油；用户查看优惠券、查看消息；无加油无查看。除此之外，针对场景需要定义了两类 Action 周期：动作周期和沉默周期，传统强化学习当 Agent 发出动作 Environment 会马上给出一个响应，但在 O2O 场景下用户需要一定周期（动作周期）对动作做出响应，如加油，并且在下一时刻用户不可能再去加油，会进入沉默周期。



Double DQN 算法流程

因为经典的 DQN 会带来 Q 值预估的 overstimatation，进而引起接下来训练的震荡，所以具体的算法采用 Double deep Q-learning network，算法分为训练部分和预测部分 :训练部分的核心是损失值 loss 的不同 ,double DQN 中左边网络训练的 Q 值会周期性的复制给右侧的 target Q-network，二者共同去计算得到最终的损失值 loss；预测部分是 Environment 会把当前的状态输出给 Double DQN，通过计算将 Q 值最大的 Action 返回给环境，如发一张合适的优惠券、一个消息推送或者空动作。此外针对正负比例不均衡的情况，算法采用了负采样的方法。

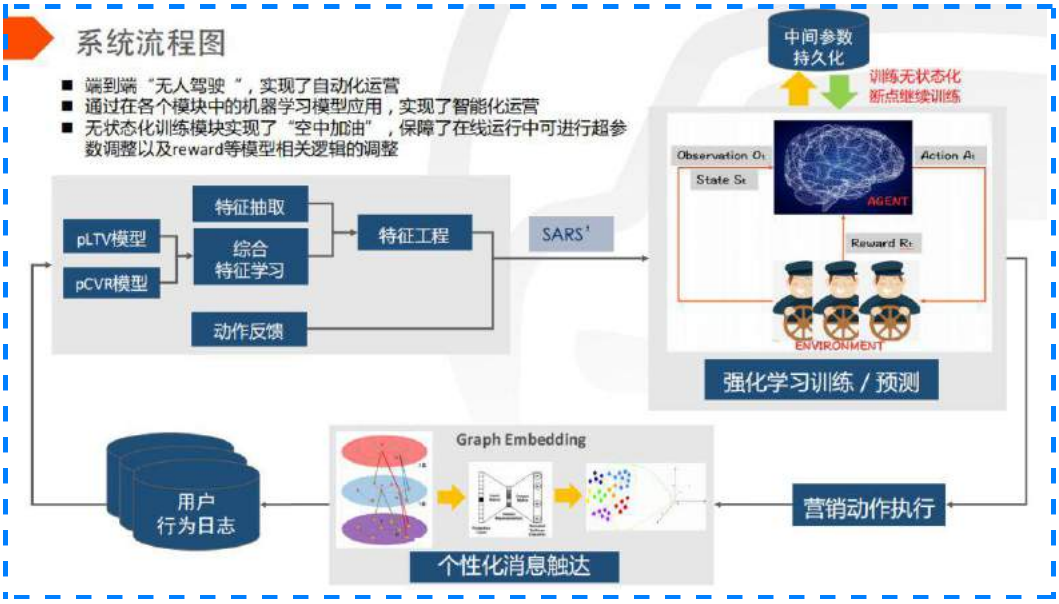
2. Graph embedding 个性化消息触达



消息提醒部分可分为三个阶段：首先是运营统一配置，在这个阶段所有人收到的消息是一样的，用户容易产生疲劳；其次是初步的个性化，同时构建用

户的需求画像和平台的供给画像，然后将二者匹配起来，这样可以达到一个相对较好的结果；最后是为了达到更好的个性化结果，基于用户的行为序列构建相应的图，采用 graph embedding 学习方法实现个性化消息的推送，具体使用了三种方法：LINE、TransE、Graphsage。LINE 适用同质图进行学习，在同一空间表征用户和场站，使用用户在特权场站与非特权场站的融合转化序列数据；TransE 适用于异质图，节点的种类包含用户或场站，边为用户与场站的关系；Graphsage 适用于同质或异质图，可同时使用结构信息和量化特征。

3. 强化学习和 graph embedding 相结合



将强化学习和 graph embedding 两者结合，目前在端到端 "无人驾驶" 的大流量上已经实现了自动化运营，通过在各个模块中机器学习模型的应用，

实现了智能化运营。首先通过特征提取模块的学习得到强化学习所需要的 State、Action、Reword 三要素，然后放到强化学习算法中去学习和训练，产出营销动作进而执行，执行的时候通过 graph embedding 个性化的消息推送，给每个用户匹配合适的服务，进而提高用户的消息打开率和转化率，最后通过用户行为收集进入下一轮的迭代，而且在强化学习过程中，实现了中间参数持久化的无状态化训练。

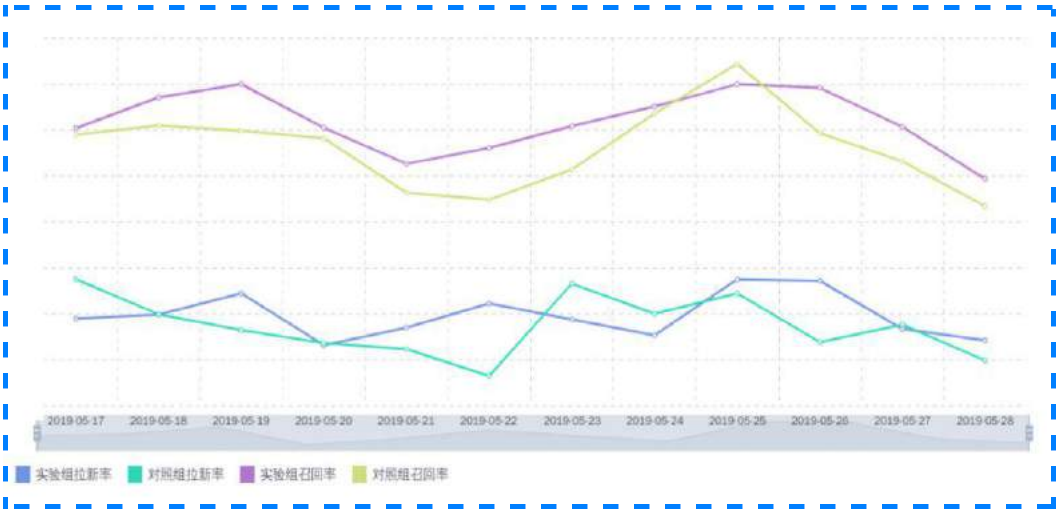
——效果展示——

1. 强化学习算法效果

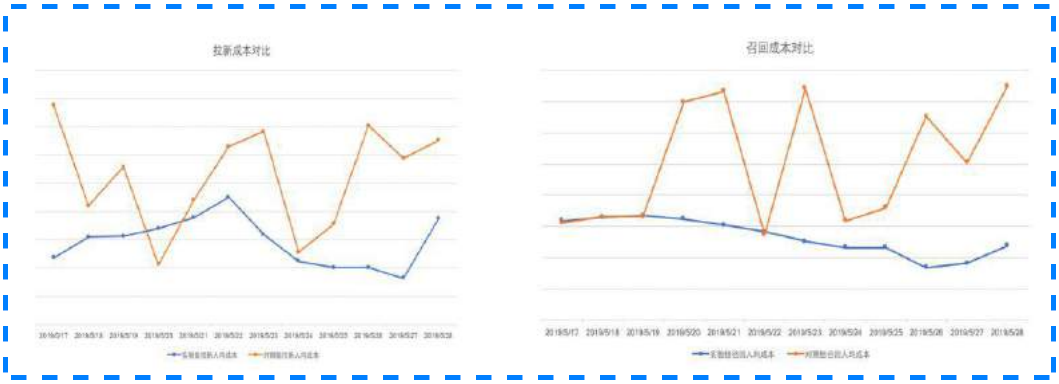
强化学习部分，从 ROI 的趋势图可以看到强化学习实验组 ROI 是稳定的，基本上是稳定的高于对照组的。

ROI 趋势图

目前强化学习算法已经全流量覆盖加油业务用户，包括有券提醒和无券推送，强化学习桶比人工运营桶在拉新率和召回率上都有约 8%的提升，同时成本大约降低了一半，真正实现了一个比较高的 ROI，从而做到了更加精细化的运营。



强化学习拉新召回率对比



拉新成本对比和召回成本对比

2. Graph embedding 个性化消息提醒效果

关于个性化消息提醒方面的优化，首先用 LINE 方法和人工供需匹配做对比，在消息打开率和转化率上分别提升了 7%和 10%；然后将 TransE 方法和 LINE 方法做对比，在打开率和转化率方面分别提升了 4%和 6%，累计起来的话 TransE 方法比人工供需匹配在打开率上提升 11%，转化率上提升 16%。总之，目前基于 graph embedding 的个性化消息提醒方法在加油业务全流量上，每天对大量用户进行个性化的消息提醒，用户体验也明显提升。

在用户增长上，车服算法团队初步建立起来一套智能营销的体系，通过数据驱动的方式精细化地赋能了运营，提升了用户增长的效果和效率。

本次分享就到这里，谢谢大家！

PS：滴滴车服智能营销算法团队也期待算法牛人加入，一起玩算法拿结果！

邮箱：kailiusci@163.com

——Q&A ——

Q：智能运营的业务背景是？

A：智能运营的含义是数据化运营，通过用户在线上线下的各种行为所反映出来的偏好，通过相应的算法策略，能够精确地进行个性化的营销，提高用户

的转化率，降低转化成本，从而实现 ROI 可控的规模获客，智能的对用户全生命周期进行精细运营。

Q：要招什么样的人？

A：智能营销所解决的本质问题，还是如何实现人服务以及营销策略三者的精确匹配，所以从这个意义上来讲，跟推荐广告实际上是可以触类旁通，基于用户行为进行偏好建模，并匹配个性化的营销策略。所以需要的基础还是机器学习，以及对数据业务场景的理解能力。

Q：智能营销的其他解决方案及效果？

A：我们只是列举了两个模块中的相应方法，那么在其他场景中，我们也采用了相应的新方法，并取得了正向的效果。比如说如何对明显延迟反馈的场景进行转化率预估的建模，还有如何在相关任务中使用多任务学习，既可以提高模型训练的效率，也可以通过相互引入噪声，提高模型训练的泛化能力，以进一步提高对于人群分层的效果等等。

Q：具体的模型实现？

A：关于具体的实现，现在很容易找到具体方法的开源实现。比如说强化学习的 DQN 算法，可以找到很多。关键的有两点，第一个是对于场景的理解，能够对这个业务场景的问题抽象成一个算法问题；第二点，能够深入的理解

算法的细节以及优缺点，这样就可以，将场景和一些算法能够去匹配起来，用合适的算法来解决形式化出来的算法问题，这样就可以去适配业务场景，拿到正向的效果。

小米移动搜索中的 AI 技术

嘉宾：宫士敏　编辑：王成林

一、背景

以深度学习为代表的 AI 技术近年来在搜索推荐场景中得到广泛的应用，对用户 Query 和内容侧的理解不断深入，用户体验得到显著提升。

搜索是一种典型的 AI 应用，关键技术包括 Query 分析 、内容理解、文本相关性计算以及排序等。从统计机器学习到深度学习，算法结合大数据将搜索引擎精度不断的提升。本次分享以部分技术为例探讨小米移动搜索中应用的 AI 技术。

移动搜索

搜索提示

小米的移动搜索主要有两个场景，MIUI 的主搜索入口，以及浏览器中搜索提示。小米的移动搜索针对移动端搜索的特点，致力于为用户提供精准便捷的体验。实现了应用、内容、服务的直达。例如搜索视频时，直接够唤起爱奇艺/腾讯视频等 app 观看特定选集。

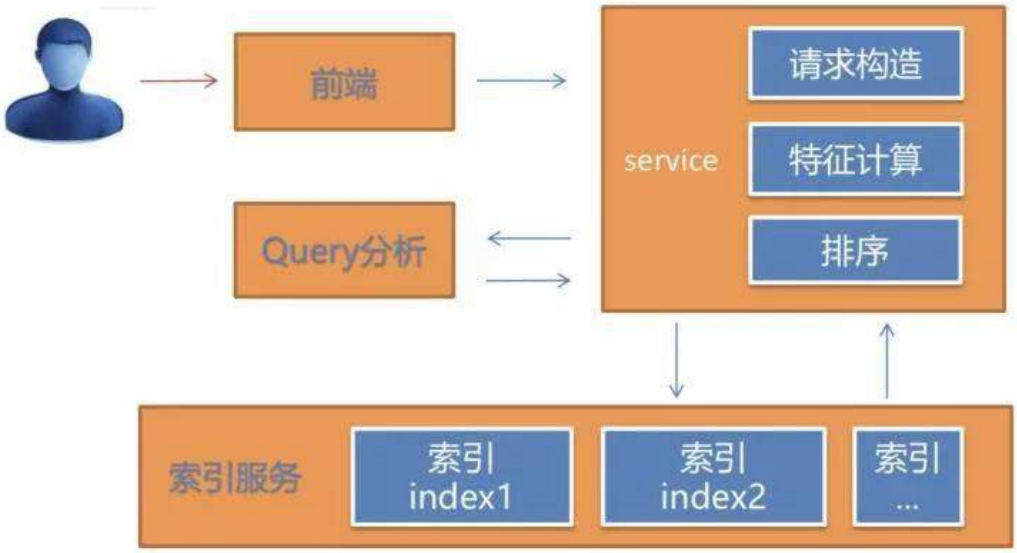
覆盖小米体系和外部合作方内容，以下是部分合作垂类内容提供方：

垂类	内容提供方
应用	小米应用商店
视频	小米视频 爱奇艺 腾讯视频 优酷视频
新闻	一点资讯 网易新闻
百科	搜狗百科
问答	百度知道 搜狗问问 新浪爱问
音乐	小米音乐
小说	多看阅读 百度小说
...	

今天分享的技术内容分几个部分：

- 文本相关性
- 意图识别和词权重
- 点击模型排序

先介绍下搜索系统的典型结构：



最基本的搜索架构大体上也都有这样几个部分，百度、搜狗、神马、360等各大搜索厂商采用的基本架构都是类似的，包括：

- 前端模块
- Query 分析模块，负责计算意图/词权重/同义词和 Query 改写等。
- 取得 Query 分析结果后，发送请求到各索引中进行结果召回。包括关键词召回、向量召回等。通常还会在这个阶段完成低级别特征的计算和粗排。
- 将上述结果进行高阶特征生成和重排序(精排)，排序后对结果再做一些处理逻辑后返回给前端。

搜索与推荐系统的处理过程有不少类似的地方。

二、文本相关性计算

问答和新闻 Query 场景下，文本相关性计算的精度对搜索体验的影响显著。

2.1 问题的定义：以查询 Q 和文档 D 的标题为例，计算相关性分数。

2.2 挑战：缺词，多词，逆序，临近，同义词...

中文的文本相关性计算是一个比较复杂的问题。

例如：

– Q:邮箱密码的格式
– D:邮箱密码格式是什么样的

– Q:辽宁抚顺发生地震
– D:辽宁抚顺发生3.1级地震

举一个简单的例子，第一个问答中的两种表述存在差异；第二个例子中，在句子中插入不同的字符，对语义的影响也存在不同。

我们设计了两个基本模型解决文本相关性计算的问题，最终使用两个模型融合的结果。

2.3 相关性 GBDT 模型

首先介绍使用 GBDT 模型计算 query 和 document 文本相关性。通过特征工程从人工标注数据中进行学习。有如下几大类特征：

- 特征工程
  - 匹配特征(匹配位置，最长匹配，同义词，编辑距离 ...)
  - 统计特征(匹配比例，重要词命中比例 ...)
  - 浅层语义特征(embedding相似度)
- 样本
  - 人工标注，按相关程度标注为0-3分的离散分数

3	鬼吹灯黄皮子坟	《鬼吹灯：黄皮子坟》首爆预告
2	海参的做法	福建葱爆海参的家常做法
1	锦绣未央33集	锦绣未央：唐嫣受伤

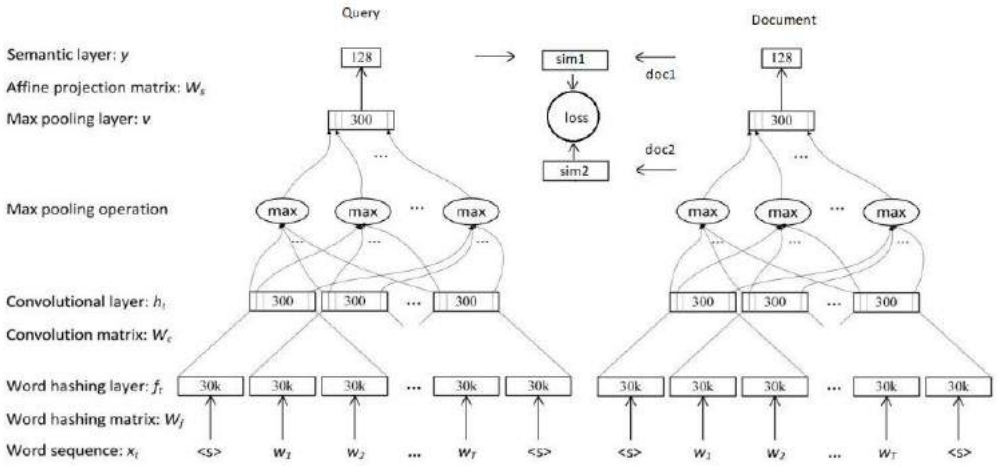
在图中的样本打分为 1 的样本 ,query 中的核心词 " 33 集 " 在查询结果中没有命中，核心词未命中是模型中很强的一个特征。这两句话在浅层语义上差异也很大。在 " 锦绣未央 :唐嫣受伤 " 中主体有 " 锦绣未央 " 和 " 唐嫣 " ,而在 query " 锦绣未央 33 集 " 中主体是 " 锦绣未央 " 和 " 33 集 " 。打分为 2 的样本，主体葱爆海参对海参做了限定，但没有超出海参的范畴。

在匹配特征设计中需要分析命中词权重、是否命中同义词、未匹配词是否为停动词等，例如 query 中的词没有在 document 中出现。在上图中标注 3 分的例子中，去掉不重要词 "："，整个 query 依然是连续命中，该项特征得分就较高。还有就是同义词的处理，这里需要同构离线挖掘同

义词的表。类似的特征还有编辑距离，逆序对的数量，考虑词权重的匹配等等。

相关性计算还有一种方法是使用用户行为日志，利用用户更多点击和未点击文档构建训练样本对。与经典的基于关键词匹配相关性算法相比，DNN 模型极大的提高了语义相关性的判断范围和准确性，对于长尾查询的效果也更好。DNN 模型使用的训练样本数据量比较大，我们的场景中目前用到亿级别样本。下图以 C-DSSM 为例（基于微软的论文）。中文与英文相比处理有一点差异，需要先分词之后是取单字和词的 embedding，另外这里推荐 trainable 的 embedding 。模型能够学习出查询与文档的量化表示，并计算相似度。我们还做了一些改进包括输入层增加额外信息。

C-DSSM



Ref : Learning Semantic Representations Using Convolutional Neural Networks for Web Search

获取 DSSM 模型处理的结果后，进行模型融合，利用好两种模型的优势。

人工标注样本的优点是精度高，交叉标注可以使得准确率达到 90%以上。从用户行为中收集的样本，样本精度大约只有 70%，优势是数据量大长尾覆盖度好，能将 query 与 document 的语义结果放在同一个向量空间。以下是两种方式比较具体结果：

三、意图识别与词权重

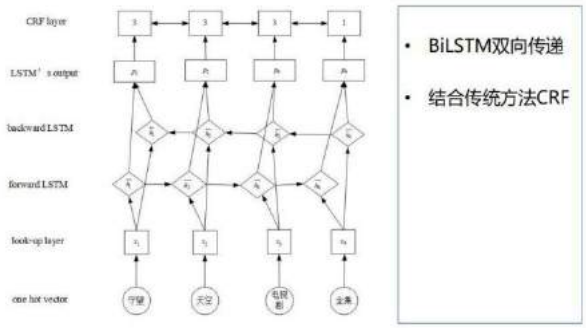
首先看下意图识别的例子，根据输入来识别出不同的意图：

输入是 " 男士用洗面奶洗脸的正确方法 "，在识别出意图后，放到 " jingyan " 、 " zhidao " 等垂类索引中去搜索，能够提升性能并为后续的排序等流程提供信息。

这个项目早期解决的方法是词典+规则的方式，这种方法是精度高、依赖人工挖掘，覆盖度有限，规则之间可能冲突，维护成本高。后来改进为在词典+规则的基础上，使用 LR 和 DNN 两个模型计算意图分数。

使用 FTRL 优化的 LR 模型做意图识别能获得在线学习的方法的好处，是因为意图是动态变化的过程，这使得线上模型能够快速的更新。这里将意图识别建模分为多个二分类问题；构建的特征有 N-Gram 、长度特征、首位位置特征等；单类别的样本量在千万级别。此外还使用了 DNN 模型做意图识别，把多类别的样本放到 DNN 模型中，需要数量保持一定程度的均衡。

在对 query 进行分词后，还需要计算分词结果的词权重，最容易想到的方法是 IDF，这种方法比较简单应用在工业级系统中效果不够理想。因为一个句子具有整体含义，是一个上下文相关的问题。传统的方法是使用 CRF 模型来进行计算，目前 SOTA 的方法是用 BiLSTM+CRF 的方法。



四、点击模型排序

用户的浏览和点击等行为数据除了计算相关性还可以用来改善排序的效果。

移动搜索业务与传统的网页搜索有一个明显的区别，传统的网页搜索结果，不同的网页之间有类似的结构，移动搜索排序面临的一个挑战是异构结果。比如搜索结果中其字段表述的含义跟视频跟应用是不同的，视频里面分级、主演，应用是有哪家厂商发布/应用类型，这些信息在不同结果之间是对不齐的，网页搜索场景中的排序一般是同构结果排序。业务中采用了点击模型进行异构结果的排序。下面介绍几种点击模型： PBM 、Cascade Model 、DBN 。

Expression	Meaning
u	A document
q	A user's query
r	The rank of a document
u_r	A document at rank r
r_u	The rank of a document u

Expression	Meaning
E	Estimate
A	Attractive
C	Click
S	Satisfaction

4.1 点击模型—PBM

$$
\begin{aligned}
&P(C=1|u,p) \\
&\quad = \sum_{e\in\{0,1\}} P(C=1|u,p,E=e)P(E=e|u,p) \\
&\quad = \underbrace{P(C=1|u,E=1)}_{:=\alpha_u}\underbrace{P(E=1|p)}_{:=\beta_p}.
\end{aligned}
$$

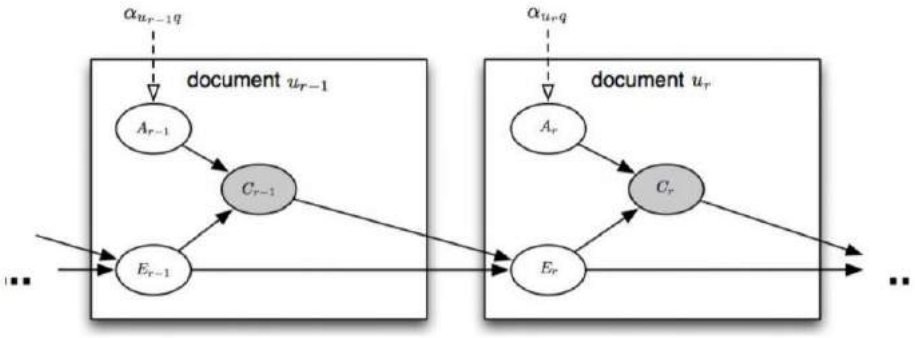
$$
\alpha_u = \frac{\sum_{i=1}^{N} c_i}{\sum_{i=1}^{N} \beta_{p_i}}.
$$

Ref : Chapelle, Olivier, and Ya Zhang. A dynamic bayesian network click model for web search ranking

PBM 是一个比较简单的基于位置模型，即结果与用户看的概率、实际看的概率来决定。

4.2 点击模型-CM

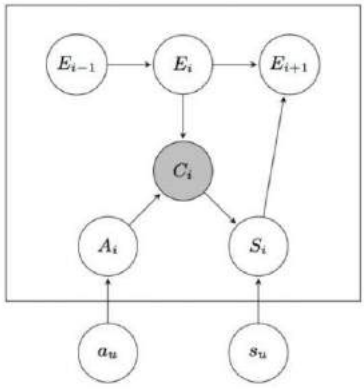
假设用户看结果是带有顺序的状态，过程如下：



$$P(E_1 = 1) = 1$$
$$P(E_r = 1 \mid E_{r-1} = 0) = 0$$
$$P(E_r = 1 \mid C_{r-1} = 1) = 0$$
$$P(E_r = 1 \mid E_{r-1} = 1, C_{r-1} = 0) = 1$$

4.3 点击模型-DBN

使用 DBN 的假设包括，第一点是用户点过，并不代表对结果完全满意，即这是概率事件。另外一个假设是给出的所有结果中，并不一定有用户满意的结果，用户可能都不太感兴趣而放弃搜索。DBN 的训练过程中用到了 EM 算法。



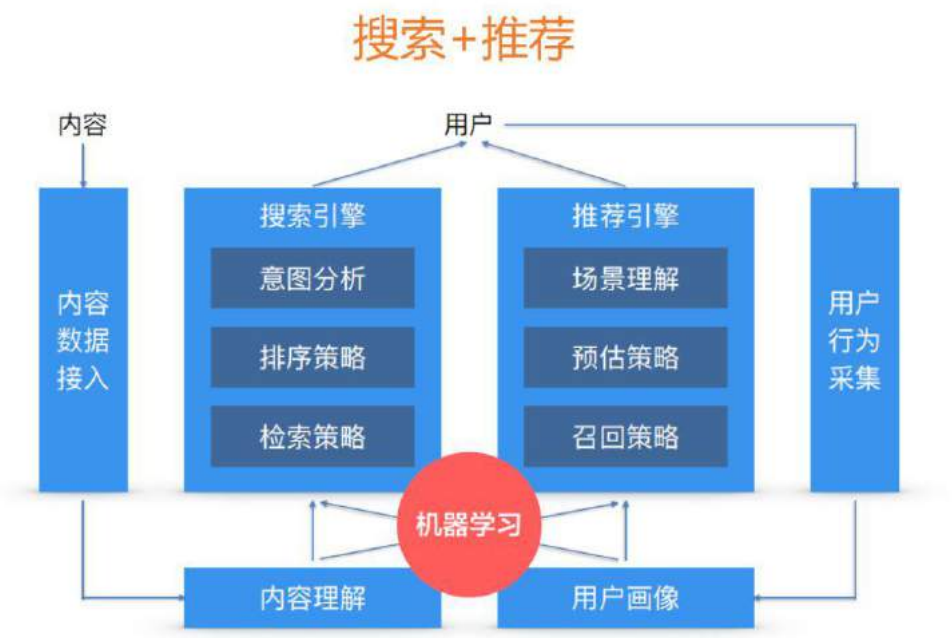
$$
\begin{aligned}
A_i = 1, E_i = 1 &\Leftrightarrow C_i = 1 \\
P(A_i = 1) &= a_u \\
P(S_i = 1 | C_i = 1) &= s_u \\
C_i = 0 &\Rightarrow S_i = 0 \\
S_i = 1 &\Rightarrow E_{i+1} = 0 \\
P(E_{i+1} = 1 | E_i = 1, S_i = 0) &= \gamma \\
E_i = 0 &\Rightarrow E_{i+1} = 0
\end{aligned}
$$

我们使用了多模型组合并对模型针对移动搜索场景做了额外的优化。模型中考虑是否有图以及结果的大小等带来的 bias 。

例如下图的 query = " 赵薇老公 "，改进效果对比：

五、总结

搜索推荐的整体架构：


搜索+推荐
内容
用户
内容数据接入
搜索引擎
意图分析
排序策略
检索策略
推荐引擎
场景理解
预估策略
召回策略
用户行为采集
机器学习
内容理解
用户画像

搜索引擎中的 web 数据挖掘

嘉宾：沐沐老师　编辑：赵丽

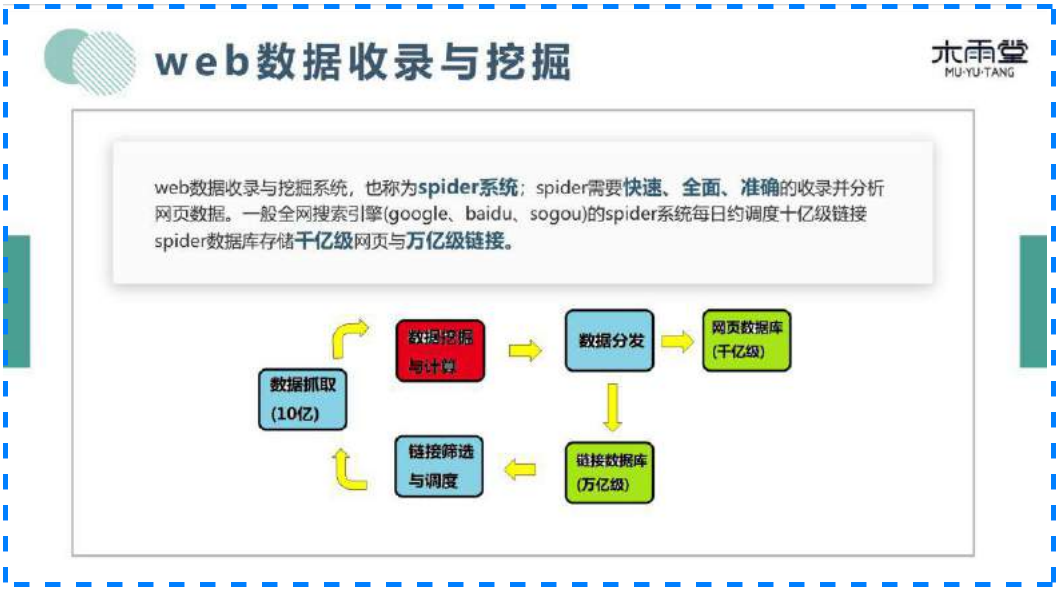
导读：今天为大家分享的是搜索引擎中的 web 数据挖掘。首先介绍下搜索引擎。实际上，我们每天都会使用的搜索引擎，我们会输入关键词 query 和需求，搜索引擎会根据算法将于 query 最相关且最权威的结果呈现给用户。

搜索引擎有 3 个核心部分：

第一：理解用户行为，从最初文字输入到语音和图片输入。

第二：收录并分析互联网数据。

第三：将用户行为与数据建立联系，为用户推荐结果。



今天主要介绍第二部分内容：互联网 web 数据分析技术。首先看一下 web 数据挖掘与收录系统，也就是 spider 系统。Spider 系统是比较成熟的技术，目标是：快速、全面、准确的收录并分析网页数据，一般全网搜索引擎类似 google、百度、搜狗的 spider 系统，每日调度 10 亿级链接，Spider 数据库存储千亿级网页与万亿级链接。我们可以认为 spider 系统是一个环状的系统，从链接数据库开始，经过相关算法：如链接调度与筛选相关算法，对周期内需要调度的链接进行抓取，然后发给抓取器，进行网页的下载和渲染。然后输入到数据挖掘与计算系统中，对内容进行提取，将网页中的内容保存到内容数据库，链接保存到链接数据库中。我们今天的重点是网页下载之后，网页相关的一些数据挖掘算法。

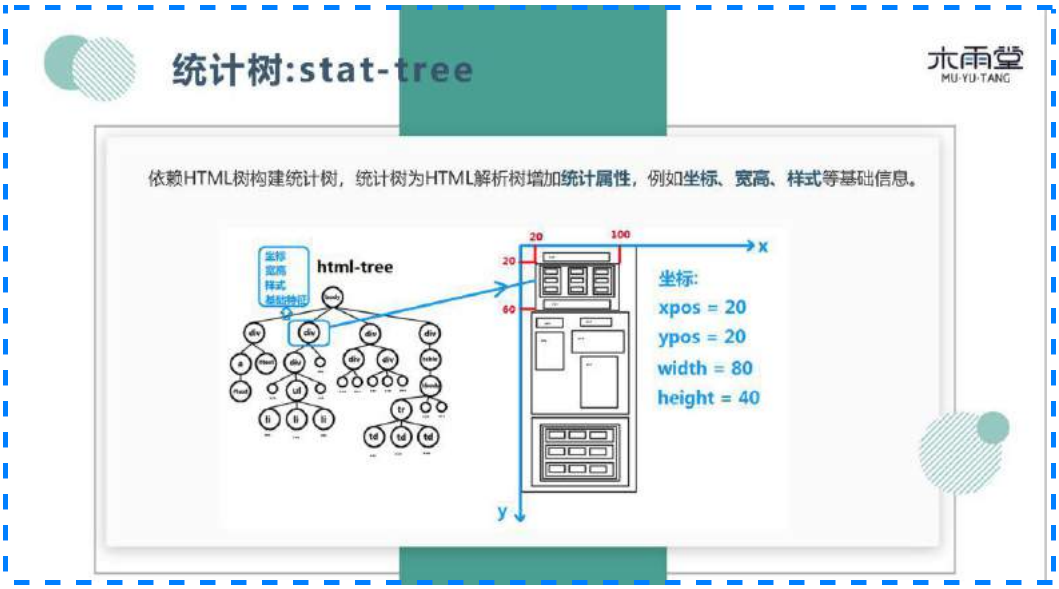
首先我们看一个例子，互联网用户会浏览新闻、微博等各式各样的网页。我们观察一下网页，思考两个问题，第一个问题，在你观察的网页有什么特征及内容。第二个问题，从整体看来，这是怎样的网页，如何描述这个网页。具体的分析如下：

Web 分析技术最核心是对网页特征提取，比如：导航、标题、时间、主图等等。网页内部特征是数据引擎进行排序时用到的关键特征。例如在排序中，时间特征也比较关键，将实效性高且相关网页排在前面。除内部特征之外，还有网页整体特征。我们会用到一些分类和自然语言处理的技术，刚才的例子就是一个新闻网页，内容是科技相关的。会同时计算语义特征和结构特征，例如主图、排版、文本段落等。

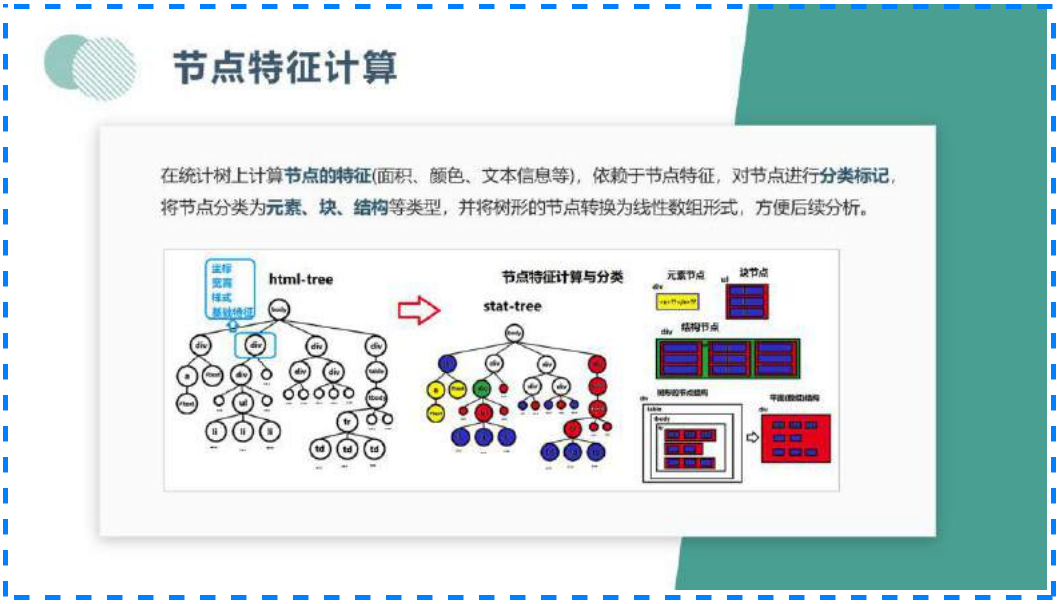
Web 数据挖掘中，需要计算并提取网页的百余个网页属性字段，在此之中会使用一些技术，如机器学习的分类、聚类、回归、自然语言处理、规则聚合、主题模型等等。最终的目标是充分的理解网页，为搜索引擎排序提供准确的网页属性。

网页分析用到了许多基础的算法和数据结构。获取网页属性，需要用很多模型计算相应的字段，低层需要构造很多数据结构。了解网页首先需要建立一个 html 树的结构。在这个数据结构中，如图蓝色的代表标签节点，绿色代表文本节点。文本节点是 html 的内容，内容可以展示在浏览器中。在分析过程中，一些内容比较关键，如标题、超链接与文本内容等相关重要字段。

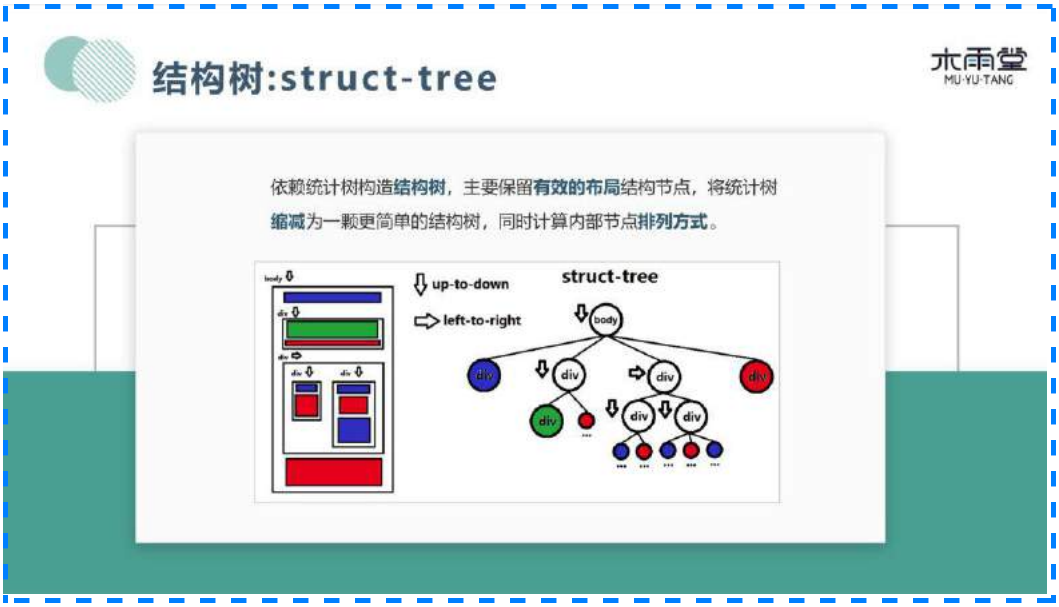
下面介绍页面分析的一个算法：我们希望充分的理解网页，在此过程中，我们可以通过算法划分出不同的区域。如划分为上下左右区域，根据不同的区域提取字段，最终可以对网页进行更深层次的理解，比如说主体边框、标题、关键内容识别理解。



在将网页划分为不同区域的过程中，需要用到较多的数据结构。例如：依赖 HTML 树构建统计树。所谓统计树就是在建立了 HTML 树之后，对树节点增加统计属性，例如坐标、宽高、样式等等基础信息。往往在识别节点类型（正文节点、图片节点）时，这些基本信息可以帮我们排除没有意义的节点。



在构造树的过程中，会对节点的特征进行计算，例如节点的面积、颜色等。依赖这些节点特征对节点进行分类标记。例如说网页的布局节点，js 节点，这些具有对应的特征属性，基于此对网页进行分类。分类之后将树形的节点转化成数组的形式，方便后续分析。



如左图，该网页包含 body 主体，布局节点从上到下一次排开。同样在右图的统计树上，根节点就是 body 节点，下面的四个子节点代表网页块元素。在统计树上进行层次遍历，以保留有效布局结构节点，将树缩减为一个更加简单的树。在遍历过程中，需要算法来计算节点的排列顺序。例如根节点下有 4 个子节点，分别代表网页中的不同布局块。左图四个布局块是从上到下的排列方式，在右图中会根据深度进行标记。

构造完结构树之后，还会对树进行变换：节点删除，节点压缩。左图网页中橙色的节点，对于表示页面意义不大，原因是：根节点下的子节点都是从上到下排列的，橙色节点的子节点排序顺序相同，因此可以对橙色节点删除，将其子节点上移。

总结：为网页划分区域，为构造平面数据结构做准备。我们从 html tree 做了许多化简得到 layout tree，根据 layout tree 计算区域，从而得到多个分区。

如何根据 layout tree 划分平面，划分思想就是构造划分平面的容器（类比与 c++ 中的 vector）。 我们按广度优先遍历 layout tree，来判断节点是否可以放入到容器中。可以使用规则或者机器学习模型来进行判断。如图中黄框的节点，面积比较小，宽比高的值较大，且包含一些语义信息：导航等，就可判断节点属于上区域容器中。但是 5 号节点面积较大,判断为上区域是不合适的。那么上区域容器收集完成了。所以 5 号节点会在下一层进行相应的划分。也可以通过分类模型来判断是否要放入到容器中。

完成平面构造之后，可以利用它们对网页中元素提取。例如图中在提取评论内容，训练一个评论块识别模型，就可以从不同网页中，商品、新闻等网页中提取评论内容。

最后介绍网页分类，这是一个比较通用技术，无论是结构分类还是语义分类思路是比较类似的。例如对网页分类，就是对提取网页的特征向量（如图片数量，链接数量等等统计信息），利用有监督的机器学习模型（如随机森林、逻辑回归等）进行训练，识别网页类型。一般的网页分类系统是多个二分类模型叠加在一起的。例如有多个模型：娱乐，体育新闻等，最终我们会对这些模型结果串联到一起，这样每个类型相互没有依赖，升级迭代比较方便。

本次分享的技术部分就到这里，谢谢大家。

利用高效率的补贴策略驱动用户增长

嘉宾：白朔天　编辑：张祥

本次分享的内容有：

- 什么是用户增长
- 增长的必杀技是补贴
- 智能化补贴的框架

关键词：增长、补贴、效率

增长是目标，补贴是手段，效率是评价。

用户增长

定义：用户增长是用户的新增量大于用户的流失量，就像一个游泳池一样，在动态维持用户的增长。增长包括两个方面：人数（用户的绝对数量）和质量（每个用户贡献的价值）。

整体上可以用一个公式来表达：

规模=(DAU + growing - dying) * 质量

当 growing > dying 时，表明用户量是增长的。若平均质量也在增长，说明整体盘子的用户质量很高，也是属于增长。所以做增长，需从人数和质量两个角度着手。

1. 增长模型的演变

随着时间的变化，增长模型发生了如下变化：

图 1：用户增长模型演变

共同点：增长模型依然是 5 件事情，只是顺序发生了改变。

不同点：在互联网的早期，各个垂直领域都是缺少竞品的空白区域，平台属于野蛮生长时期，最需要做好获客，适合 AARRR 模型。而在当下各个细分领域的供需双边都饱和的情况下，获客成本越来越高，且留存的成本远比获取新用户成本低，所以做好留存的重要度明显比获客高。

2. 用户生命周期曲线

将用户从了解某 app 到离开 app 的过程划分为 6 个阶段，分别为**潜在期，新手期，成长期，成熟期，衰退期，沉默期。**按照用户价值与时间的关系如下所示：

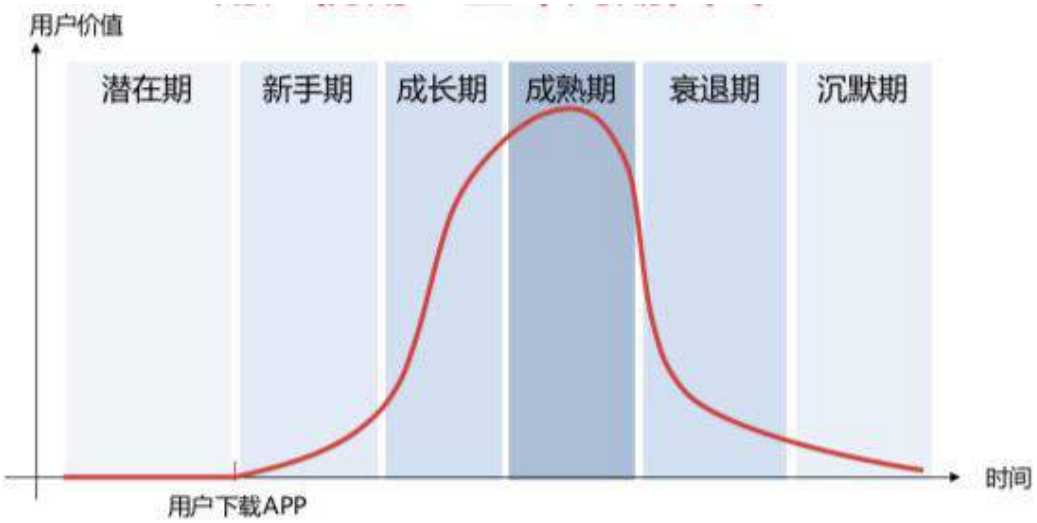


图 2：用户生命周期曲线

很明显，用户在成熟期价值最大，其次是成长期和衰退期。那么，不同生命周期，我们能做什么呢？

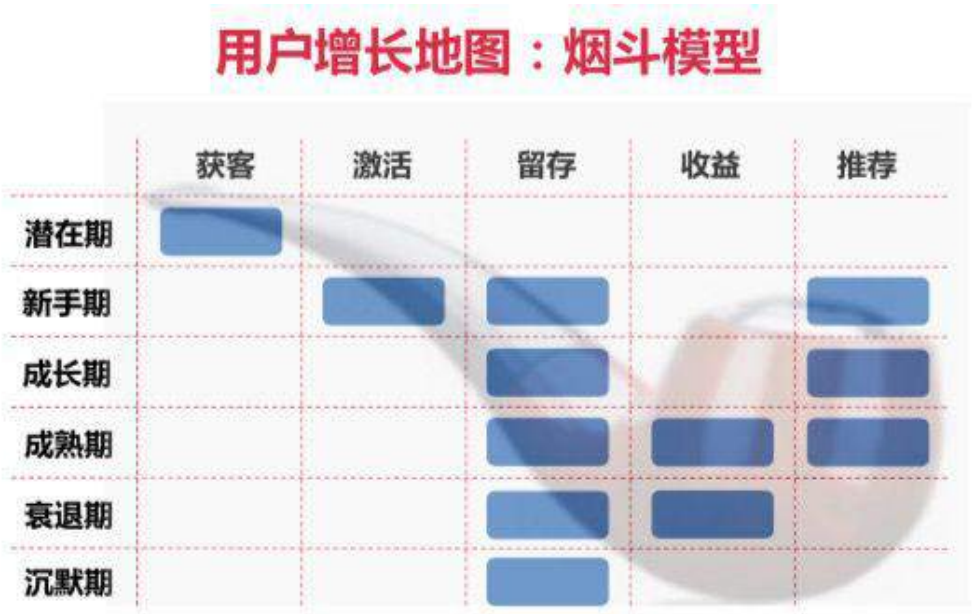


图 3：AARRR 模型应用于用户生命周期中

从图 3 来看，图形类似于一个烟斗，这里称呼为烟斗模型，烟斗模型解释了每个时期，我们能对用户做哪些操作，使用用户给平台带来更大价值。

获客：发生在潜在期，对潜在用户进入引导，让其进入平台

激活：发生在新手期，用户进入到平台，引导用户熟悉 app

留存：从用户进入到 app 到离开 app 都要做留存，以最大程度的延长用户生命周期。

收益：即让用户向平台付费，主要发生在用户使用 app 一段时间后，即成熟期和衰退期。

推荐：品牌认可，用户将 app 向好友推荐，获取优质的用户。发生在新手期、成长期、成熟期。

增长的必杀技是补贴

定义：通过补贴，让用户尝试使用产品，给予用户好的体验，并让其在未来留下来继续使用 app。

1. 补贴案例

案例 1：加班补贴：用补贴换时长增加

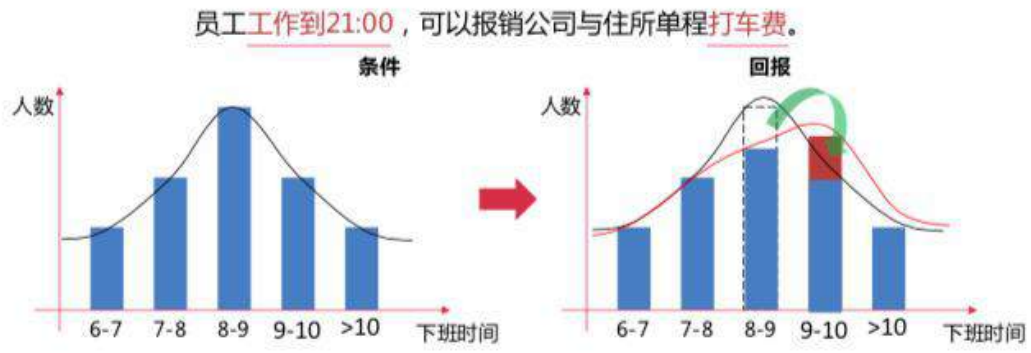


图 4：调整前后，不同时间段下班的员工数量分布

实施打车策略后，原本 8-9 点下班的员工会工作到 9-10 点，对公司而言，付出了打车费用，总的工作时长增加了，但 ROI 是否会增加具有不确定性。

方案 2：补贴增强用户体验

图 5：优惠券抵扣，增加体验

对有打车需求的用户，给予补贴。通过平台让利的策略，提高用户发单的概率，以提高平台营收。

小结：用补贴的方式使更多用户来体验产品，让用户更加频繁的使用 app 并留下来。常见的补贴有现金红包、现金券、折扣券、积分、实物奖励等。

2. 该如何补贴

发小钱办大事，提高补贴效率，补贴效率公式：

$$ROI = \frac{\text{补贴后的效果} - \text{补贴前的效果}}{\text{补贴金额}}$$

分子为补贴后的效果减去补贴前的效果，即补贴收益 ；补贴金额等于活动投入的金额 ,即补贴成本 ;ROI 等于补贴前后的收益处以补贴金额。若想使 ROI 呈现递增趋势，在补贴成本不变的时候，使补贴前后收益增加，或补贴收益不变的时候，减少补贴成本。

3. 如何提升补贴效率

提升补贴效率的关键因素：市场供需

图 6：供需关系

投放补贴，应该围绕市场供需来做，满足用户的需要，驱动增长。对于增长来说，提高 ROI，有两种方式，保持分子不变，分母变小，即补贴前与补贴后的收益效果不变 ,认为市场需求达到了天花板 ,通过减少补贴 ,驱动 ROI 增长；保持分子变大，分母不变，认为市场有供给需求，即市场供需不足，在补贴成本不变的情况下，带来了收益增加，就需要提高补贴效率。

智能化补贴

1. 必备条件

多部门合作，需具备人工智能、心理学、经济学知识。

图 7：智能补贴协作

2. 优化思路

如何对上文案例 1 中的打车补贴与工作时长关系优化 ,最大化 ROI 呢？下面从两个角度来优化策略：**补贴敏感度**和**市场供需环境。**

角度一：补贴敏感度

问题：无效补贴

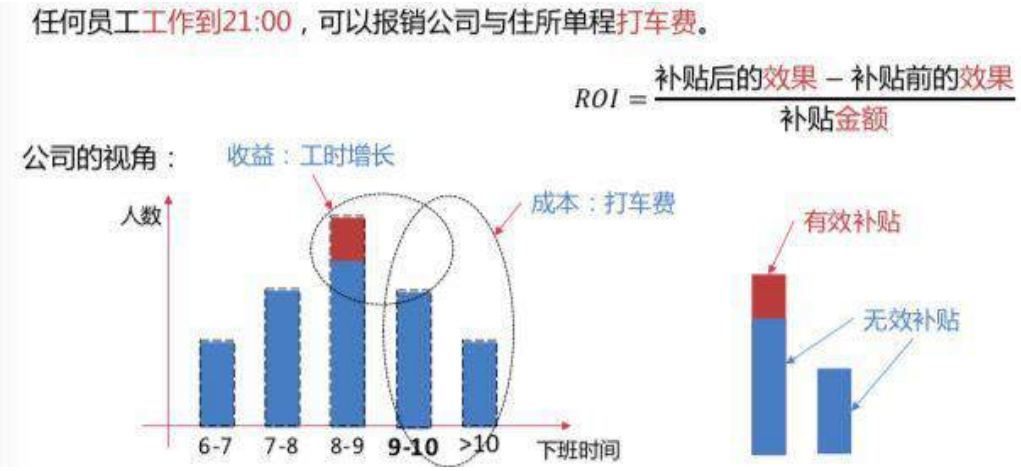


图 8：减少无效补贴

明确哪些因素会影响收益和成本，找到对收益没有影响但增加了成本的因素。在打车补贴案例，实施补贴策略后，会有部分原先不属于补贴范围的员工向补贴区域转移，使得享受补贴的人数增加，补贴成本也会增加，收益是转移进入补贴范围用户的工时增长。对于原先工作到 9 点后下班的员工，工作时长没有增加，补贴成本增加了，这部分人的补贴就是无效补贴。

对于无效补贴的用户，是需要待优化的。

3. 优化策略

① 引导用户增加工时

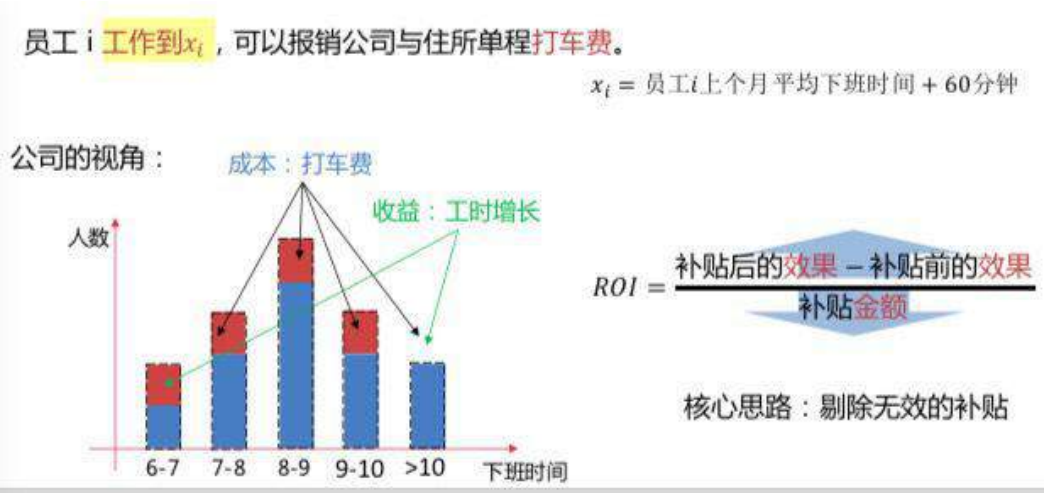


图 9：引导用户增加工时

实施打车方案后，工作时长没有得到有效增加，但支出增加了。结合员工历史下班时间，如果员工工作时长增加 60 分钟，就给予打车费报销。采用策略

引导员工增长工作时长，达到了预期的工作时长，对其给予补贴。对公司而言，获得了更多的工作时长，同时剔除了无效的补贴，ROI 的分子变大，同时分母变小。

② 弹性补贴

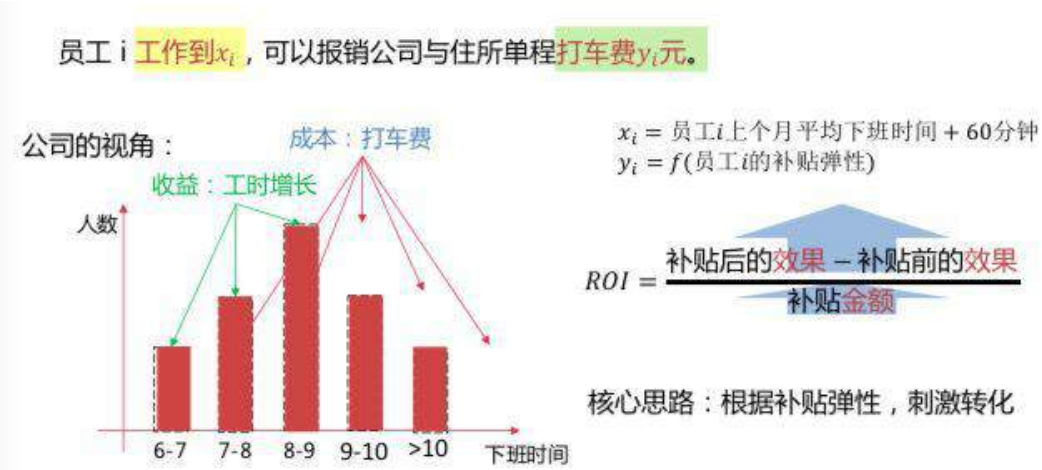


图 10：弹性补贴

在策略 1 的基础上，继续优化补贴成本，对不同的员工给予弹性的补贴，从整体角度来看，收获的工时不变，公司投放的成本减少，ROI 增加。

③ 价值转化

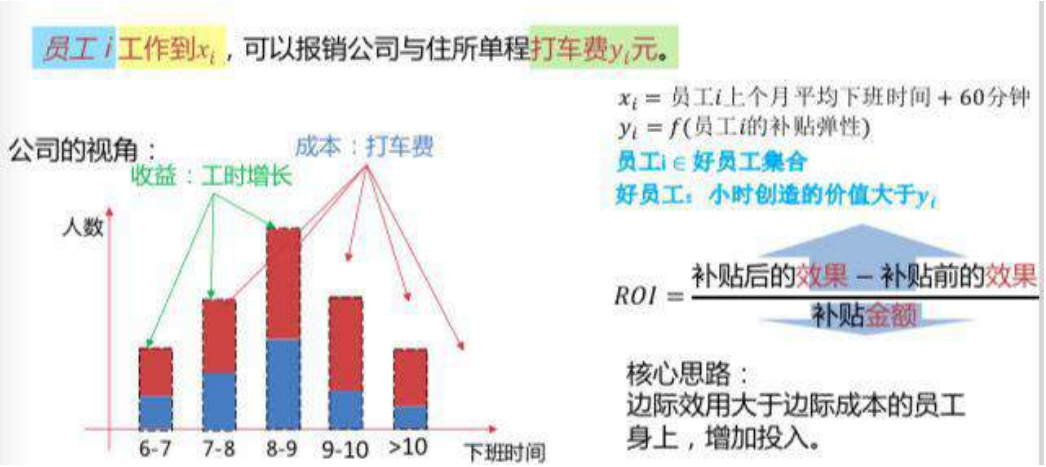


图 11：员工边际效用与边际成本

在策略 2 的基础上，考虑每个员工的边际效用与边际成本，若是边际效用大于边际成本，给予补贴，否则不给与补贴。同时，给予补贴的员工给予弹性补贴。整体上来看，工时增长了，投入的成本减少，ROI 减少。

角度二：市场供需环境

这时，结合市场的供需环境，给予员工福利待遇：

④ 增加补贴，换留存

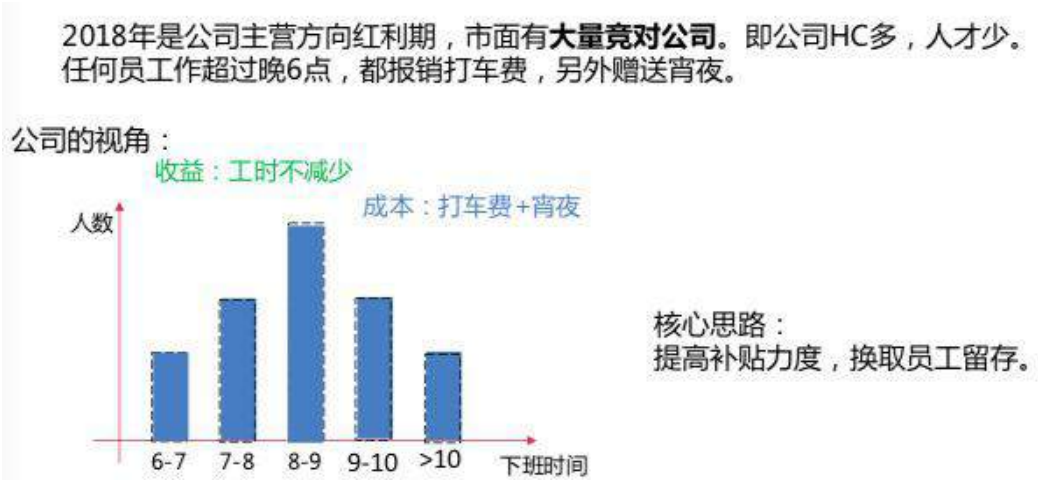


图 12：增加补贴，提高留存率

⑤ **减少补贴**

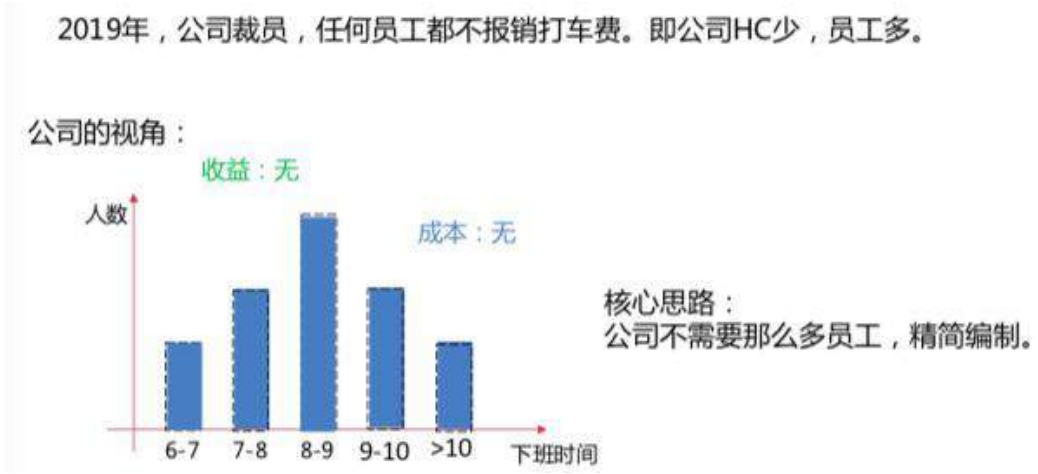


图 13：市场环境恶劣，对员工不利

4. 复盘

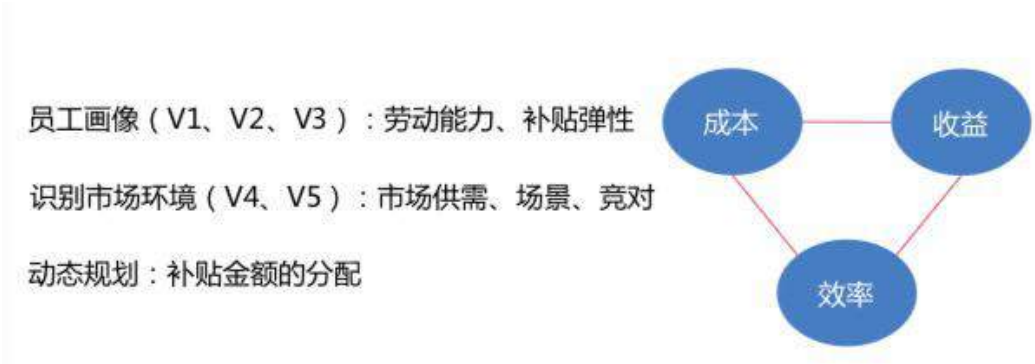


图 14：复盘

前面 3 版本，从员工画像角度考虑，利用大数据千人千面的能力，对员工的劳动能力、劳动价值、对钱的敏感度做出运营策略；第 4 和第 5 个版本，结合市场的供需环境、企业提供的岗位数量之间的供需关系做出的策略。整体上，就是补贴与收益的动态平衡。

精细化营销框架

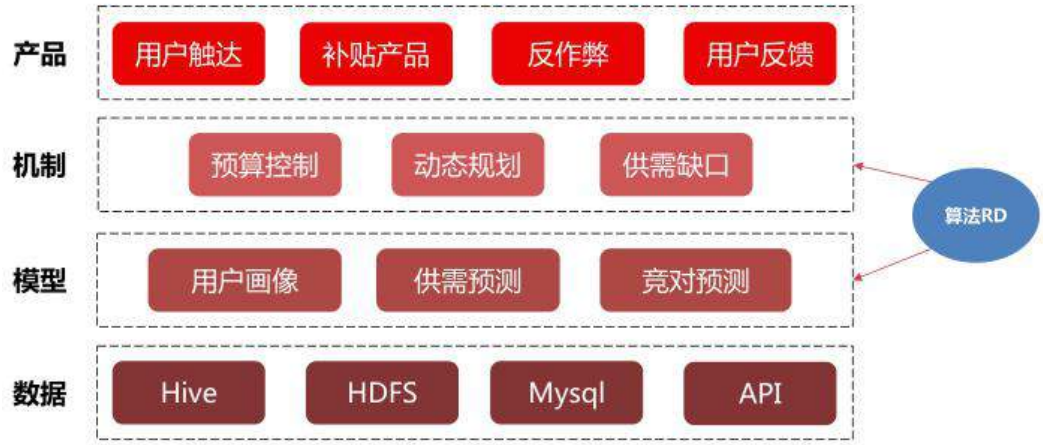


图 15：精细化营销引擎系统框图

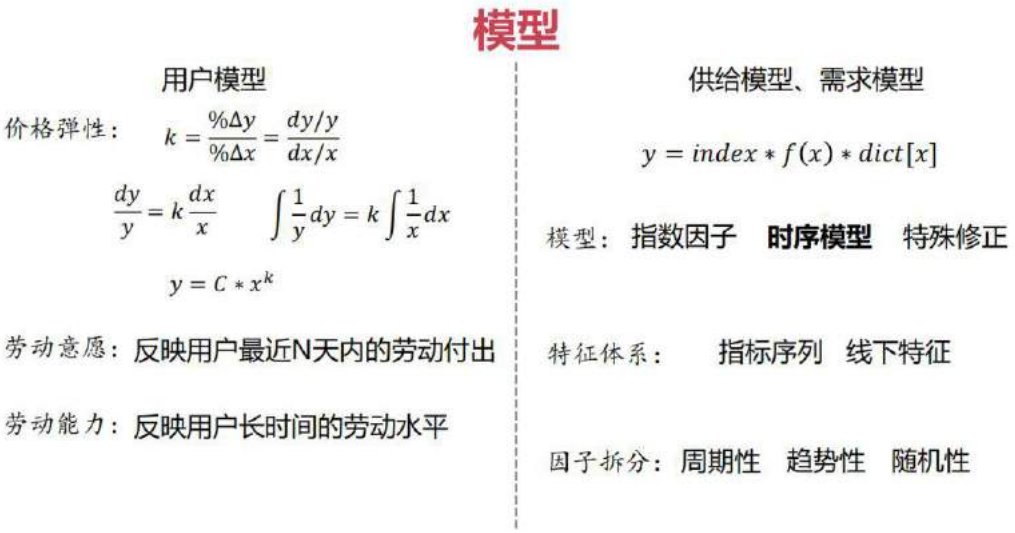
每个框中包含了相应的数据和技术要求。底层搭建系统和提供数据支持，模型层将数据加工成用户画像以及做预测问题，机制层结合用户画像、预测结果等设计动态收益模型，保持平台盈利和生态平衡，最上层是产品，直达用户。

1. 算法工程师工作规范

图 16：算法工程师工作规范

明确目标，知道到底在做什么事情；有合理的指标体系来分解目标；从指标中提炼出行之有效的方法，也就是机制；机制抽象为模型，做成系统化的需求；最后是评估，对策略做数据评估，以便于迭代优化。不同职业发展阶段，不同方面的能力的重视程度不同，在刚步入职业阶段，模型尤为重要，越是职业发展后期，指标和评估更为重要。

2. 模型



用户模型：刻画用户能力和劳动意愿。

供需模型：预测类型，包含时序模型、指数因子模型、特殊修正模型，其特征体系包括指标序列、线下特征，还具有一定的周期性、趋势性、随机性等特征。

2. **补贴机制**

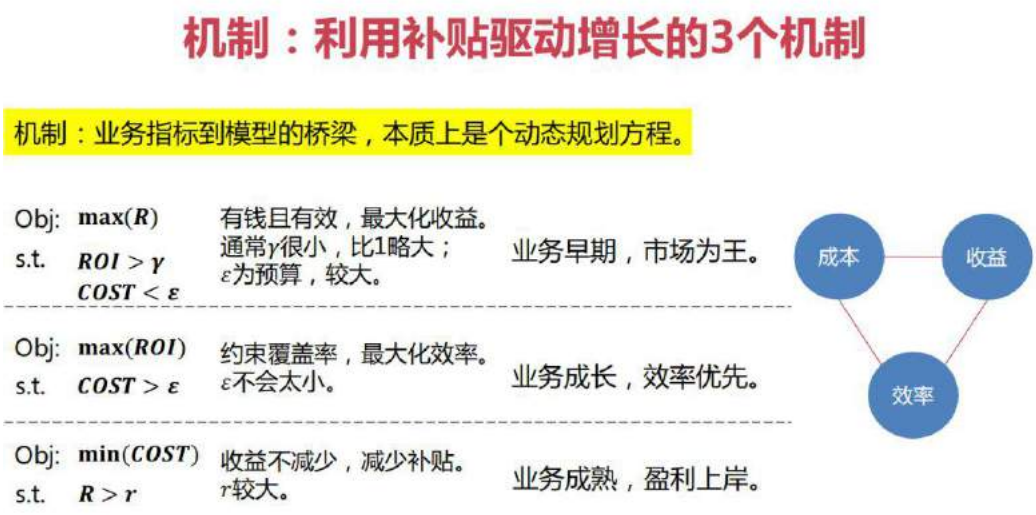


图 17：补贴机制

第一个：在业务早期，验证市场有效，目标是最大化收益，同时保证投入就可以带来收益；

第二个：在业务快速成长阶段，加大投入，提高业务覆盖面，最大化 ROI，保持投入效率最大化；

第三个：在业务成熟阶段，保持收益不减少，减少补贴，提高 ROI。

4. 产品思维

信息不对称，供给不足：在信息不对称或垄断的情况下，采取涨价措施，提高收益。

损失厌恶的心理：利用损失厌恶的心里，换取未来一段时间的活跃度，适用于竞争态，提高用户忠诚度和留存度的场景。

通过不确定性的奖励来增加趣味性。

5. 风险

可能的风险：用户感知杀熟，认为不公平的情况。

如何规避：

规则透明：明确告知用户规则机制。

提高定价，反向补贴，完成差异化定价，先提高定价，再给用户实惠折扣来吸引用户下单。

总结

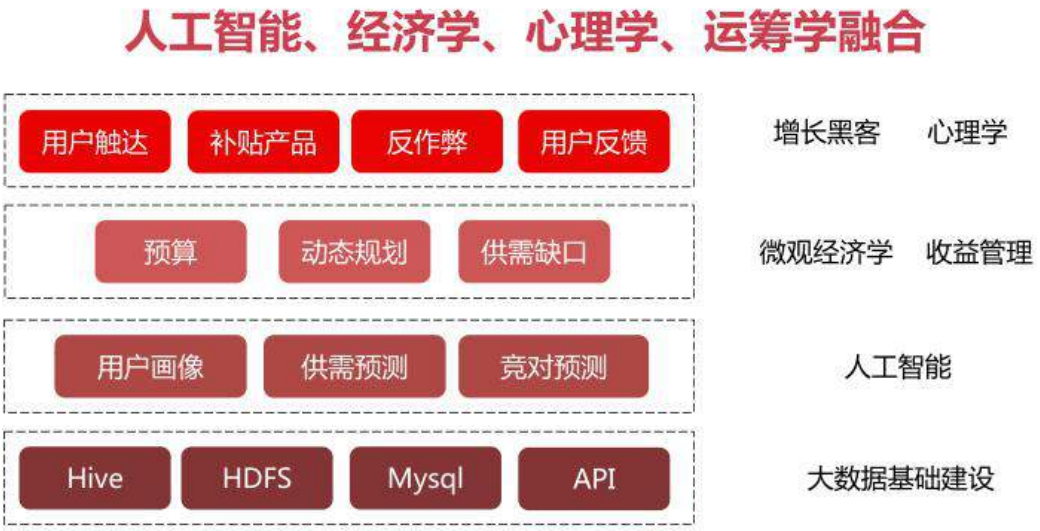


图 18：跨学科能力

智能化用户增长，需要具备多项技能，技术使得我们具备接触数据、熟悉数据、加工数据的能力，人工智能帮助我们准确的预测用户，微观经济学帮助我们理解收益平衡机制、明确 ROI 增长思路，黑客增长和心理学拓宽我们的思维和了解用户，拥有更多的途径来实现策略的落地成为产品，更加好用的产品直达用户。

风控篇

携程金融大数据风控算法实践

嘉宾：曾凡祥　编辑：贾冬雪

导读：本次报告曾老师主要分享了携程金融风控算法从 0-1 的发展进程，以时间为主线，经历了数据样本由少到多，特征由粗到细，模型由简单到复杂，效果由坏到好的全过程，重点以申请评分模型和反欺诈模型进行阐述，是一次很好的风控模型实践报告。

「业务介绍」

模型一定是基于业务的，所以首先介绍一下携程的业务情况，即本文所提及模型的实际应用场景。携程金融主营业务有三大模块：

- 消费金融，包括消费分期（拿去花）和现金分期（借去花）；
- 信用卡；
- 供应链金融。

Fig . 1 携程金融的主要业务范围

「数据中台」

模型亦是基于完善的技术平台的，携程金融数据中台可以抽象为三个层次，底层是基础数据层，中层业务抽象模型层，以及最上层的算法模型层。

Fig . 2 携程金融大数据中台的抽象结构

中台这个概念早期是由美军的作战体系演化而来的，技术上所说的“中台”主要是指学习这种高效、灵活和强大的指挥作战体系。比如电商领域，经过十几年的发展，组织庞大而复杂，业务不断细化拆分，也导致野蛮发展的系统越来越不可维护，开发和改造效率极低，也有很多新业务不得不重复造轮子，因此业界诞生了不少知名中台系统，最著名的是阿里云的数据中台建设。

在携程金融内部，大数据中台的目标是为了解决效率问题，同时降低创新成本。

- 中台的目标：减少沟通成本，提升协作效率；

- 中台的实现手段：制定标准/规范、提供高可用数据/算法/应用服务、提供统一、标准的数据研发工具；
- 中台的原则：数据资产的集中管控，分布式执行。

携程金融的数据中台收集了包括携程 OTA 整个生态环境数据，框架的最底层是计算与存储资源层，其上是数据准备层，融合多数据源，并对其做了抽取，清洗，能够提供在线与离线的服务，使其能够为用户画像，特征引擎提供基础数据，并应用于模型算法。基础数据、用户画像、特征集市、模型服务这些内容以数据资产的形式来管理。基于这些数据资产，可以为各个业务线-获客，准入，经营，留存的全生命周期提供各种服务，框架图如下：

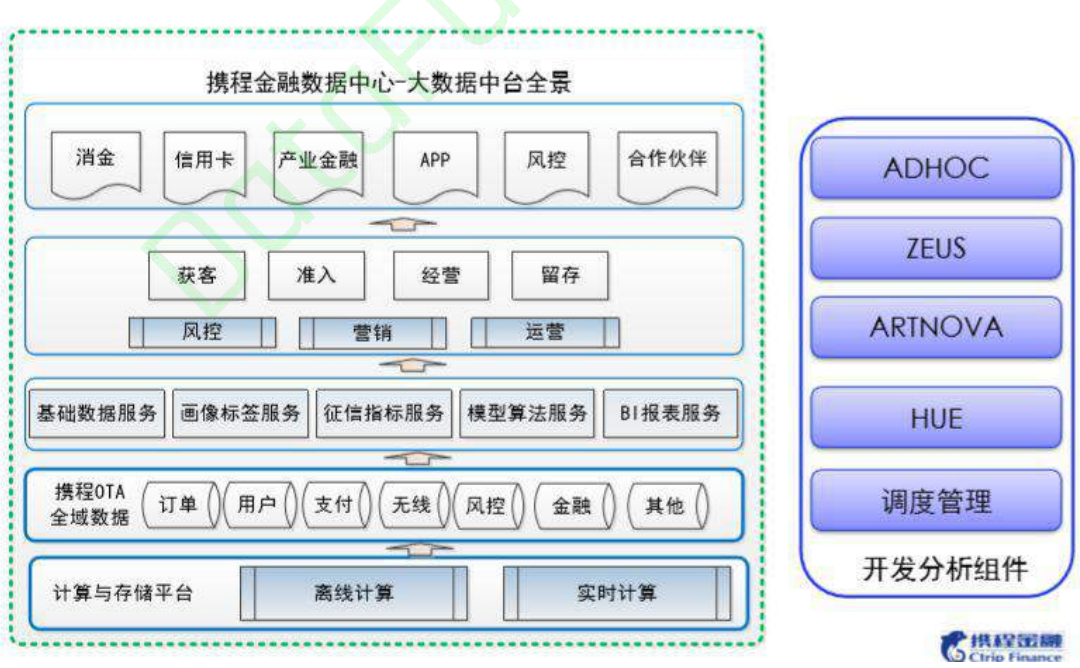


Fig . 3 携程金融大数据中台全景

「风控模型体系」

消费金融的风险大体可分为可控风险及不可控风险，算法能解决的主要是可控风险。可控风险包含欺诈风险、信用风险及作业风险；其中，欺诈风险指的是客户在发起借款请求时即无意还款，按照人数可以分为团伙欺诈和个人欺诈，欺诈者往往通过伪造身份信息、联系方式信息、设备信息、资产信息等方式实施欺诈；信用风险指的是借款人因各种原因未能及时、足额偿还债务或银行贷款而违约的可能性。不可控风险包括市场风险、实质风险及名义风险。

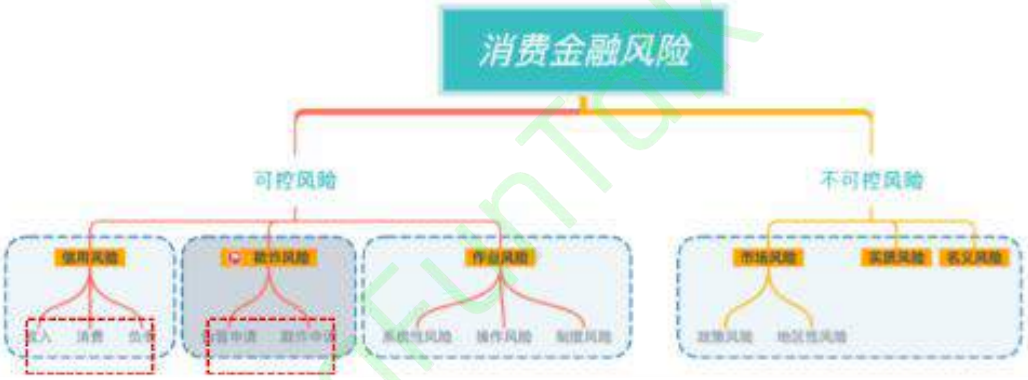


Fig . 4 风险基本分类

风控模型贯穿获客、准入、经营、逾期的整个客户生命周期，根据用户生命周期的不同阶段，可将风控模型分为贷前信用风险模型、贷中行为风险模型、欺诈检测及贷后催收模型。事实上，抓住信贷审批管理就能控制 80%的风险，一旦用户获得授信，后续的管理只能控制 20%的风险。除此之外，贷前、贷中、贷后不同场景，可以从不同的观测粒度进行建模与抽象。拿携程金融的业务来讲，可以从每一笔交易角度来看，也可以从携程生态中用户账户来看，也可以从自然人概念为核心的客户级别来看。一个自然人客户与账号可以是一对多的关系，一个账号与交易也可以是一对多的关系。

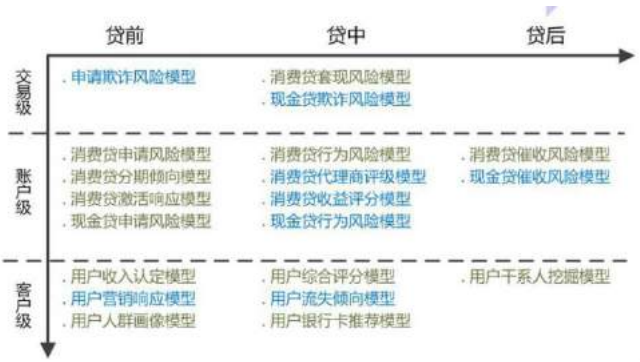


Fig . 5 携程金融风控模型体系

如今在银行、消费金融公司等各种贷款业务机构，普遍使用信用评分，对客户实行打分制，以期对客户的风险水平有一个准确的判断，并作为风险定价的重要手段。行业内常用的是 ABC 三张评分卡。A 卡、B 卡、C 卡分别表示：申请评分卡(Application Score Card)，行为评分卡(Behavior Score Card)，和催收评分卡(Collection Score Card)。

A 卡：在获客过程中用到的信用风险模型。从模型的角度来看，它会对用户未来一定周期内的逾期风险作预测，即模型会在用户授权的情况下收集用户多维度的信息，以此来预测逾期概率。预测的逾期概率被用于风控策略或者转换成信用评分，比如国外经典的 FICO 评分，国内的蚂蚁信用评分、京东小白评分、携程金融的程信分等。A 卡评分除了用于决定是否通过用户的信用申请，还用于风险定价，比如额度、利率等。

B 卡：行为评分是指，用户拿到信用额度后，模型根据用户的贷中行为数据，进行风险水平的预测。本质上讲，这个模型是一个事件驱动的模型，在互联网金融领域，一般会比 A 卡的预测时间窗口要短，对用户的行为更为敏感。B 卡除了可以用于高风险用户的拦截，也可以作为额度、利率调整的重要参考因素。

C 卡：催收评分会判断，例如当用户出现逾期时，机构应该先催谁，或者哪些用户不用催，就自动会把钱还回来。催收模型一定程度节约催收成本，提高回催率。

「风控模型体系之贷前信用风险模型(A 卡)」

贷前主要解决用户准入和风险定价问题，即面对一个新申请的进件用户，判断用户是否符合产品的放款条件及相应的放款额度、价格、期限等问题。主要包括三类问题：

- 反欺诈识别：根据用户提交的材料进行身份核实，确保用户不存在欺诈行为；
- 信用评级：与传统银行的信用评分卡原理类似，数据维度更加丰富，综合用户的社交数据、行为数据、收入数据等，判定用户的信用风险等级，评估用户的履约能力；
- 风险定价：根据用户的负债能力和收入稳定性，判断用户可承担的月供金额，确定用户的放款额度、偿还期限等，并根据用户风险等级确定用户的费率。这三个问题往往是互相影响、互为前提的。举个简单的例子，对一个月收入 3000 的用户来说，月供在 1000 左右，用户可能履约良好，信用等级良好；但如果月供提高到 4000，严重超出了其收入水平，即便不是有意欺诈，也可能出现断供的情况，从而得到比较差的信用等级。

本节重点阐述携程金融在信用风险建模（A 卡）上的演进和一些创新工作。

A 卡建模目前包括如下几方面：

- 确保策略的一致性，尽量减少人工干预，并利用机器学习的优势提升决策效率；

- 准确反映并量化用户的风险级别，策略人员可以控制和减少风险损失，因此对评分卡等级的排序能力、稳定性要求会比较高。

好坏用户定义：这一点可能是 A 卡甚至是互金大部分风控模型的最基础最核心的工作。对样本标签的定义，需要与实际业务场景、策略目标相一致，并综合考虑不同定义下的样本量。比如在现金分期场景中，可以画一下用户回款率（或者滚动率）和逾期天数趋势分布曲线，用户逾期 N 天以后回款率或者滚动率便已经趋于稳定(梯度平稳)，则可以 N 天以上逾期作为筛选坏样本的依据。在某些场景下，如曾经的 Payday Loan，由于整个业务周期只有半月或 1 个月，为加快模型迭代速度，有时甚至会定义 7+甚至 1+逾期用户为坏客户。在一些银行场景中，出于坏账计提考虑，可能定义 90 天以上逾期为坏客户。总之，好坏用户的定义不能纯靠人工经验，应该以场景的数据为基础，进行数据分析之后确定。

样本规模与算法演进：携程金融的业务最早开始于 2015 年，模型进行了多个版本的迭代。下面的表格展示的是拿去花业务中，我们 A 卡模型的演进。业务初期，样本数据量极少，往往根据相关业务经验确定使用的特征和规则；随着数据的慢慢积累，开始采用部分精细特征，使用简单的机器学习算法训练；当样本数据量积累到百万级以上，我们尝试采用神经网络算法进行特征自动提取或者 end-to-end 的风控模型训练；从 18 年上半年开始，我们升级到更加先进的迁移学习体系，未来会持续落地更多的前沿算法。总之，携程金融的风控模型优化的过程，实质是紧随着业务从无到有、从小到大，数据量由少变多，特征由粗到细，模型由简单到复杂，效果由一般到突破的过程。

申请信用风险模型(拿去花)						
-	V0.1	V1.0	V2.0	V3.0	V4.0	VX
样本规模		千级别	万级别	十万级别	百万以上	迁移学习、无监督学习+在线监督学习 ---------- TO BE CONTINUED ...
特征规模	100	40+	300+	800+	2000+	
算法选择	业务经验	LR	GBDT+LR	DNN	分形网络、CNN	
参数规模		40+	400+	百万级	亿级	
K-S值	16	31	40	49	52（E）	

Fig．6 携程金融拿去花业务 A 卡模型演进

模型的评估与监控：模型建立后，需要对模型的预测能力、稳定性进行评估。看模型效果不能只看 KS，KS 定义是从 0-1 概率之间好坏样本累计概率最大差值，实际应用中一般不会直接取这个阈值（cutoff）作为策略，因为在这种 cutoff 下，通过率可能会很低；风控不能不管业务，举个极端的例子，通过调整 cutoff，风控几乎可以做到任意想要的逾期率，但这样通过率就很低了，业务规模可能只停留在极少数资质优秀的客户；所以评估模型时，基于风险的评估及基于业务的评估是必须的。因此，模型评估可分为三层：

第一层：机器学习模型评估指标。信用评分模型常用的评估指标为 KS、AUC 等。 考虑到金融业务反馈周期长的特点，除了划分训练集、测试集外，通常会预留一段训练样本覆盖时间段之外的数据集，作为 OOT（跨时间）测试集，以测量模型在时间上的稳定性；

第二层：风控层面，比如在不同 bucket 下，预测概率的排序性能；

第三层：业务层面的拦截率，通过率，逾期表现等。

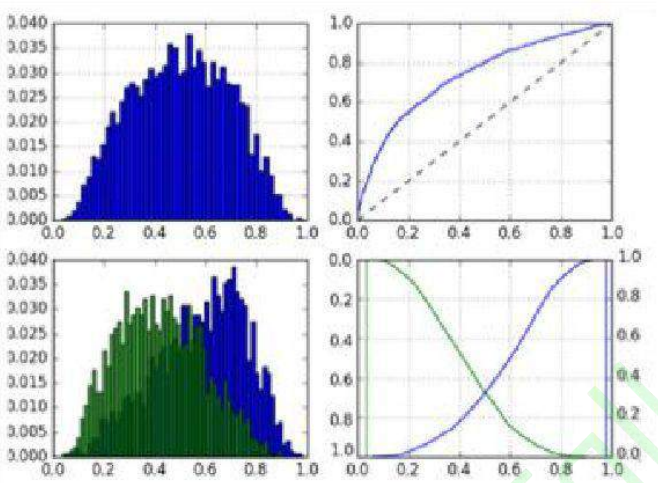
基于上面的评估分层，我们的监控也做对应的分层监控，除了包含上述三个层面，还对输入到模型中的特征进行监控，比如特征的分布、波动率等。

「风控模型体系之贷前信用风险模型发展历程」

经历过了完全靠经验的规则模型，当积累了一定数据量时，便可以用少量的维度与数据开始训练了，当数据量较少时，使用简单的 LR 就能达到不错的效果。随着数据量的不断增加，慢慢构建起了身份属性、消费能力、用户关系、信用记录、出行记录等特征，GBDT+LR,RF，XGBOOOST，LightGBM 等更复杂的算法便可以派上用场了。当数据达到一定规模时，采用深度学习来进行自动化表征学习或则 end-to-end 的风控模型学习，我们测试过 DNN，这个算法在同等情况下，和 GBDT 之类的算法性能类似，并没有太明显的效果，一方面说明 GBDT 这类算法在处理非图像、语音、文本这类局部结构并不是特别明显的数据的优势，另一方面说明，不能直接将 CV\NLP 领域的算法拿来用，需要做一定的改造和优化，因此我们陆续使用和改造了 ResNet、FractalNet 等网络结构，相比于 DNN 和 GBDT 模型效果有比较明显的提升；再进一步，通过分析风控这个场景，我们发现，通过审批的用户与开放自然流量的数据分布差异比较明显，这一现象，基本打破了我们传统监督学习中训练集和预测集数据分布相似的假设。为解决这个问题，我们引入迁移学习框架，同等逾期条件下，通过率能明显的提升。

贷前信用风险模型v1.0

性能分析

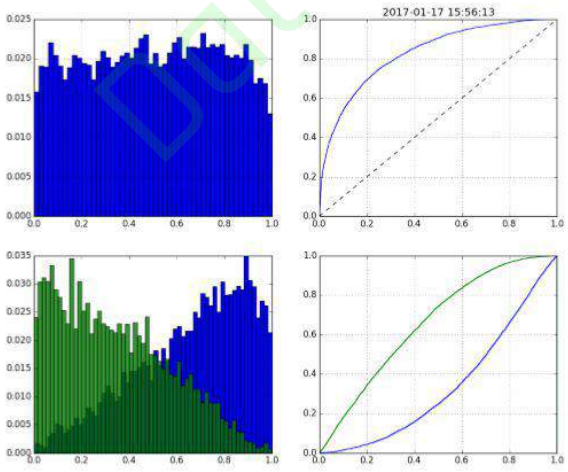


- 模型：LR
- 入参特征=40+
- KS = 31
- AUC = 72.4

Fig . 7 A 卡 V1-LR 模型

贷前信用风险模型v2.0

性能分析



- 模型：GDBT
- 入参特征=300+
- KS = 40
- AUC = 79

Fig . 8 A 卡 V2-GBDT 模型

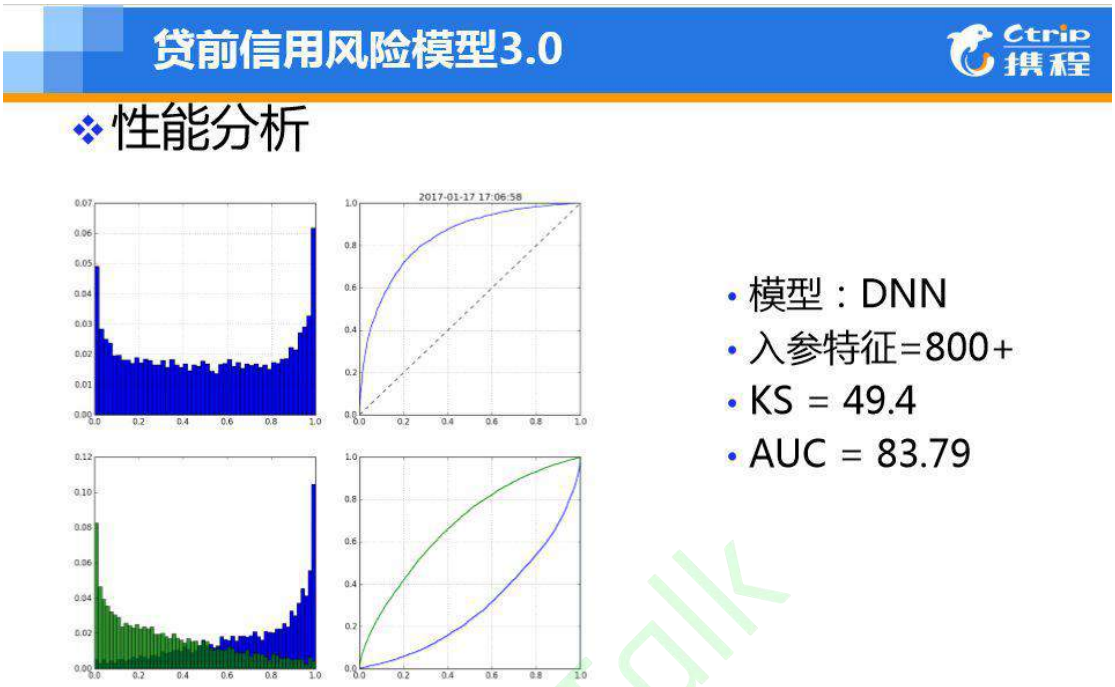


Fig . 9 A 卡 V3-DNN 模型

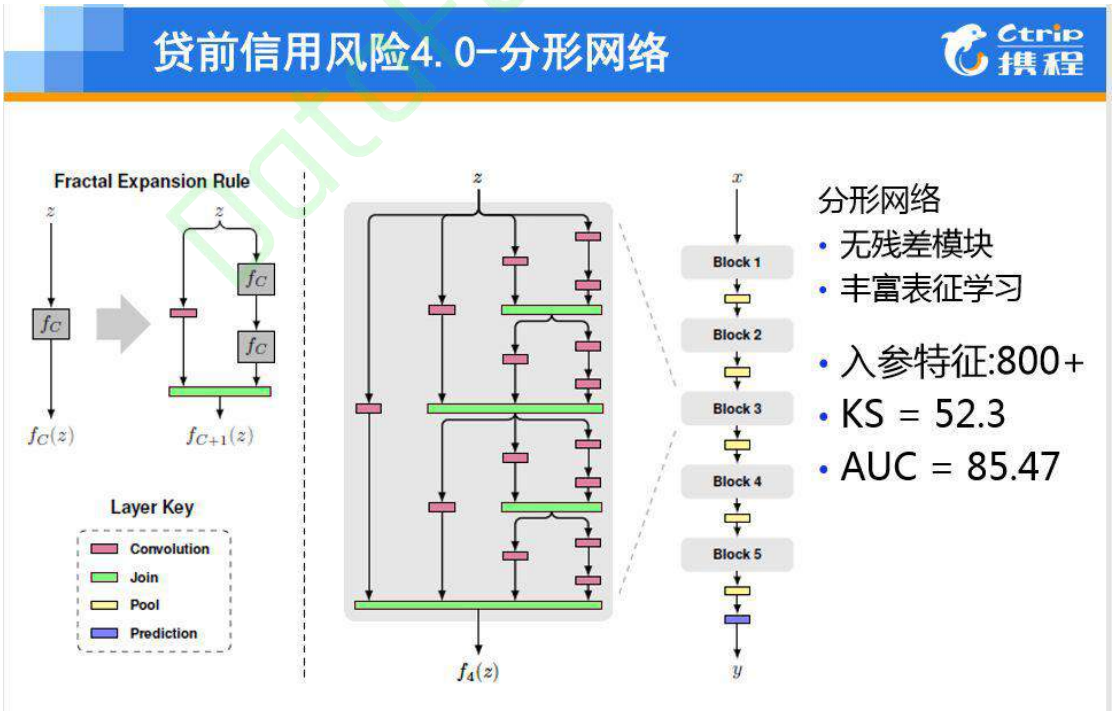


Fig . 10 A 卡 V4-分形网络模型

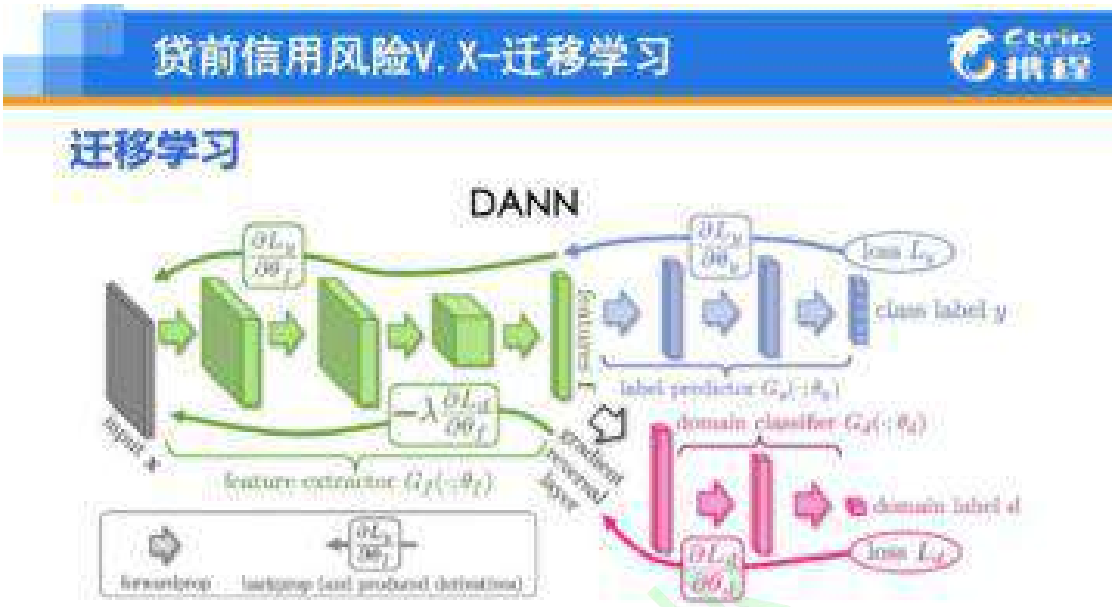


Fig . 11 A 卡 VX-迁移学习模型

「风控模型体系之贷中反欺诈模型」

贷中反欺诈按粒度可分为两类，用户级与交易级。用户级粒度相对粗一些，即断定当前客户为欺诈客户，可能的策略就是不允许欺诈用户在平台上发生交易行为；交易级是较细粒度的，即根据交易上下文、IP、设备、地域判断当前交易是否为欺诈交易，如果是，即不允许客户进行此笔交易。

贷中反欺诈有 3 个难点：

① 长尾分布：欺诈用户其实是极少的

② 对抗性显著：欺诈用户会想办法找出系统及规则的漏洞

③ 模仿正常行为：欺诈用户会利用伪造消费流水，前期正常还款等行为等，让金融公司放松警惕，当提额到一定程度后，便开始逾期。

在反欺诈领域，我们除了使用一般的机器学习模型，也构建了一套基于社交网络的模型体系。

「社交网络在风控模型中的应用」

基于社交网络的反欺诈，基本思想其实很简单，物以类聚，人以群分。比如一个欺诈分子，可能与其有关系（在 Graph 上表现为有直接的边连接，这种也称之为一阶亲密度；或者通过边的游走能够触达，这种称之为多阶亲密度），那么可能这些与之有关系的用户也是欺诈分子。如图所示，通过梳理携程生态内关键实体、关系，我们首先构建了一个庞大的异构社交网络，该网络包含 10 亿级别的顶点，50 亿级别的边。接下来是通过算法，发现社区(Community)。由于社交网络的数据量相对来讲是比较大的，因此在算法层面，对运算效率要求也是比较高的，同时对于社区划分的稳定性有一定要求。在实际落地中采用 LPA、改进的 Louvain，实现 T+1 的社区发现。最后基于划分的社区，可以获得社区的各种属性统计，这个作为反欺诈策略的重要参考。当有一个用户到来的时候，看其属于哪个社区，根据改社区的属性确定该用户是否为欺诈用户。

目前在携程金融的实际应用中，基于社交网络的风控指标已经覆盖了贷中 80%的贷款请求，同时通过社交网络，挖掘关系人一度或者多度关系，对严重的逾期行为，通过多度关系进行催收，提升回催率。

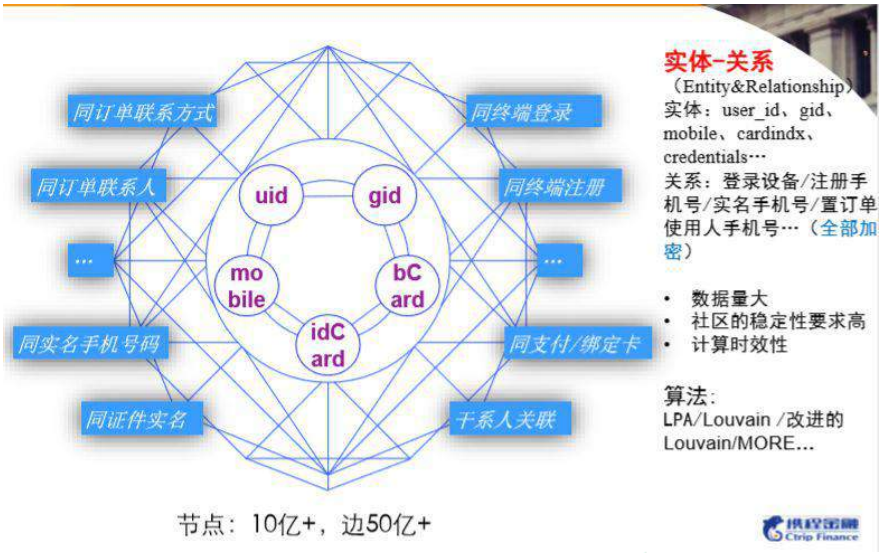


Fig . 12 携程金融关系网络

Fig . 13 社交网络应用的基本流程

金融风控反欺诈之图算法

嘉宾：张行军　编辑：HohXil

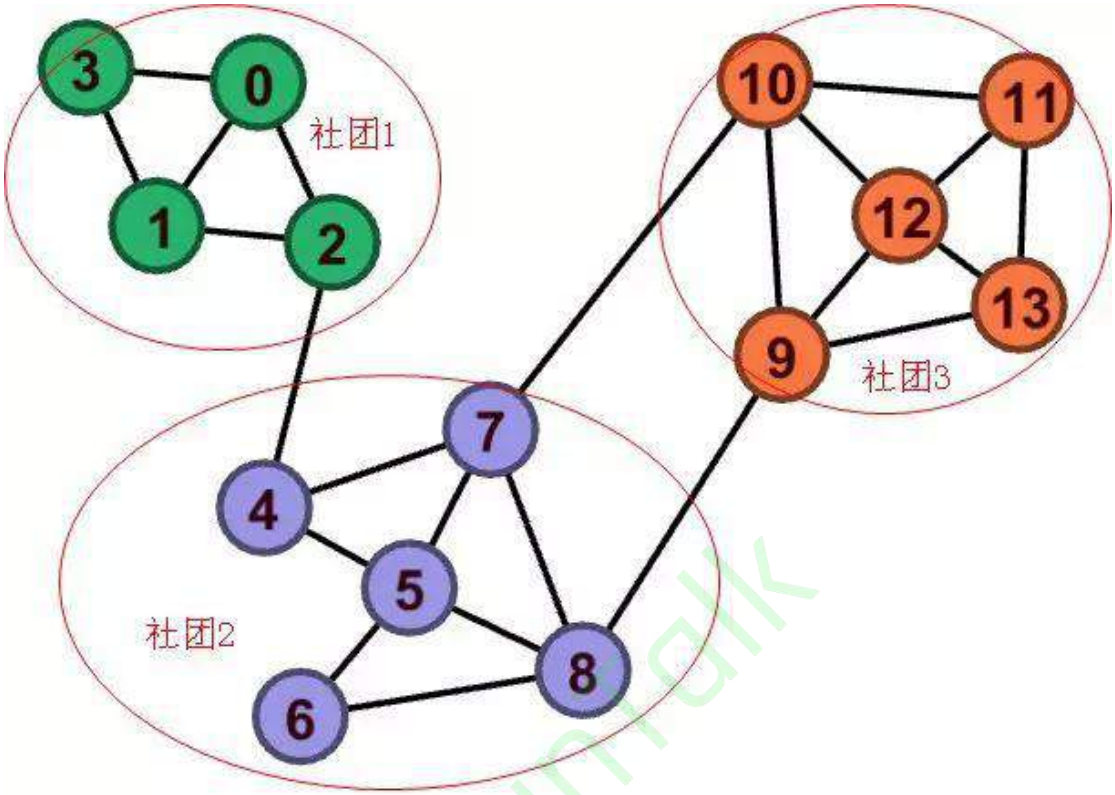
先介绍下金融借贷业务流程：用户前来申请借贷，会先经过欺诈识别，把欺诈团伙和主观欺诈的个人拒绝掉，然后对通过的人做信用评估，最后根据额度模型，算出利润最大化时放款金额。

刚才提到了团队欺诈，举个真实的例子。宜人贷在他们的财报中公布的，他们被一个团伙成功撸走了 2000 多单，当时宜人贷的件均 4w， 一下损失了 8000w！！

那么如何防范这种风险呢。这就是今天要分享的图算法。图可以将这些一个个有良好记录的个体关联起来,一网打尽。

再举一些团伙欺诈的行为。比如一个团伙，注册真实的淘宝商家，然后刷出良好的淘宝购物记录。或者来回转账，刷出良好的银行流水。

刚才前两位老师都没有提到额度模型，简单介绍下，如果只给用户放款 5000，可能坏账风险很小，但是利息也少，如果放款 10000，利息虽然收到利息多了，但是坏账风险高岭，所以需要做个权衡。



Graph 简介

G=(V,E)G=(V,E)

V:vertex set

E:edge set (有向，无向，有权重和没有权重)

> 举例，两个人之间的联系， A 给 B 买了东西，A 和 B 之间的通话次数
>
> 时长多于 A 和 C 之间。

- 度中心性（Degree Centrality） - 表示连接到某节点的边数。在有向图中，我们可以有 2 个度中心性度量：流入和流出。一个节点的节点度越大就意味着该节点在网络中就越重要。

- 接近中心性（Closeness Centrality） - 从某节点到所有其他节点的最短路径的平均长度。反映在网络中某一节点与其他节点之间的接近程度。

- 介中心性（Betweenness Centrality） - 某节点在多少对节点的最短路径上。介数中心性是比较能体现节点在图中桥梁作用的中心性度量方法。介数反映了相应的节点或者边在整个网络中的作用和影响力，具有很强的现实意义。例如，在交通网络中，介数较高的道路拥挤的概率很大；在电力网络中，介数较高的输电线路和节点容易发生危险。

社团发现算法一般有：

最小割， 正则化割:通过计算图的最小割，即将网络划分为预定的分组数，并使连接各分组的边的条数最少。

- 非负矩阵分解:基本原理是将原始矩阵分解得到社区指示矩阵和基矩阵

- 基于模块度的社区划分

- 基于节点相似性的社区划分

最小割算法广泛应用在分布式计算的负载均衡中，对集群节点的分组有利于减少不相关节点之间的通信。然而由于该算法限定了网络最终分组的个数，而不能通过算法“发现”节点间的内在联系并自然地构成若干个社区，因此最小割算法应用较为局限。

本文主要分享这两类的主要算法，基于模块度的 **louvain** 和基于信息熵 **infomap**，基于相似度的 **node2vec**

模块度(Modularity)公式及简化

优化目标：一般认为社团内部的点之间的连接相对稠密，而不同社团的点之间的连接相对稀疏。

所以模块度也可以理解是社区内部边的权重减去所有与社区节点相连的边的权重和，对无向图更好理解，即社区内部边的度数(内部的连线数)减去社区内节点的总度数。

$$
\begin{aligned}
Q &= \frac{1}{2m}\sum_{i,j}\left[A_{ij}-\frac{k_i k_j}{2m}\right]\delta(c_i,c_j) \\
&= \frac{1}{2m}\left[\sum_{i,j}A_{ij}-\frac{\sum_i k_i \sum_j k_j k_j}{2m}\right]\delta(c_i,c_j) \\
&= \frac{1}{2m}\sum_{c}\left[\sum_{in}-\frac{(\sum_{tot})^2}{2m}\right] \\
&= \sum_{c}\left[\frac{\sum_{in}}{2m}-\left(\frac{\sum_{tot}}{2m}\right)^2\right] \\
&= \sum_{a}\left[e_c-a_c^2\right]
\end{aligned}
$$

$$
\delta(u,v)=\begin{cases} 1 \text{when } u==v \\ 0 \text{ else} \end{cases}
$$

模块度公式的解释

A_{ij} 节点 i 和节点 j 之间边的权重，网络不是带权图时，所有边的权重可以看做是 1；

$k_i = \sum A_{ij}$ 表示所有与节点 i 相连的边的权重之和（度数）；

c_i 表示节点 i 所属的社区；

$m = \frac{1}{2} \sum_{i,j} A_{ij}$ 表示所有边的权重之和（边的数目）。

其中 Σin 表示社区 c 内的边的权重之和，Σtot 表示与社区 c 内的节点相连的边的权重之和，即社区 c 节点的度之和（包含与其他社区相连边的度）。

从概率的角度去看：

e_c 表示实际情况下，c 社区内产生边的概率。

a_c^2 表示在一种理想情况下，给定任意节点 i 的的度 ki，对节点 i 和节点 j 进行随机连边，边属于社区 c 的概率期望。

于是上式就表示了社区内连边数与随机期望的一个差值。连边数比随机期望值越高，表明社区划分的越好。

DataFunTalk

一般使用后面简化的公式，简化后的公式删除了判断两个节点是否划为同一个社区的函数，所以在一定程度上大大减少了 Q 值计算量。

Louvain

Louvain 算法的思想很简单：

将图中的每个节点看成一个独立的社区，此时社区的数目与节点个数相同；

对每个节点 i，依次尝试把节点 i 分配到其每个邻居节点所在的社区，计算分配前与分配后的模块度变化 ΔQ，并记录 ΔQ 最大的那个邻居节点，如果 $\max \Delta Q > 0$，则把节点 i 分配 ΔQ 最大的那个邻居节点所在的社区，否则保持不变；

重复 2，直到所有节点的所属社区不再变化；

对图进行压缩，将所有在同一个社区的节点压缩成一个新节点，社区内节点之间的边的权重转化为新节点的环的权重，社区间的边权重转化为新节点间的边权重，然后重复 2，3；

重复 2~4，直到整个图的模块度不再发生变化。

第一阶段称为 Modularity Optimization，主要是将每个节点划分到与其邻接的节点所在的社区中，以使得模块度的值不断变大；

第二阶段称为 Community Aggregation ,主要是将第一步划分出来的社区聚合成为一个点，即根据上一步生成的社区结构重新构造网络。重复以上的过程，直到网络中的结构不再改变为止。

移动

$$Q_1 = \left[\frac{E_c}{m} - \left(\frac{\sum tot}{2m}\right)^2\right] + \left[\frac{o}{m} - \left(\frac{k_i}{2m}\right)^2\right]$$

$$Q_2 = \frac{E_c + E_{i,c}}{m} - \left(\frac{\sum tot + k_i}{2m}\right)^2$$

$$\begin{aligned}\Delta Q &= Q_2 - Q_1 \\ &= \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \\ &= \left[\frac{k_{i,in}}{2m} - \frac{\sum_{tot} k_i}{2m^2}\right]\end{aligned}$$

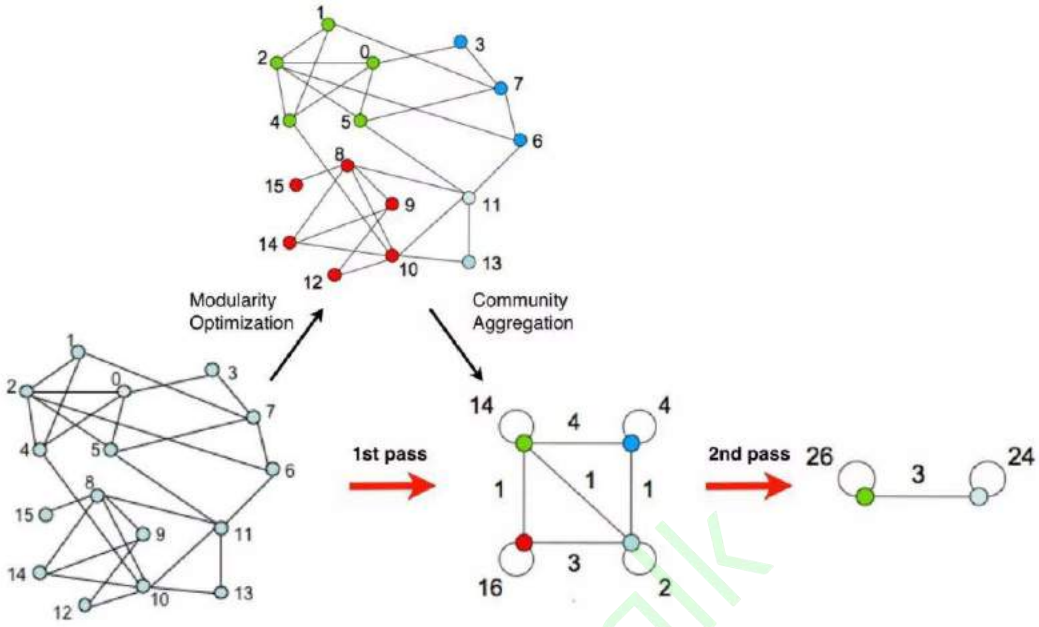
$k_{i,in}$ 是社区 c 内节点与节点 i 的边权重之和，再乘以 2

前面部分表示把节点 i 加入到社区 c 后的模块度，后一部分是节点 i 作为一个独立社区和社区 c 的模块度

1. 模块度与 Louvain 社区发现算法

http://www.cnblogs.com/fengfenggirl/p/louvain.html

2. Spark GraphX 分布式图计算实战



infomap

从信息论的角度出发，假设一个 random worker 在图上进行随机游走，那么怎么用最少的编码长度来表示其路径呢？

如果节点存在社区结构，那么社区内的节点就可以共享社区的 bit 位码，可以得到更小的平均比特，所以社区划分的越好，那么表示任意一条随机游走的路径所需的平均比特就越小。

如果我们能够计算出每个节点的到达概率，就可以依据信息熵的公式来量化平均比特了：

$$L(P) = H(P) \equiv -\sum_{\alpha} p_{\alpha} \log p_{\alpha}$$

怎么计算每个点的到达概率呢？

一个暴力的办法是在图上进行长时间的随机游走，最后统计每个节点的出现概率。太暴力了。

利用 pagerank 思路，初始化了每个节点的到达概率之后，就可以不断地迭代更新每个节点的到达概率，这个结果会很快趋于收敛。

其实这过程就是一个马尔科夫随机过程，随机初始化起始值，然后随机游走就相当于不停地用概率转移矩阵相乘，最后就可以达到马尔科夫稳态。

把随机游走事件归为三类：进入某个社团，离开某个社团，再社团内部游走。

定义清楚各类事件的发生概率，依据信息熵公式，就可以得到此时编码所需的平均比特了,其本质就是从信息论的角度出发。

$$L(\mathrm{M}) = q_{\curvearrowright} H(\mathcal{Q}) + \sum_{i=1}^{m} p_{\circlearrowright}^{i} H\left(\mathcal{P}^{i}\right)$$

$L(M)$ 将图划分成 m 个社区，编码随机游走路径时的平均比特；

$q_{\curvearrowright} = \sum_{i=1}^{m} q_{i\curvearrowright}$ 产生进入社区这种事件的总概率；

$H(Q) = -\sum_{i=1}^{m}(q_{i\curvearrowright}/q_{\curvearrowright})\log(q_{i\curvearrowright}/q_{\curvearrowright})$ 进入社区事件发生的信息熵；

$p_{i\circlearrowright} = \sum_{\alpha \in i} p_{\alpha} + q_{i\curvearrowright}$ random worker在社区 i 内部的概率，这里包括了在社区内跳转和离开该社区两类事件；

$H(\mathcal{P}^i) = -(q_{i\curvearrowright}/p_{i\circlearrowright})\log(q_{i\curvearrowright}/p_{i\circlearrowright})$
$-\sum_{\alpha \in i}(p_{\alpha}/p_{i\circlearrowright})\log(p_{\alpha}/p_{i\circlearrowright})$ random worker 下一步事件发生在社区 i 内部的信息熵；

Infomap 算法的迭代过程

1. 初始化，对每个节点都视作独立的社区；
2. while 平均比特的值不再下降；
3. 对图里的节点随机采样出一个序列，按顺序依次尝试将每个节点赋给邻居节点所在的社区，取平均比特下降最大时的社区赋给该节点，如果没有下降，该节点的社区不变。

参考链接

1. The map equation

http://www.mapequation.org/apps/MapDemo.html

2. https://mp.weixin.qq.com/s/qUxMesQA-edSyHeudQRRGA

3. DEEP GRAPH INFOMAX 阅读笔记

https://zhuanlan.zhihu.com/p/58682802

Graph embeddings

Deepwalk

1．使用随机游走(RandomWalk)的方式在图中进行节点采样获得节点共关系，

2．使用 skip-gram，根据步骤 1 中生成的节点序列学习每个节点的向量表示。skip-gram 就是根据给定输入的节点，预测上下文节点。

3．Deepwalk 有多不足，比如泛化能力，有新节点加入时，它必须重新训练模型以表示该节点。

其中一个就是采样，从其邻居中随机采样节点作为下一个访问节点，是一种可重复访问已访问节点的深度优先遍历算法。

node2vec 是一种综合考虑 DFS 邻域和 BFS 邻域的 graph embedding 方法

node2vec

优化目标：

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

条件独立假设：

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u))$$

特征空间的对称性：

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i)\cdot f(u))}{\sum_{v\in V}\exp(f(v)\cdot f(u))}$$

优化目标：

$$\max_f \sum_{u\in V} [-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i)\cdot f(u)]$$

$Z_u = \sum_{u\in V}\exp(f(u)\cdot f(v))$ 计算量非常大，所以论文采用负采样(negative sample)进行近似计算。

这个 node2vec 优化目标函数，因为它跟大名鼎鼎的 word2vec 是一样。

我们最初是用一个 Python 写的包，跑一遍算法需要一周。后来想，既然优化目标是一样的，那能不能用 word2vec 包，因为 word2vec 用 c 写的，而且还采用了 Hierarchical Softmax，negative sampling 加速。

然后在网上找到了一个套用 word2vec 实现的 node2vec 包，速度快很多。

随机游走的方式

复杂网络处理的任务其实离不开两种特性，前面也提到过：一种是同质性，就是之前所说的社区。一种就是结构相似性，值得注意的是，结构相似的两个点未必相连，可以是相距很远的两个节点。

能不能改进 DeepWalk 中随机游走的方式，使它综合 DFS 和 BFS 的特性呢？所以本文引入了两个参数用来控制随机游走产生的方式。

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\alpha_{pq}(t,x) \cdot w_{vx}}{Z} & \text{if(v,x)} \in E \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Z 是分子的归一化常数

如果已经采样了（t,v），也就是说现在停留在节点 v 上，那么下一个要采样的节点 x 是哪个？作者定义了一个概率分布，也就是一个节点到它的不同邻居的转移概率：

直观的解释一下这个分布：

- 如果 t 与 x 相等，那么采样 x 的概率为 $\frac{1}{p}$；
- 如果 t 与 x 相连，那么采样 x 的概率 1；
- 如果 t 与 x 不相连，那么采样 x 概率为 $\frac{1}{q}$

参数 p、q 的意义分别如下：

返回概率 p：

- 如果 p>max(q,1) ，那么采样会尽量不往回走，对应上图的情况，就是下一个节点不太可能是上一个访问的节点 t。
- 如果 p<min(q,1) ，那么采样会更倾向于返回上一个节点，这样就会一直在起始点周围某些节点来回转来转去。

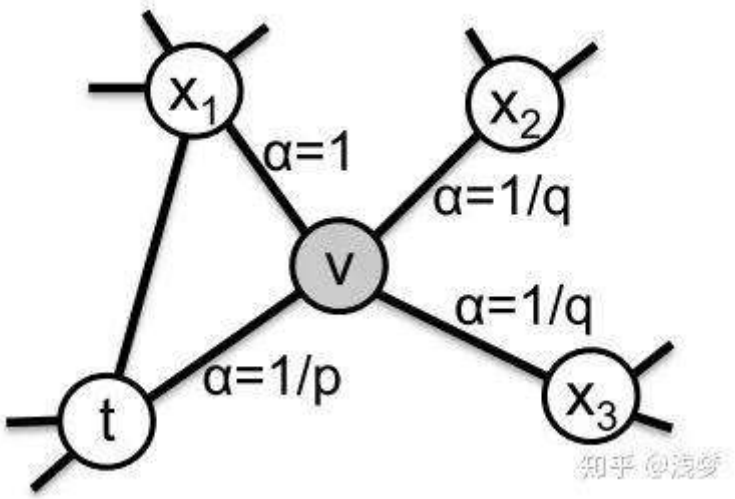
出入参数 q：

- 如果 q>1 ，那么游走会倾向于在起始点周围的节点之间跑，可以反映出一个节点的 BFS 特性。
- 如果 q<1 ，那么游走会倾向于往远处跑，反映出 DFS 特性。
- 当 p=1，q=1 时，游走方式就等同于 DeepWalk 中的随机游走。

简而言之：

参数 p 控制重复访问刚刚访问过的顶点的概率，

参数 q 控制着游走是向外还是向内，若 q>1，随机游走倾向于访问和 t 接近的顶点(偏向 BFS)。若 q<1，倾向于访问远离 t 的顶点(偏向 DFS)。



缺点

- 慢
- 先 embedding 再聚类，感觉这两个过程很割裂！！融合一下

comE

Graphembedding 得到向量后，可以做很多事情，在我们这个主题可以简单的通过聚类来讲节点分组。

但是这个过程比较割裂，先优化 node2vec，然后再优化聚类。能不能整体上一次性优化完呢。

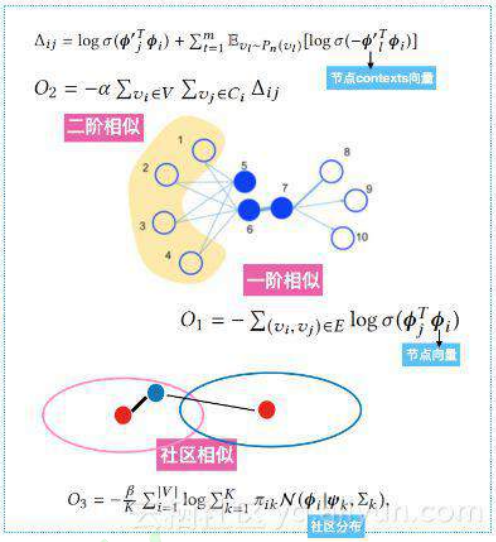
comE 这个算法优化目标中加入了社区的检测和嵌入。通过一个混合高斯模型将节点划分开。

一阶LOSS 二阶LOSS 社区LOSS

$$\mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) = O_1(\Phi) + O_2(\Phi, \Phi') + O_3(\Phi, \Pi, \Psi, \Sigma)$$

Our final optimization problem becomes:

$$(\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*) \leftarrow \underset{\forall k, diag(\Sigma_k)>0}{\arg\min} \mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma)$$



<1>.社区检测和社区嵌入。

这篇论文比较创新的一点，就是将社区嵌入定义为低维空间中的多元高斯分布。即对于社区k来说($k \in \{1, \ldots, K\}$)，它在d维空间下的社区嵌入是一个多元高斯分布 $\mathcal{N}(\psi_k, \Sigma_k)$ ，其中 $\psi_k \in R^d$ 表示的是社区内节点的平均向量（即中心点）， $\Sigma_k \in R^{d*d}$ 表示的是协方差矩阵。

整篇文章最大的难点在于，如何用统一的目标函数来使得社区嵌入、社区检测和节点嵌入三者问题得到最优化的结果。由于社区嵌入是由多元高斯分布来表达的，因此就可以通过高斯混合模型来完成。

假设每一个节点的嵌入向量 ϕ_i 可以由该节点所在社区的多元高斯分布产生，这样一来，就可以得到一个似然目标函数：

$$\prod_{i=1}^{|V|} \sum_{k=1}^{K} p(z_i = k) p(v_i | z_i = k; \phi_i, \psi_k, \Sigma_k) \quad (1)$$

其中， $p(z_i = k)$ 表示的是节点 v_i 属于社区k的概率。方便表示，可以将这个概率表达为 π_{ik} ： $\pi_{ik} \in [0, 1], \sum_{k=1}^{K} \pi_{ik} = 1$ 。在社区发现问题中，这个概率是未知的。

而 $p(v_i | z_i = k; \phi_i, \psi_k, \Sigma_k) = \mathcal{N}(\phi_i | \psi_k, \Sigma_k)$ 表示的是多元高斯分布。在社区嵌入问题中， ψ_k, Σ_k 是未知的。

综上，将公式1对上述三个未知变量求导，就可以同时实现社区检测和嵌入。

优化目标中前面两项跟 LINE 定义的相似度相似：

1st

对于每一条无向边(i,j)，定义顶点v_i和v_j之间的联合概率为：

$p_1(v_i,v_j)=\frac{1}{1+\exp(-\vec{u}_i^T\cdot\vec{u}_j)}$，$\vec{u}_i$为顶点$v_i$的低维向量表示。（可以看作一个内积模型，计算两个item之间的匹配程度）

同时定义经验分布$\hat{p}_1=\frac{w_{ij}}{W}, W=\sum_{(i,j)\in E}w_{ij}$

优化目标为最小化：$O_1=d(\hat{p}_1(\cdot,\cdot),p_1(\cdot,\cdot))$

$d(\cdot,\cdot)$是两个分布的距离，常用的衡量两个概率分布差异的指标为KL散度，使用KL散度并忽略常数项后有

$O_1=-\sum_{(i,j)\in E}w_{ij}\log p_1(v_i,v_j)$

1st order 相似度只能用于无向图当中。

2nd

这里对于每个顶点维护两个embedding向量，一个是该顶点本身的表示向量，一个是该点作为其他顶点的上下文顶点时的表示向量。

对于有向边(i,j)，定义给定顶点v_i条件下，产生上下文(邻居)顶点v_j的概率为

$p_2(v_j|v_i)=\frac{\exp(\vec{c}_j^T\cdot\vec{u}_i)}{\sum_{k=1}^{|V|}\exp(\vec{c}_k^T\cdot\vec{u}_i)}$，其中$|V|$为上下文顶点的个数。

优化目标为

$O_2=\sum_{i\in V}\lambda_i d(\hat{p}_2(\cdot|v_i),p_2(\cdot|v_i))$，其中$\lambda_i$为控制节点重要性的因子，可以通过顶点的度数或者PageRank等方法估计得到。

经验分布定义为：$\hat{p}_2(v_j|v_i)=\frac{w_{ij}}{d_i}$，$w_{ij}$是边$(i,j)$的边权，$d_i$是顶点$v_i$的出度，对于带权图，$d_i=\sum_{k\in N(i)}W_{ik}$

使用KL散度并设$\lambda_i=d_i$，忽略常数项，有

$O_2=-\sum_{(i,j)\in E}w_{ij}\log p_2(v_j|v_i)$

1. https://blog.csdn.net/u012151283/article/details/87013915

2. Learning Community Embedding with Community Detection and Node Embedding on Graphs

https://zhuanlan.zhihu.com/p/36924789

3. Learning Community Embedding with Community Detection and Node Embedding on Graphs

https://github.com/vwz/ComE

4. http://sentic.net/community-embedding.pdf

评价指标

Modularity

标准化互信息 NMI（Normalized Mutual Information）

$$\mathrm{NMI}(U,V)=\frac{\mathrm{MI}(U,V)}{\sqrt{H(U)H(V)}}$$

$$\begin{aligned}\mathrm{MI}(U,V)&=\sum_{i=1}^{|U|}\sum_{j=1}^{|V|}P(i,j)\log\left(\frac{P(i,j)}{P(i)P'(j)}\right)\\&=\sum_{i=1}^{|U|}\sum_{j=1}^{|V|}\frac{|U_i\cap V_j|}{N}\log\left(\frac{N|U_i\cap V_j|}{|U_i||V_j|}\right)\end{aligned}$$

假设对于 N 个样本点的两种标签划分为 U 和 V. 熵为划分集的不准确性

$$H(U)=-\sum_{i=1}^{|U|}P(i)\log(P(i))$$

$$H(V)=-\sum_{j=1}^{|V|}P'(j)\log(P'(j))$$

$P(i)=|U_i|/N$ 表示任取一个样本划分为U_i的概率

$$P'(j)=|V_j|/N$$

实时社区发现

1. 基于个体稳定度博弈的动态社区发现算法研究

http://www.deliwenku.com/p-84779.html

2. 基于标签传播的实时社区发现算法研究

http://www.doc88.com/p-1055432963471.html

3. 基于 spark streaming 的动态社区发现及其在个性化推荐应用中的研究

http://www.doc88.com/p-6991331093268.html

4. 一种基于局部模块度的增量式动态社区发现算法

http://www.doc88.com/p-9146960189736.html

5. 基于标签传播的实时社区发现算法研究

http://www.doc88.com/p-1055432963471.html

6. 图流合璧——基于 Spark Streaming 和 GraphX 的动态图计算

https://wenku.baidu.com/view/fa65268159eef8c75fbfb3a6.html?re=view

参考佳文

1. 深度学习时代的图模型，清华发文综述图网络

2. 社区发现(Community Detection)算法

https://blog.csdn.net/cleverlzc/article/details/39494957

3. 社区发现算法 FastUnfolding 的 GraphX 实现

http://www.aboutyun.com/forum.php?mod=viewthread&tid=19817

4. 谱聚类（spectral clustering）原理总结

https://www.cnblogs.com/pinard/p/6221564.html

5. GraphWave：一种全新的无监督网络嵌入方法

6. 社团划分结果评估指标：Q、ARI、NMI

https://blog.csdn.net/DreamHome_S/article/details/78167267

7. 论文解读：从乌合之众到群体智慧的一步之遥

8. 2017CIKM - Network Embedding 专题论文分享

https://yq.aliyun.com/articles/294450

9. node2vec: Scalable Feature Learning for Networks

https://www.jianshu.com/p/a9a2ed8b98be

10. node2vec: Scalable Feature Learning for Networks

https://arxiv.org/pdf/1607.00653.pdf

11. node2vec: Scalable Feature Learning for Networks 阅读笔记

https://zhuanlan.zhihu.com/p/30599602

12. 基于 PageRank 的复杂网络社区发现

https://blog.csdn.net/qq_32284189/article/details/80827899

13. Modularity 的计算方法——社团检测中模块度计算公式详解

http://www.yalewoo.com/2017/03/10/modularity_community_detection/

14. 社区发现算法 - Fast Unfolding（Louvian）算法初探

https://www.cnblogs.com/LittleHann/p/9078909.html

15. 论文 | 蚂蚁金服亮相数据挖掘顶会 KDD 2018，这些你不可错过！

DataFunTalk

风险价值量化术—互金爆发期的风控技术实践

嘉宾：苏晓林　编辑：HohXil

本次分享分为两部分：业务和技术。

把业务和技术分称为务虚和务实并没有褒贬之义。只是从技术的视角来说，在技术实现之前，我们需要把业务弄明白，否则容易做错方向，浪费了团队的时间和精力。

一. 业务部分

1. 哪些是风控要解决的问题

首先是业务方面，这些年来我们的风控同学都在解决什么问题。

（1）身份认证

传统方式：

- 身份证
- 现场拍照

现在大部分互金公司的做法：

- 账号体系

（同业接入→其他业务共享→市场解决方案）

- 人脸识别

- 活体验证

我们需要解决的问题：

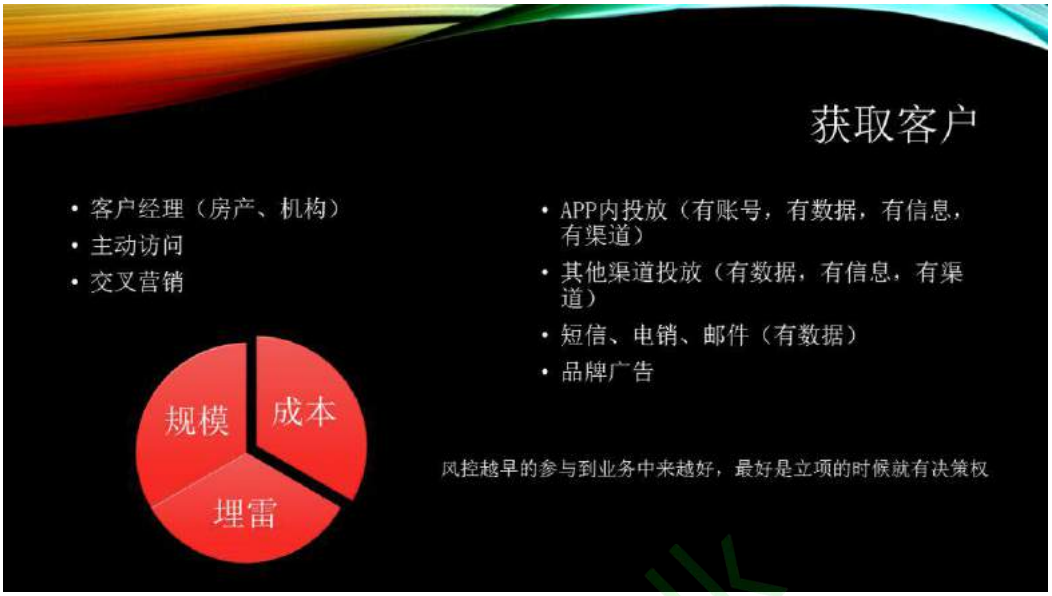
- 虚假资料
- 第三方欺诈
- 本人欺诈
- 账户盗用

对应金融机构的主要业务有：

- 理财
- 转账
- 支付
- 消费
- 信贷

本文聚焦在信贷这个业务下，风险价值量化的作用和方法。

（2）获取客户



我们常见的金融业务获客方式如上，左边为传统获客，右边是互金领域的获客方式。为什么我们先提到获客？因为在获客过程中，金融业务风险因素就已经在其中起到重要的影响作用。

我们认为风控越早参与到业务中越好，最好是立项的时候就有决策权，这里考虑的因素如下：

• 有效的转化和客单价一样的重要；转化率太高太低都不行；因为风控成本要计算到获客成本中来；（我们在日常的团队和业务管理上经常有一个误区，就是把 KPI 拆的太散太细，以至于不同团队的目标是冲突的）；

• 可识别的客群规模决定了该方案是否可以规模化；

• 不稳定的客群锻炼不出万能的策略，高手都是提前一招布局；我们需要用差异化的策略服务差异化的用户；（物以类聚，你肯定不愿意跟资质比你低的人平摊各项成本）

• 充分竞争的市场里，整合资金、客户、产品、风控的能力就会产生价值。

（3）产品的定位（金额、期限、利率、循环、增信）

风险量化能力在如下几个方面，影响着产品的定位：

• 大额产品看客户资质、重视准入规则，小额产品则研究概率、重视用户信用评级；

• 长期产品要找准个人还款能力的门槛，短期产品则着重把握个人的还款意愿（及容忍尺度的把控）；

• 银行胜在资金成本，消金赢在风控能力，小贷活在流量吸取（金主的不同在于各自优势的差异）；

• 贷后风控能力决定了资产循环的质量；

• 增信的标的资产能否做兜底，是解决劣后风险的关键；

（4）客群的定位

• 用户画像：是谁，在哪儿，什么时候，用哪些产品（被讲烂的 5W）；合适的时候出现在合适的场景（瑞幸的线下广告投放就是一个成功的案例）

2. 风控的武器有哪些？

各位在金融机构的同事都知道，公司内部我们常用的名称是风险管理部，而非风险控制部，我们的目标不是完全杜绝风险，而是把它控制在我们 ROI 最大的区间。之所以叫风险管理就是让大家用 ROI 的思想来解决问题：

Investment 投入：公司内部有什么资源，外面能买到什么服务，客户能够提供给你什么材料，我们能制造点什么工具。它们都是数据，在风控解决信息不对称的过程中，这些数据就是我们所需信息的载体。

（1）内部数据整理

首先是内部数据的有效利用，图中为银行和互联网公司的常用特征，其中：

银行：防作弊、稳定性、质量、相关性

互联网：丰富度、数量、覆盖率

其他部门的特征数据：例如社交、广告、LBS 等部门的已有特征，其中有类似于社交网络活跃度、广告客群的用户标签；电商产品的积分值；BBS 站内用户等级之类现成的特征；

公司级的大数据共享：大公司的数据中心都有个人兴趣、偏好、习惯、脱敏属性的用户画像；

最后一部分则是通过金融部门自己的专家团队来挖掘金融场景相关的特征，如收入、支出、资产、负债的离散分级标签，BATJ、TMD 等企业都有足够的技术能力和资源完成这项工作。

（2）特征挖掘

在上节例举的各类数据中，我们的工作主要集中在自研金融特征上。流程如下：

首先是专家 Pattern 扩展：比如 BAT 都有自己的 LBS 信息，我们可以基于家庭住址做一个特征，假如你住在一个均价 10W 的小区，无论自有还是租赁，你的收入能力应该都不错。

其次是特征工程：主要指怎么样用不同的特征来做加工组合，然后跟抽象概念能够关联起来。

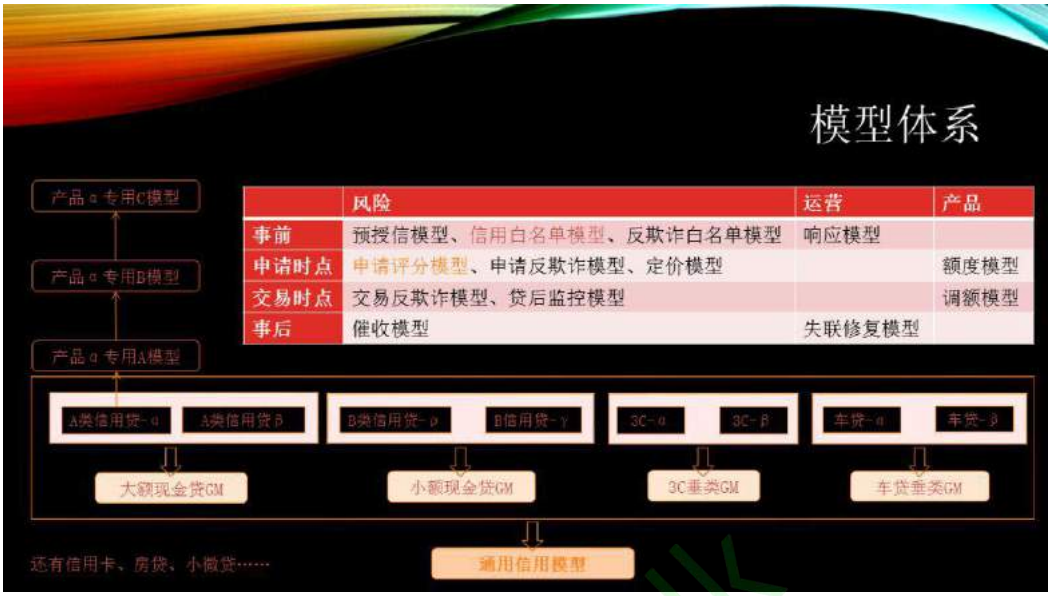
最后是埋点衍生：当我所有数据都没办法验证这件事情的时候，可以在 APP 中加一些合规的工具，来获取一些有用的埋点序列数据。

（3）外部接口接入及数据合作：

（4）数据的进一步扩充：

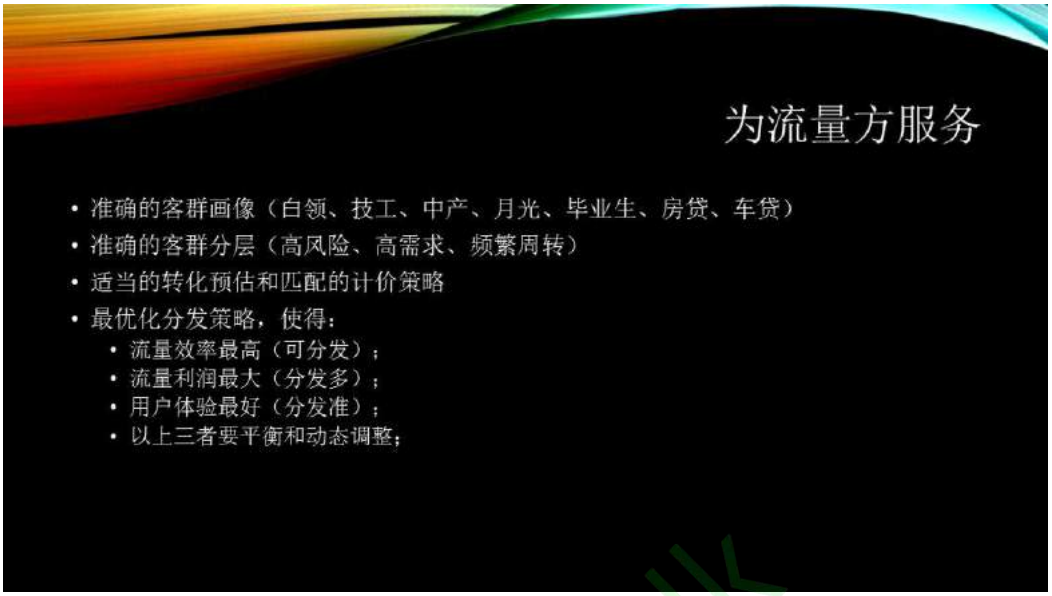
3. 找钉子就是做生态

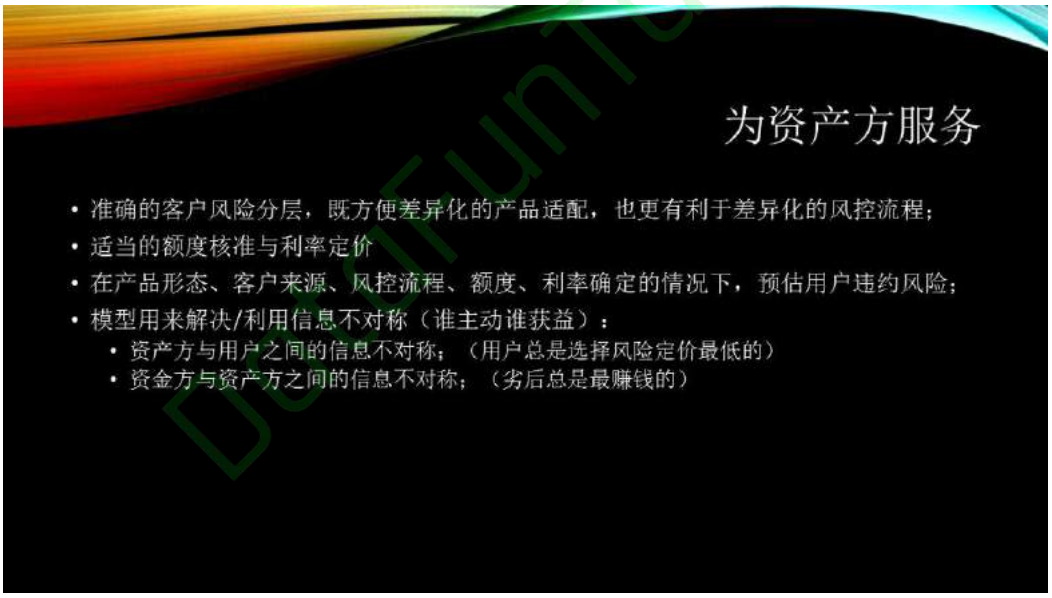
最开始互金领域是没有模型体系的，银行里有 A 卡、B 卡和 C 卡，我们在互金行业成长的过程中慢慢的把适合它的模型体系建立起来，结构上我们可以分成几个事件时点：1.事前、2.申请、3.交易、4.事后。然后我们针对各产品来做 A、B、C 卡。



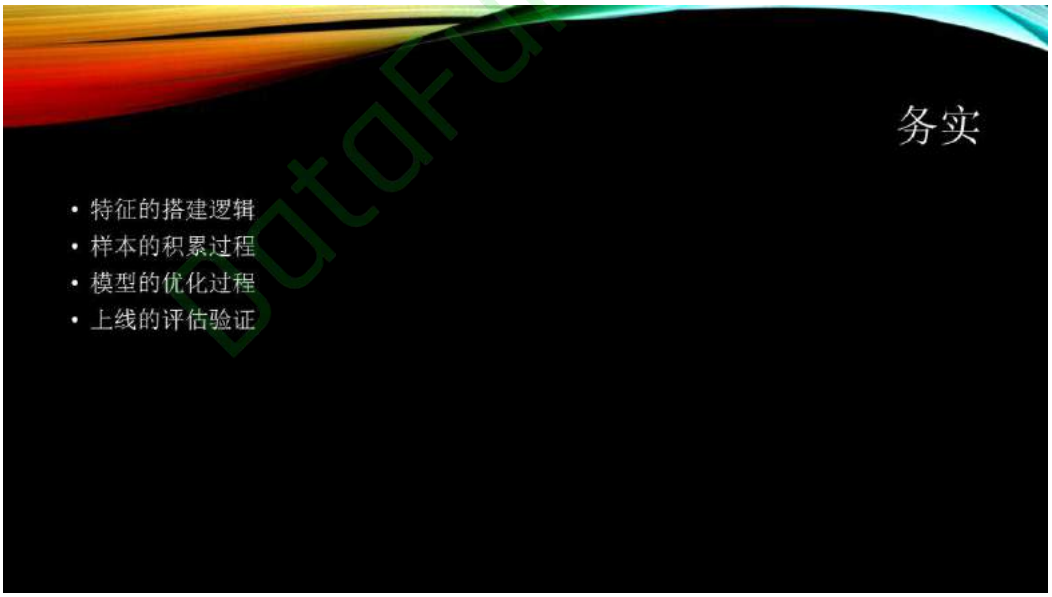
4. 我在为谁服务

读者中有很多朋友是做风控业务的，那么一个很重要的问题就是，你需要知道你是在为谁服务，也就是你的视角 POV。


为流量方服务
• 准确的客群画像（白领、技工、中产、月光、毕业生、房贷、车贷）
• 准确的客群分层（高风险、高需求、频繁周转）
• 适当的转化预估和匹配的计价策略
• 最优化分发策略，使得：
• 流量效率最高（可分发）；
• 流量利润最大（分发多）；
• 用户体验最好（分发准）；
• 以上三者要平衡和动态调整；


为资产方服务
• 准确的客户风险分层，既方便差异化的产品适配，也更有利于差异化的风控流程；
• 适当的额度核准与利率定价
• 在产品形态、客户来源、风控流程、额度、利率确定的情况下，预估用户违约风险；
• 模型用来解决/利用信息不对称（谁主动谁获益）：
• 资产方与用户之间的信息不对称；（用户总是选择风险定价最低的）
• 资金方与资产方之间的信息不对称；（劣后总是最赚钱的）

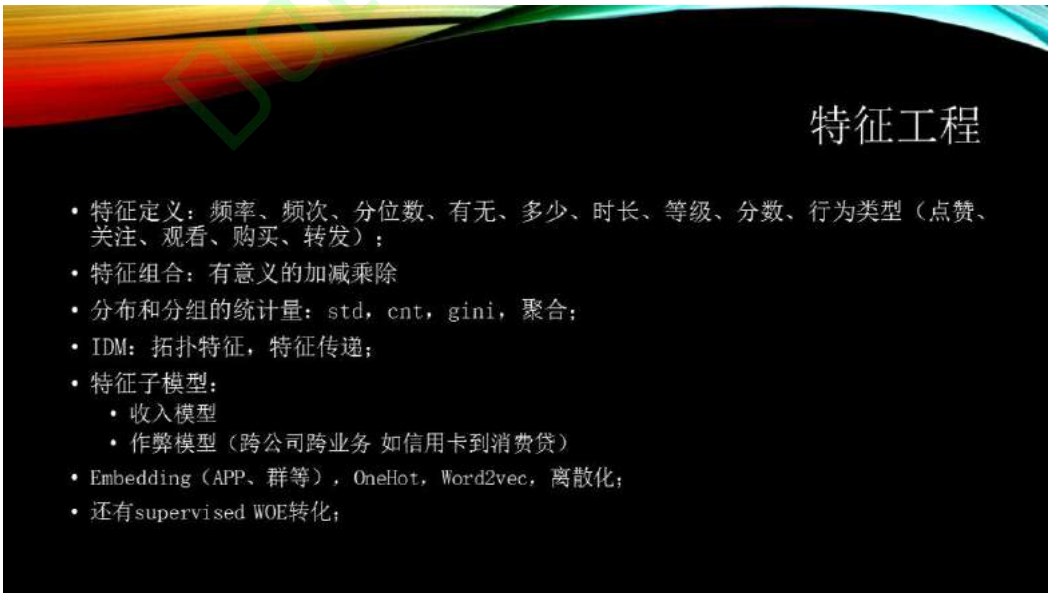
二．技术部分



介绍了业务背景和我们面对的问题之后，我们才来解释这些问题是怎样解决的。原因在于我们要先想得清楚，然后才能做得明白；业务和技术的关系是方向和数量的关系，方向错了，技术越好，那么结果可能越糟。

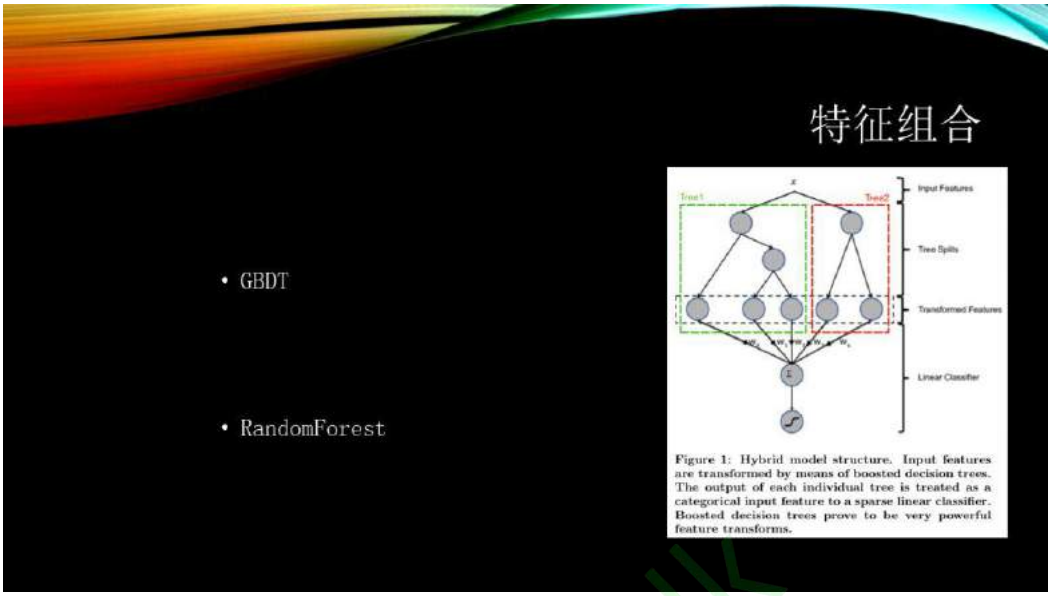
下面介绍的建模过程不只限于风控，在获客，导流，分发，催收，资产打包等环节都可以使用。

1. 特征的搭建逻辑



以上大部分的概念相信大家都不陌生，这里简单说明一下 Embedding 的作用，它是我们用来从高维稀疏的空间里，加工出一系列可解释的、低维度的特征集合。

左边为初版模型搭建时需要考虑的特征评估指标，右边是新的产品上线或者老产品模型迭代的时候参考的特征评估方法。



在特征组合方面，我们主要可以采用俩种方式来将非线性特征转化为线性特征：GBDT 和 Random Forest，他们都会生成右边绿色和红色方框中的决策树，这时每一个叶子节点，就已经代表用户画像的某些方面。细节请参考：

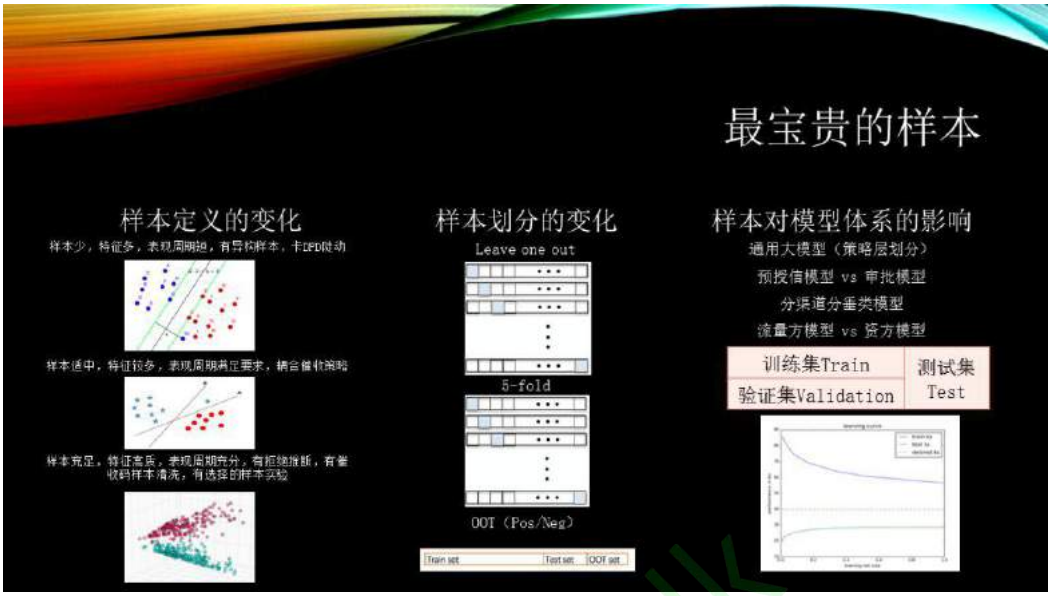
https://dl.acm.org/citation.cfm?id=2648589

2. 样本的积累过程

样本的积累过程包括：

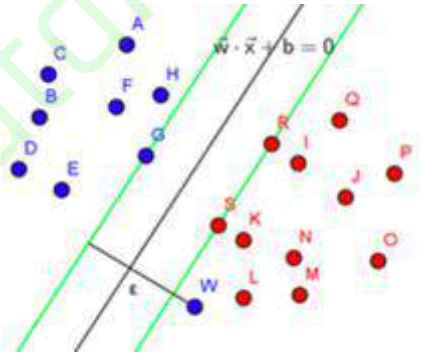
- 样本替代
- 样本划分
- 样本清洗

DataFunTalk



首先是我们在不同时期建模过程中样本定义的变化：

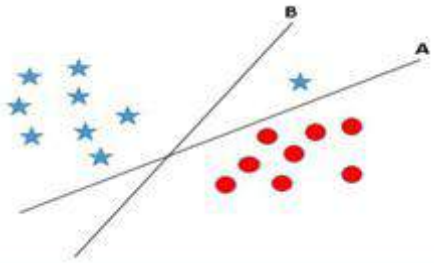
• 第一阶段：



特点：样本少，特征多，表现周期短，有异构样本。

方法：按照 DPD 的走势，选择曲线最陡的位置作为正负样本的划分点。

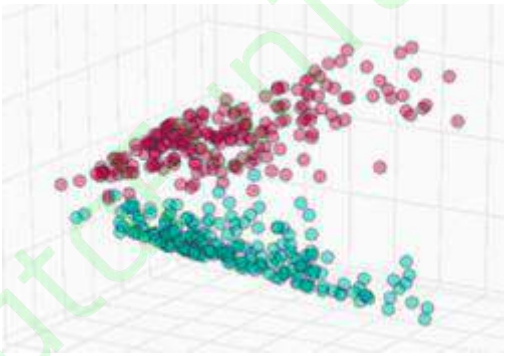
• 第二阶段：



特点：样本适中，特征较多，表现周期满足要求。

方法：耦合催收策略做样本清洗。

• 第三阶段：



特点：样本充足，特征高质，表现周期充分。

方法：加入拒绝推断，使用催收结果码做样本清洗，有选择的做样本实验。

样本划分

时间划分（授信日 vs 账单日）vs 随机抽取 vs 其他规则，避免样本穿越（尤其是子模型）。

样本划分的变化：

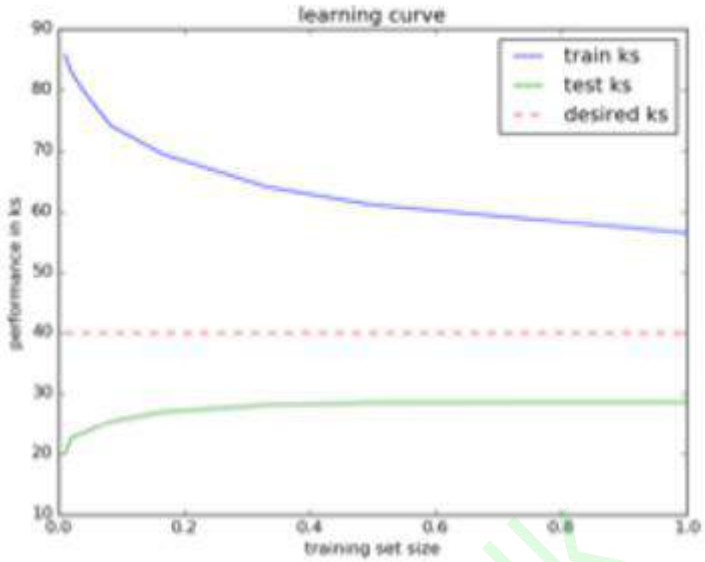
- 第一阶段，Leave one hot；
- 第二阶段，5-fold；
- 第三阶段，OOT（out of time）（Pos/Neg）。

样本对模型体系的影响：

- 通用大模型（策略层划分）
- 预授信模型 vs 审批模型
- 分渠道分垂类模型
- 流量方模型 vs 资方模型

最后是验证集和测试集的定义，如下图：

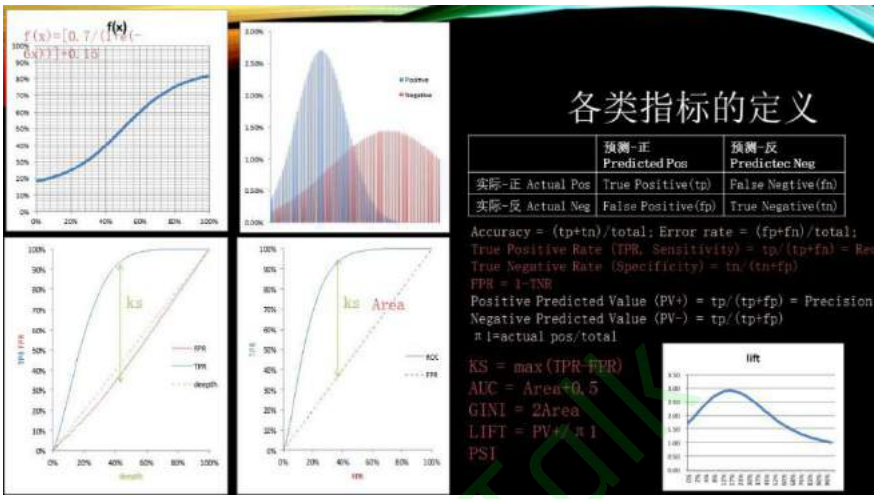
DataFunTalk



横坐标为训练集中正样本占比（保持负样本数量不变）或负样本占比（保持正样本数量不变），看测试集的指标变动趋势，评估训练集中样本数量是否充足。

3. 模型的优化过程

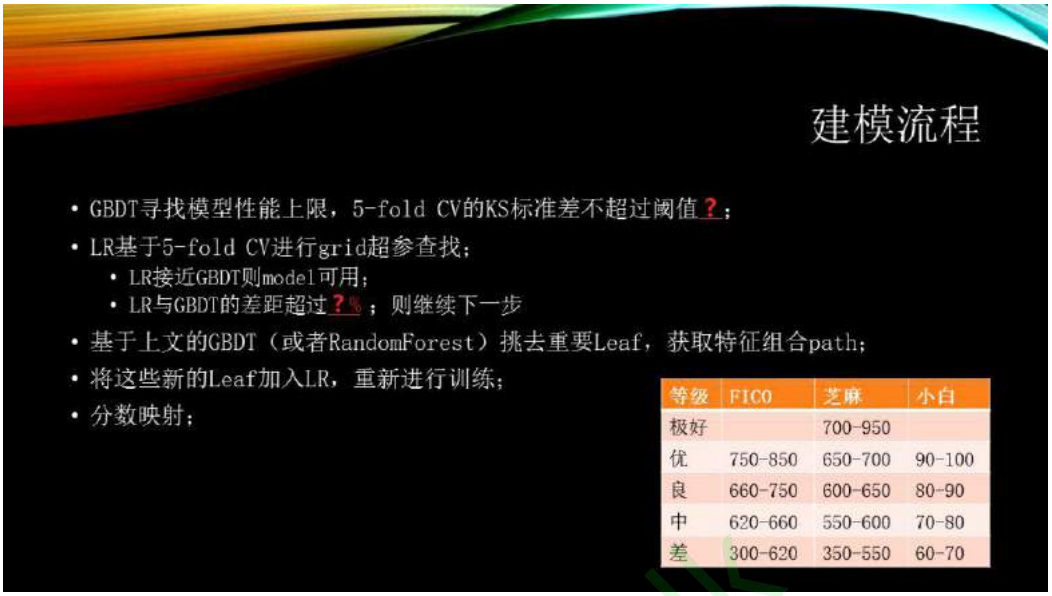
准备好了特征和样本，我们已经有了建立模型的基础。在金融领域我们面临的往往是最复杂的业务，但实际上我建议大家用最简单的模型。



AUC 本身描述了模型整体的表现效果，一般情况下 AUC 越大，那么它对风控策略的提升效果也越好。

KS：是否越大越好？不是的，实际上一个模型对实际业务的贡献取决于风控策略与风控模型的配合，模型对 cutoff 的选择和策略对 cutoff 的选择如果差异较大的话，KS 就没有太大意义了。

GINI 与 AUC 的关系如上图所示，两者用其中之一即可。



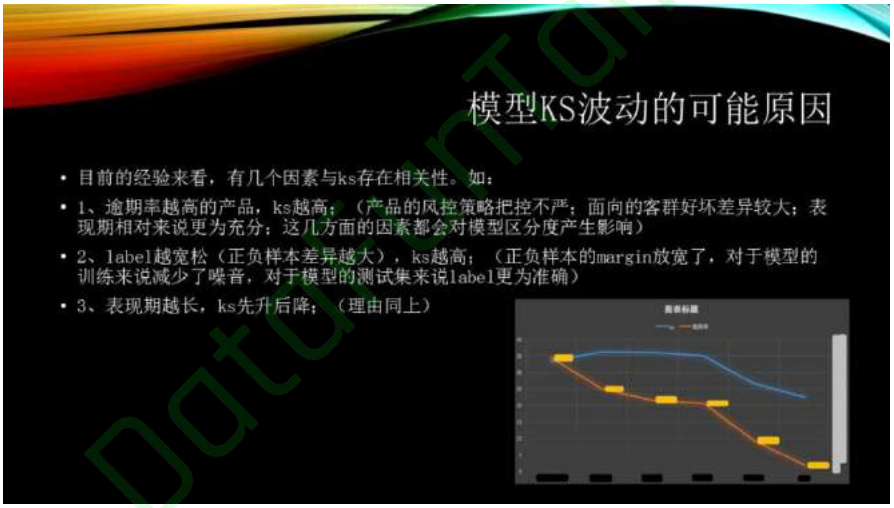
这里的建模流程比较简单，相关算法请参考其他文章，右边是分数映射的一个案例。每个公司都有自己的一套标准，或者网上有一些公开的数据，这里不细细介绍了。（此处为 2017 年的网络查得信息，不代表当前的真实情况）

4. 上线验证评估

最后是上线的评估验证，包括算法指标验证和业务指标验证。

首先要看的是模型的解释性强不强，主要看如下几点：

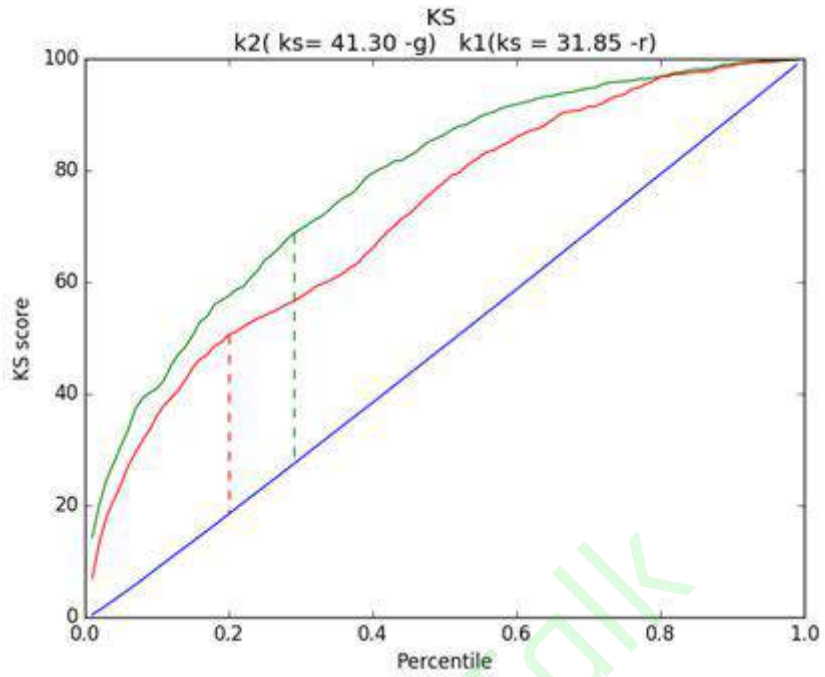
- 特征权重：如 LR 里的 coefficient，GBDT 里的 Weight、Cover、Gain；
- 样本集打分对照：如优质客群和次级客群的对照；
- 业务逻辑合理性：这块前面有过介绍。



模型评审：

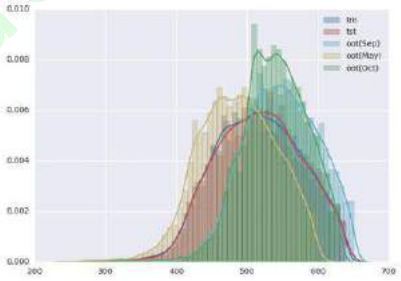
首先是技术同学的算法指标验证：

- KS 值在 Train、Test、OOT 上的提升与波动！
- KS 值@位置的偏移和稳定性！



- PSI 的波动！

- swap-in/out



然后是风控业务同学的指标验证：

- 同通过率情况下违约率的下降；

- 同违约率情况下通过率的提升；

- 通过客群的额度波动；

- 通过客群的用信率波动；

- 通过客群的 GMV 波动。

这里列举了一些上线事故：

- 特征穿越：时间、流程、label，这些情况都是大家应该尽量去避免的。

- 特征扰动：比如社交好友数越来越多、个人信用分越来越高，模型不能经常更新特征参数，此时该做的是归一化，窗口化，正则化。归一化就是我的社交好友数在全国可以 rank 到第几万位，窗口化就是我们不看累积的好友，而只看样本最近一个月新增的好友，正则化就是我把一些特征做一些正则变换。

- 特征中断：Backup 两个模型版本，随时可以切换，不会影响到业务运营。

• 大规模实时计算：我们不可能每次都实时计算或者查询某个信息，可行的解决方法是定期的离线批量更新（有时会涉及到 model 重训）。

最后上线的时候建议大家一定要经过这样三个环节：

• 离线校验

• 线上校验，上线的时候线上和线下对比；

• 小流量校验，然后再切换。

DataFunTalk

音频内容理解的关键技术

嘉宾：齐路　编辑：靳韡赟

为什么要使用机器来理解音频内容呢？一个重要的出发点就是在大量数据存在的情况下，由人来完成音频内容的理解是一件较为困难的事情，在图片和文本处理方面，快速理解尚有一定实现的可能，古代有一个形容人记忆力很好的成语叫做走马观碑，描述一个人骑着快马路过一个石碑，看到石碑上密密麻麻的小字一瞬间就能够全部记下来。但是对于音频与视频这种内容，即使在加速的情况下也需要一定的时间来听完、看完音频和视频内容才能够进一步理解它。如果采取人力处理这些问题会遇到困难，我们就可以借助于机器辅助人来进行处理。

机器在理解音频的过程中需要理解哪些内容呢？就需要我们来分析场景问题。

1. 内容安全：在 18 年的时候，红极一时的一名主播因为在直播过程中发表一些不当的言论而遭到封杀。同样的案例还有因主播在直播过程中发表涉政相关的言论而被封杀。今年是建国 70 周年，很多境外的反动组织为了扩散他们的言论在某些直播平台或者社交平台散播宣传音频或者视频。他们通常用录音机等播放设备将提前录制好的音频和视频连续不断的进行播放，这是一个典型的社交问题。此外，直播中还存在较多的色情问题，包括视频、图像方面的色情，也包括音频方面的色情。有时也会有广告导流行为，所谓广告导流就是在某一个直播平台上，大家在音视频交流过程中有人发类似于我们私下加个微信聊吧，这样这个平台的流量就会被导走。我们的工作就是要在音频中识别出这些行为，为这些行为打上标签，让运营平台知道这些音频中存在这样或者那样的问题。以上所分析的问题完全属于截流问题。

2. 内容运营：内容理解的优势在于可以进行内容推荐，一个典型的例子就是在交友的社交平台上，如果通过声音识别出是一位大叔，就可以给他推荐一位萝莉，如果声音识别是一位御姐，就可以推荐给她一个正太，这样就有希望延长他们之间的交流时间。

内容理解核心的解释就是将非结构化的内容结构化，其中重要的途径是标签，通过各种手段为音频或者视频打上各种标签，方便后面去做各种处理。比如拦截和推荐。

3. 解决方案：对于上述问题，我们的解决方案主要包括四个步骤：

- 音频切分：在得到原始音频之后首先对音频进行切分，将长语音切分为多个短语音
- 音转文：将语音转换成文字
- 识别：对文字和音频分别打标签
- 合成：汇总片段结果，并给出最终的整条语句或者整个视频。

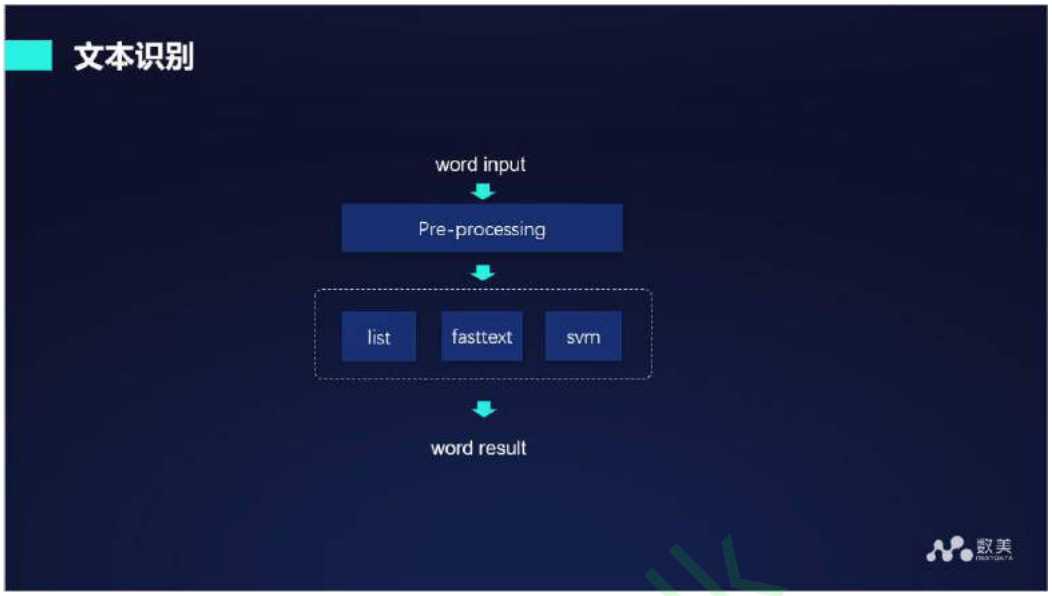
内容理解的过程中存在一定的困难，比如远场识别，对于直播来说识别过程中最大的困难是混响和噪声。主播在直播过程中为了吸引更多的人观看通常会唱歌，唱歌一般都会加混响来使歌声听起来有绕梁三日的感觉。但是这种情况就会对语音识别产生较大的影响。此外，我们在使用语音搜索和语音输入法的过程中为了获得更加正确的结果会故意放慢说话速度，表达相对更加清晰。而直播过程中为了获得良好的互动，说话都会比较随意。唱歌也是一个比较难解决的问题，在语音识别建模的过程中，很多情况下都是使用带音调的音素来进行建模，但是在唱歌的过程中语音的声调会发生变化。这样也会引起识别不准确的问题。目前已经有很多有效的方法来解决这些问题。

下面介绍下解决上述问题的相关工作，首先是**音频切割（VAD）**，这种技术是比较主流的一种方法，先通过深度学习 DNN 来预测出一段音频是静音还是非静音。然后通过加窗得到最终的结果。

在语音转写文字的过程中，采用 DNN+LSTM，然后使用 lattice-free MMI 方法训练现有模型，我们使用的语言模型是 ngram 方式。这是一个相对比较主流的框架。目前 **ASR** 主要解决的是把音频中的文字提取出来。

前面我们提到还有一部分语音识别不能通过转文字获得。比如是否有音乐，播放音乐的名称，是否存在色情声音等等。我们采用**声音分类**的框架来解决这个问题。首先需要对音频进行数据增强，因为在音频分类条件下数据的 label 并不均衡，特别是存在一些小众的声音，非常稀少，所以需要对这些数据进行增强。我们使用 TDNN+bi-GRU+Attention 框架。

上述第一步将音频转换成文字，第二步将分类信息标签集，第三步需要对转换出来的文字进行文字识别，**文字识别**主要包括文字的分类：基于一段文字判断它所属的类别，比如这段文字是不是属于色情话题或者是带有辱骂性等。模型不能完全解决这方面的问题，还需要有关键词类比。我们给出的框架通过 fasttext 模型或者一些传统的机器学习算法来进行模型分类，同时联合关键词信息进行处理。在使用模型训练之前首先对文本进行预处理，比如分词、归一化等。



最后一步是**行为识别**，例如境外反动组织要去散播反动言论会在平台上开很多账号，对于每个账号不会雇佣不同的人去宣传言论，而是使用录制好的音频来播放相同的言论内容，这样他的行为就会有一定的聚集性，在这种设备或者 IP 上的具体行为可以通过一个逻辑回归模型来打分。通过分数判断这个行为是不是存在问题。行为识别也可以检测出一些有问题的内容。

下图是我们整个框架的架构图，将上述我们分析的各个模块整合在一起。在模型层面包括 ASR 模型、文字相关模型、声音相关模型、行为相关模型、名单库等。通过引擎层输出各种各样的分数，最后我们有一套规则，规则引擎会对所有模型层面和画像层面输出的结果进行汇总，最终得到结论。

下面是我们真实的一个价值体现，某直播平台同时采用用户举报、人工抽审、数美智能审核三种方案监测平台直播内容。用户举报平均每天抓出 1 个违规音频，人工抽审平均每天抓出 20 个违规音频（审核团队 30 人），数美智能审核系统平均每天抓出 160 个违规音频，同时数美智能审核系统反馈音频转文字结果、自动记录违规音频位置、发生时间等信息。

DataFunTalk

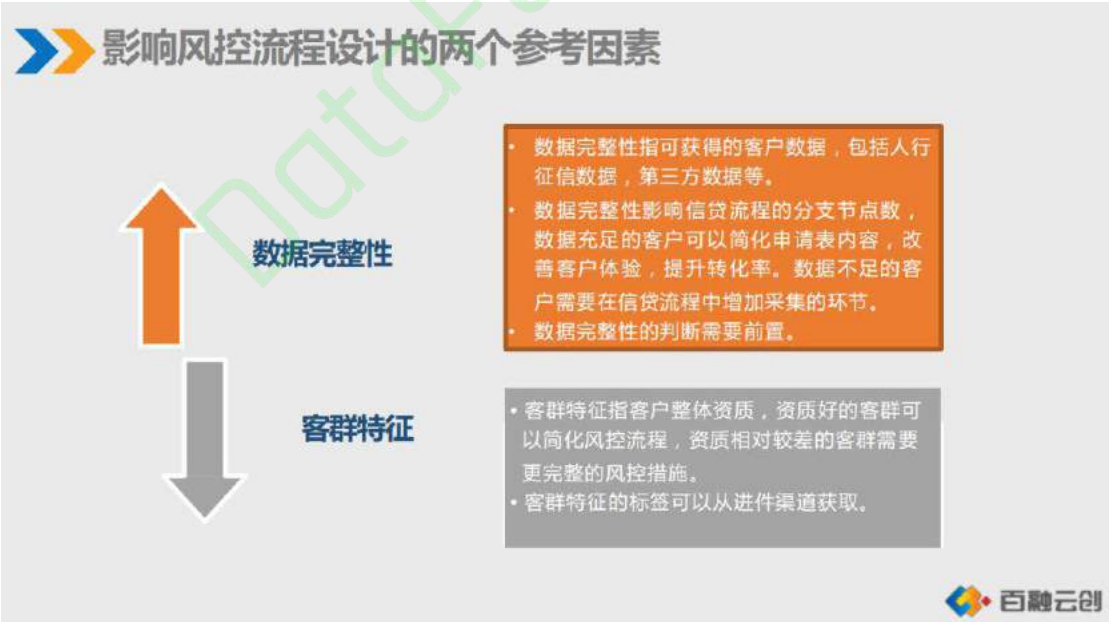
汽车金融风控流程设计和机器学习实践

嘉宾：殷翔　编辑：HohXil

本次分享内容大纲：

01. 汽车金融风控流程设计

汽车金融风控流程设计中需要关注的五个节点：获客、反欺诈、信用评估、额度、利率，整个风控流程的设计一直是围绕这几个点来设计的。

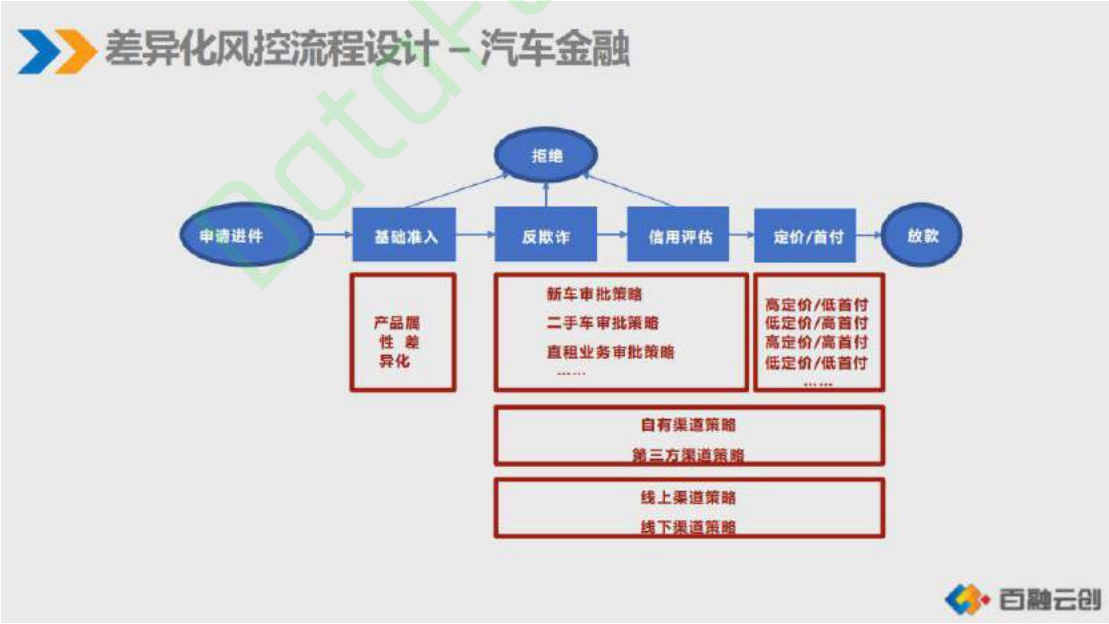


除了上述几点，还需考虑俩个重要因素：

1. 数据的完整性：是指可获得的客户数据，包括银行征信数据，第三方数据等，来补充我们的评估维度，维度越丰富，对前端客户提交申请的资料依赖就会越少，这样就会简化我们的风控流程，提高审批效率。

其次，数据完整性还会影响到风控节点的设计，因为每个节点都需要依靠数据来支撑，数据越丰富，设计节点的自由度越高。

2. 客群特征：通常我们需要做些精细化的审批，对于不同质量的客户提供不同的风控流程。比如，资质比较好的客户，会提供相对来说比较简单的流程；资质比较差的客户，会做一些更精细化的审批，把他们的风险做一个精细化的分层。另外还需要把客户按照不同的风险特征，作为不同的渠道进行分流，就是接下来的差异化审批。



差异化风控流程设计，不止在汽车金融领域，在其他业务中也是一个重要的思路。为什么要做差异化审批？因为我们可以把整个风控流程看成一个评估的准则或者标准。对于

不同客群，不可能使用同一套标准对他进行评估，通常会基于产品、场景、渠道上的差异对客群做划分，相对来说不同客群，他们的质量和风险特征是不一样的，需要我们在整个流程设计上做不同的分支，以及模型的搭建过程中，需要使用分客群来进行建模。

1. 汽车金融整体风控流程设计

前面介绍的是风控流程设计中比较关注的一些原则及关键点，接下来讲下基于上述各环节的基础上，我们在行业实践经验的一些总结。

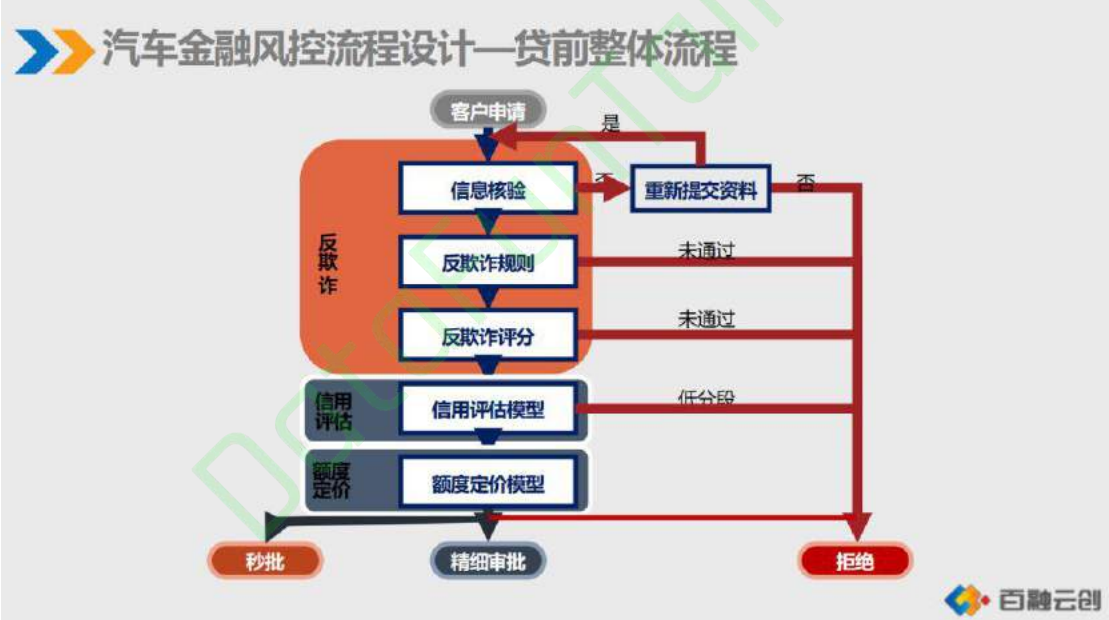
这张图就是我们在汽车金融整体风控的一个流程图，它覆盖了整个汽车金融生命周期的各个环节。大致分为五个环节：

- 准入 & 渠道评级
- 反欺诈

- 信用评估
- 贷中监控
- 贷后催收 & 回溯评级

详细内容见上图。

2. 贷前整体流程



首先讲下贷前的流程设计，这张图是比较简单的，真正在业务实践中，我们会在这张图中进行一些增删，比如橙色部分我们可以增加一些准入条件、分析客群，这张图比较通用一些，主要分为三个部分：

- 反欺诈

- 信用评估

- 额度定价

接下来是信息核验常用的一些维度。

反欺诈规则常用的维度，包括：

黑名单：通常金融客户会有自己维护的内容黑名单，同时可能也会和同行等其他金融机构进行分享，这是很重要的一部分。

申请行为：申请行为异常，这个通常是根据自己内容积累的数据，也有与其他机构共享的数据。

不良信息：是否有不良记录。

实名信息：指客户在历史的申请中是否使用了多个身份信息。

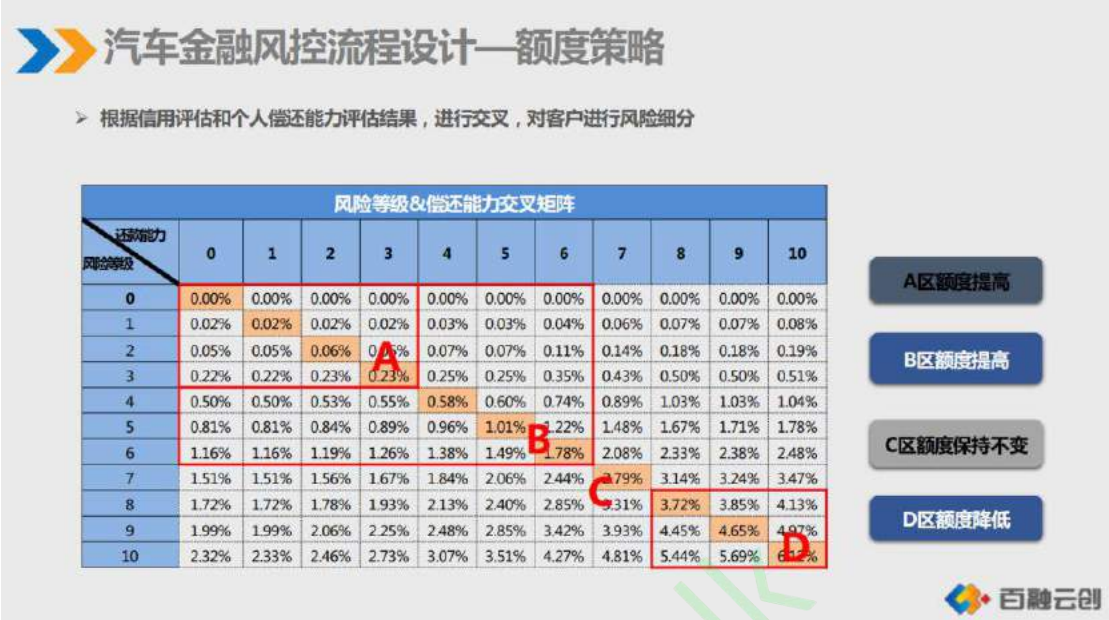
消费行为：有的公司在要求客户提交资料的时候，附上银行流水和消费的相关资料。

团伙欺诈：以上几个环节是对用户行为进行评估，团伙欺诈是对关系进行评估，通常会通过三要素等维度，进行关联，发现与他有关联的高风险客户或者在短时间内有相似行为的客户。

在做模型搭建，或者信用评估、反欺诈之前会做客群的划分。在汽车金融领域首先会做这样的一个划分（不同机构划分的可能不一样）：首先是厂商的汽车金融，然后是融资租赁、直租、二手车、商用车、车抵贷等。因为建模之前通常会要求样本是独立同分布的，不同的客群，他们的样本特征，包括分布特征都会不一样，所以需要做分客群的建模。其实这一步，也可以放在整个风控流程的最前面。

评分段	区间占比	区间逾期率	通过率	KS值
[300,350)	1.03%	65.05%	100.00%	8.62%
[350,400)	1.10%	49.09%	98.97%	15.27%
[400,450)	1.78%	37.64%	97.87%	23.07%
[450,500)	3.36%	29.55%	96.08%	33.82%
[500,550)	3.90%	22.88%	92.73%	42.54%
[550,600)	6.18%	13.61%	88.83%	48.06%
[600,650)	7.74%	10.09%	82.65%	51.02%
[650,700)	11.12%	5.95%	74.90%	48.59%
[700,750)	11.81%	4.41%	63.78%	43.38%
[750,800)	16.85%	3.33%	51.97%	33.30%
[800,850)	16.06%	1.37%	35.12%	19.15%
[850,900)	12.41%	0.56%	19.06%	6.75%
[900,950)	4.92%	0.41%	6.65%	1.73%
[950,1000]	1.73%	0.58%	1.73%	0.00%

做完模型之后，最重要的是怎么使用它，这里展示了使用的方式，通常需要把它转化成这样的一个表。这张表展示的是每个评分区间它的坏账率和 KS 值，通过这张表来划分一个阈值来审批。阈值的划分，通常是基于业务的需求。



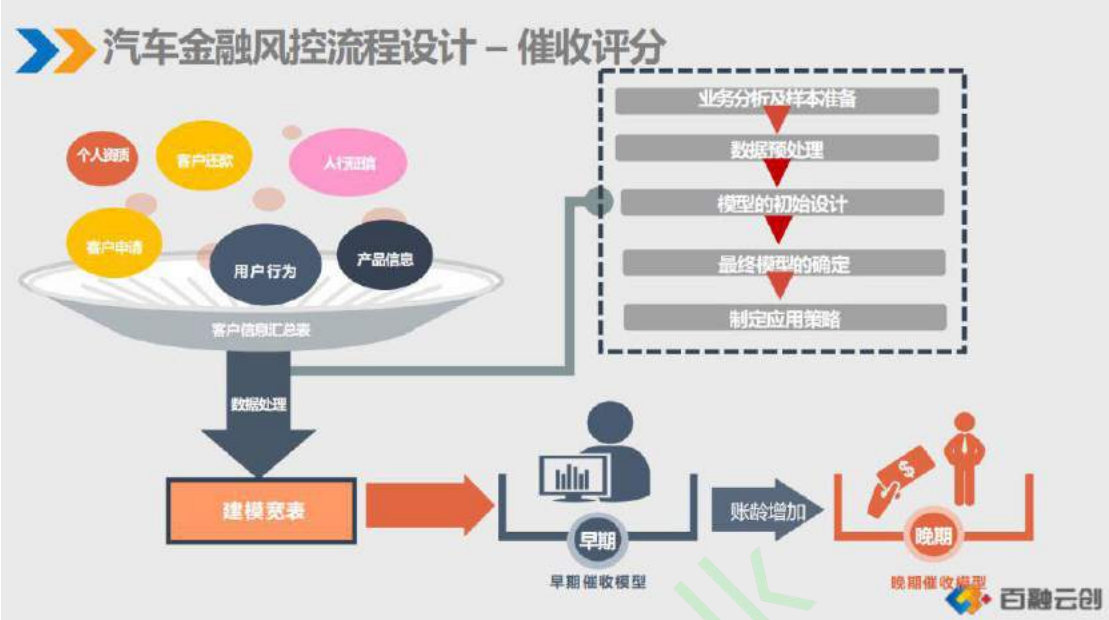
风险等级 \ 偿还能力	0	1	2	3	4	5	6	7	8	9	10
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	0.02%	0.02%	0.02%	0.02%	0.03%	0.03%	0.04%	0.06%	0.07%	0.07%	0.08%
2	0.05%	0.05%	0.06%	0.06%	0.07%	0.07%	0.11%	0.14%	0.18%	0.18%	0.19%
3	0.22%	0.22%	0.23%	0.23%	0.25%	0.25%	0.35%	0.43%	0.50%	0.50%	0.51%
4	0.50%	0.50%	0.53%	0.55%	0.58%	0.60%	0.74%	0.89%	1.03%	1.03%	1.04%
5	0.81%	0.81%	0.84%	0.89%	0.96%	1.01%	1.22%	1.48%	1.67%	1.71%	1.78%
6	1.16%	1.16%	1.19%	1.26%	1.38%	1.49%	1.78%	2.08%	2.33%	2.38%	2.48%
7	1.51%	1.51%	1.56%	1.67%	1.84%	2.06%	2.44%	2.79%	3.14%	3.24%	3.47%
8	1.72%	1.72%	1.78%	1.93%	2.13%	2.40%	2.85%	3.31%	3.72%	3.85%	4.13%
9	1.99%	1.99%	2.06%	2.25%	2.48%	2.85%	3.42%	3.93%	4.45%	4.65%	4.97%
10	2.32%	2.33%	2.46%	2.73%	3.07%	3.51%	4.27%	4.81%	5.44%	5.69%	6.12%

除了欺诈和信用风险评估，通常还会做额度的策略。通常会把信用的评估和个人的评估做一个交叉，对客户进行风险细分，图中左上角表示偿还能力强，信用风险低，右下角表示偿还能力差，这样可以把用户分为 A、B、C、D 四个等级，A 区的客户资质比较好，足够强的偿还能力，信用风险比较低，那么就会对这部分用户提高额度空间。对于最差的区间，会提高首付比，降低额度空间，这样就会提高违约的成本。

3. 贷中管理

刚才讲的是贷前的环节，汽车金融和其他产品相比有个比较明显的特点，就是风险释放周期长，图中展示了大致的风险释放周期，我们可以看到过了 18 个月，也仅仅是释放了 70%的风险。所以，这要求我们做好贷中监控的环节。需要及时监控客户的还款状况，还有信用风险的变化，以及偿还能力的变化，提前做好预警，或者催收前置。对于资质比较好的客户，我们可能会做一些存量的营销。

4. 贷后催收



最后是贷后环节，一个粗暴的方式是按照账目来分 M1、M2、M2+这样分给不同的催收员，来专门催收。

更精细化的方式，是催收评分卡，根据客户的还款意愿来贴标签，搭建这样的评分卡，来评估客户的还款意愿，指定更精细化的催收策略。

02. 汽车金融风控中的机器学习实践

接下来分享汽车金融风控中建模的一些实践经验。

机器学习是近几年比较火的词汇，在各个领域都有涉足，但实际上真正在商业领域有很成熟的应用场景的地方并不多，比如推荐系统、安防领域用的比较成熟。其次在金融风控的领域，应用也比较成熟。

机器学习通常可以分为四个部分：

- ❖ 监督学习（supervised learning）
- ❖ 无监督学习（unsupervised learning）
- ❖ 半监督学习（semi-supervised learning）
- ❖ 强化学习（reinforcement learning）

在汽车金融风控中，我们主要关注的是监督学习，也就是我们关注的模型是基于有标注的样本进行训练的。

在建模之前，我们需要对模型的目标进行设定。其中区分度和稳定性，是绝大部分场景对模型的要求。

区分度：就是说你的模型要有足够的区分度，可以对目标客群和非目标客群进行清晰的划分。

稳定性：就是要足够的稳定，实际业务中和建模建模样本上它们的表现要相似。

排序能力：汽车金融领域有一个独特的要求，就是排序能力，就是评分分组要足够的均匀、正态、平滑，同时非目标客户的占比在评分上要尽量明显的单调。为什么这么做？这主要基于策略的制定，评分的策略上，主要是基于阈值的划分，如果排序能力不足，会导致评分阈值划分，稍微调控下，就会使策略结果急剧的变化。

这是我们在汽车金融领域建模的完整流程和方法论：

定义主题：主要是标签的设定。

数据探索：这部分会做风险特征画像，看是否需要对样本进行分客群的建模；其次，需要做一些数据的清洗。

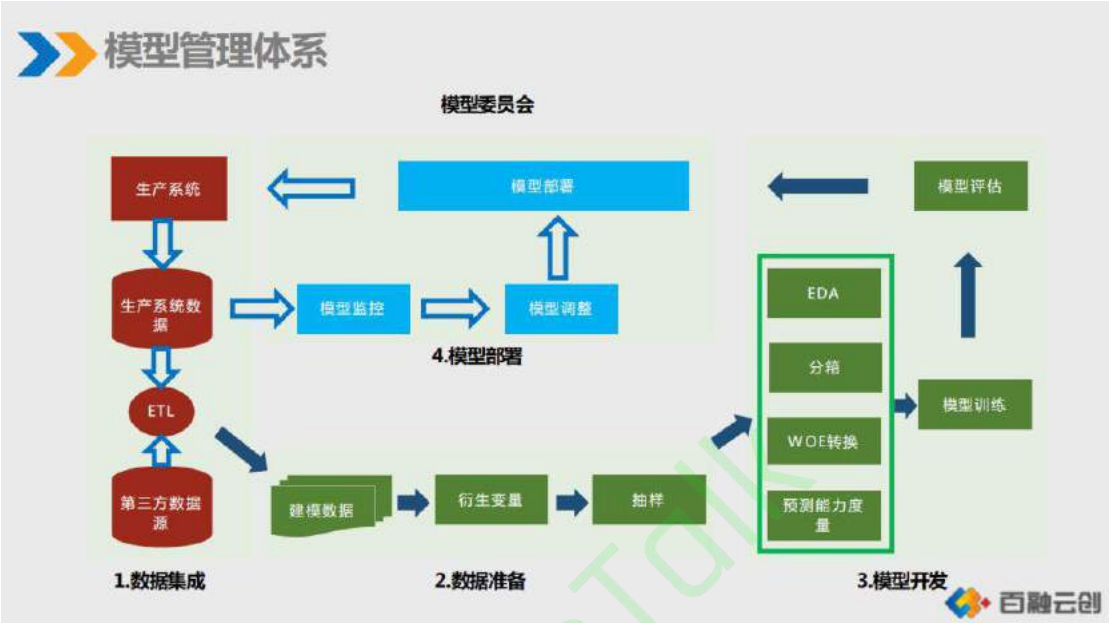
特征建模：主要是对特征重要性进行排序，然后对特征的一个转换，因为大部分的模型可能都不擅长处理离散型的数据，所以这里会把离散型的数据转化成连续型的数据。

建立模型：这部分需要选择和我们业务实际贴近的模型。

验证模型：验证模型的质量和稳定性。

实施模型：包括模型的上线部署和监控调优。

这是一个比较完整的建模流程。



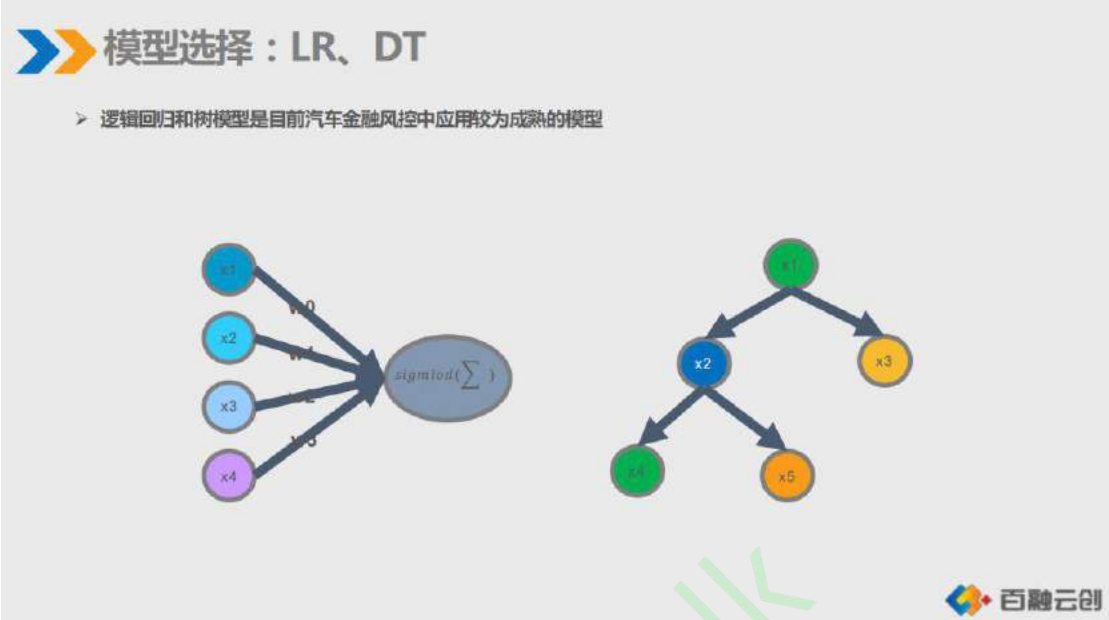
说到模型，可能我们比较关注模型的使用，和模型搭建本身，还有非常重要的一点，就是模型管理体系。因为只有完善的模型管理体系，才能持续不断的输出高质量的模型。完整的模型管理体系详见上图，主要包括：数据集成、数据准备、模型开发、模型部署几方面。

在汽车金融领域，模型评估主要关注：

KS：指好坏区分度的一个指标，衡量的是模型的精准度。

PSI：主要描述模型分布的稳定性。

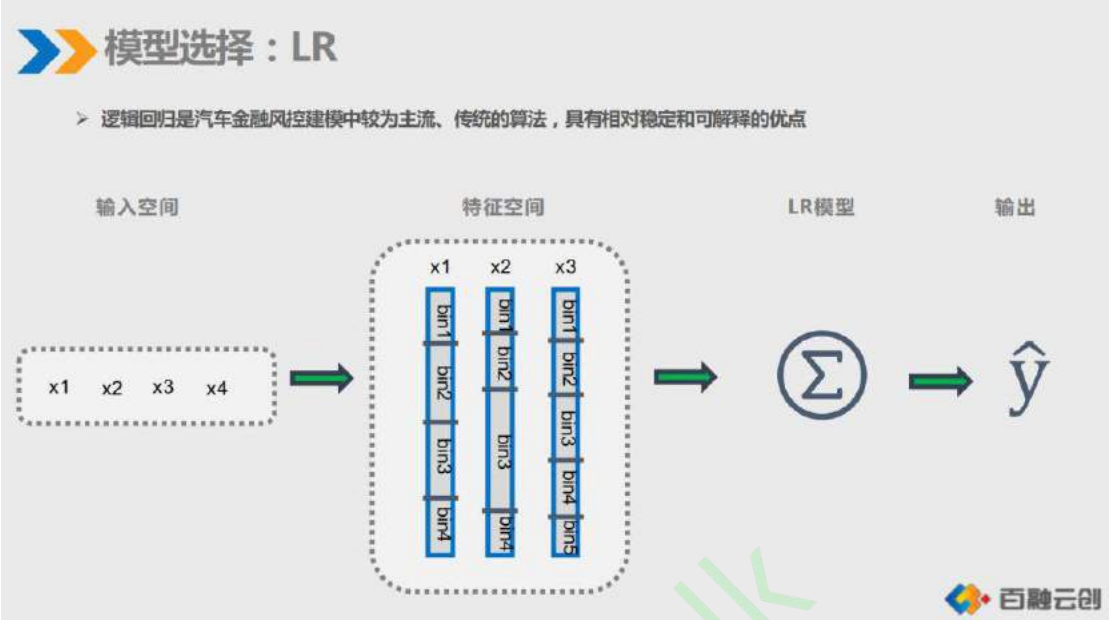
模型分布：需要尽可能的满足正态分布，好坏客户在模型中区分明显，随着模型的增加或递减，坏客户占比有明显的单调变化。



逻辑回归和决策树模型是目前汽车金融风控中应用较为成熟的模型。

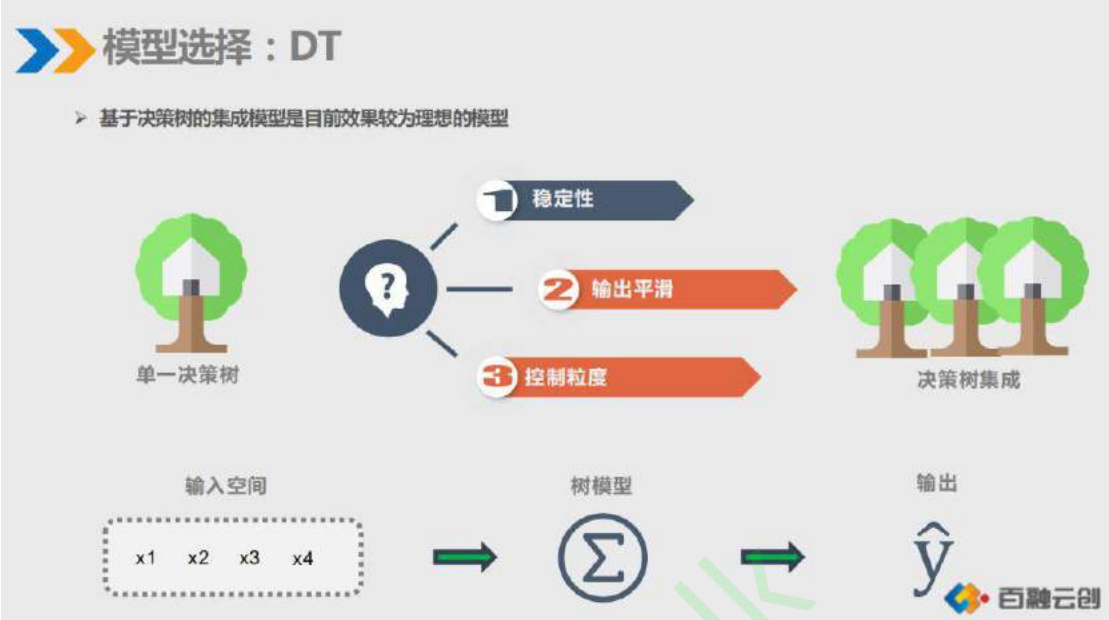
逻辑回归：比较好理解，逻辑回归做完之后可以转换成一张评分表，这样就可以直接变成另外一种形式来展示，通过不同维度打分，最终对各个维度的分数进行求和，得到最终的评分。

决策树：更好理解，它就是 if 和 else 这样的一个流程。



详细介绍下逻辑回归，因为具有相对稳定和可解释的优点，所以它目前是汽车金融风控建模中较为主流、传统的做法。

算法流程，首先对变量进行转换 WOE 分箱，作用是使模型更稳定以及提高非线性数据的拟合能力。但是整体来说，逻辑回归对高维数据，或者比较复杂的数据，处理能力相对比较弱，尤其是非线性数据。



决策树模型，它的拟合能力非常高，尤其是对非线性数据，它的缺陷是非常容易过拟合。解决方法：采用决策树集成算法进行处理，相对单个决策树来讲除了稳定性外，它的输出相对平滑，容易控制粒度。

集成算法通常分为三种：

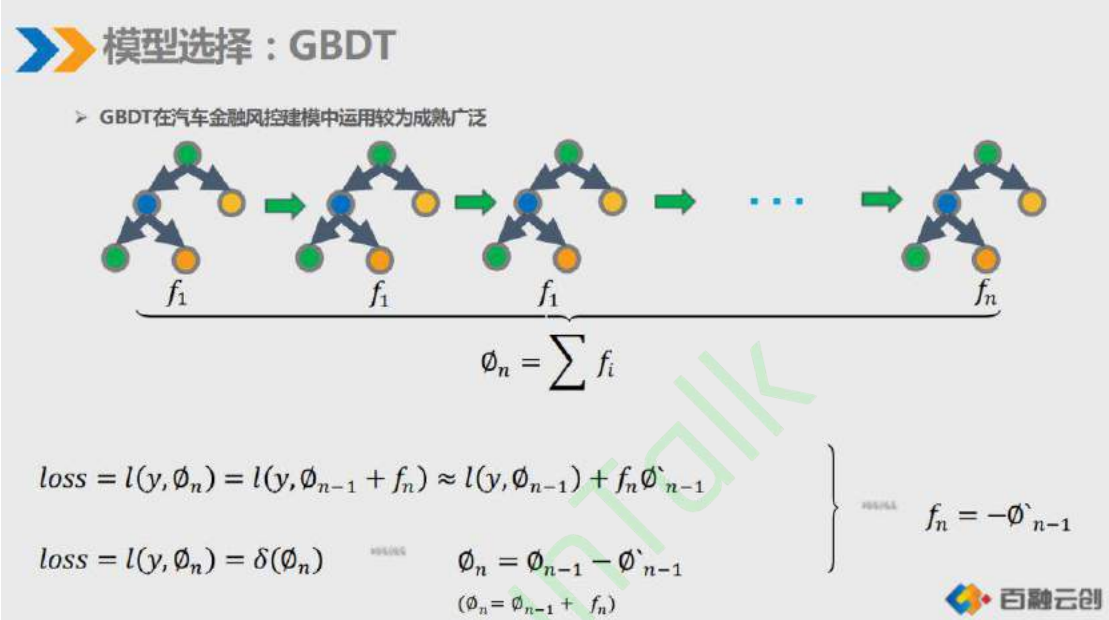
Bagging：选取多个决策树取平均值

Boosting：提升模型，从弱学习开始（AdBoost，xgBoost），串行计算

Stacking：堆叠模型，聚合多个决策树

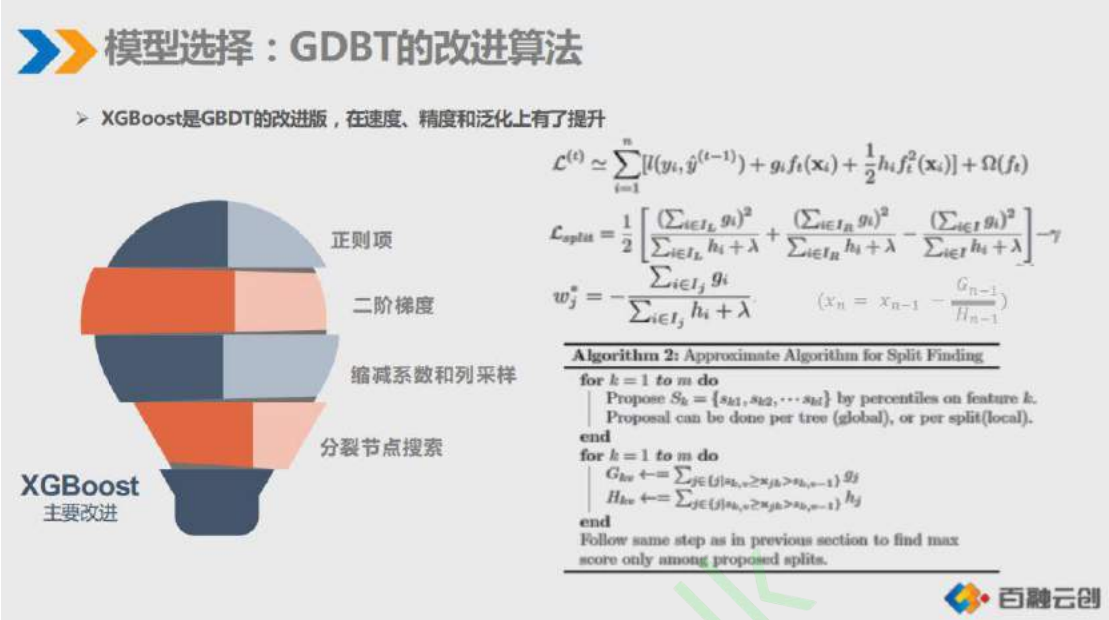
目前用的比较多的是 Boosting 算法，在汽车金融风控中用的比较成熟的就是 GBDT 算法。这种算法就是端到端的学习，我们只需要把我们的特征输入进去，不需要人工的干

预，直接输出结果，开发难度低，但需要我们对模型理解更深入一些，这涉及到精细化调优。



GBDT 模型，是一堆决策树的串行集成，当前新生成的决策树，依赖于之前所有的决策树。并不是用决策树来拟合标签，而是用决策树来拟合当前模型负梯度的方向。为什么这样做，我们做了推导（见上图），有俩个 loss 函数的形式。第一行公式是对 loss 函数进行一阶展开，理解展开之后分成两项，第一项为常数项，我们忽略掉，为了保持每次增加新的决策树的时候，loss 函数能够逐渐减少，我们就要求第二项恒为负。如何恒为负？其实主要让这两项符号相反就可以了。

第二个是从数值优化的角度理解，我们可以把树看成数值，然后通过梯度下降理解它，直接按照梯度更新，结合下面的公式，就是当前模型等于之前树的集合+新的树。把两个公式结合在一起就可以推导出后面的公式。



然后是对 GBDT 的改进算法，就是通常所说的 XGBoost 算法，相对于 GBDT 做了如下改进：

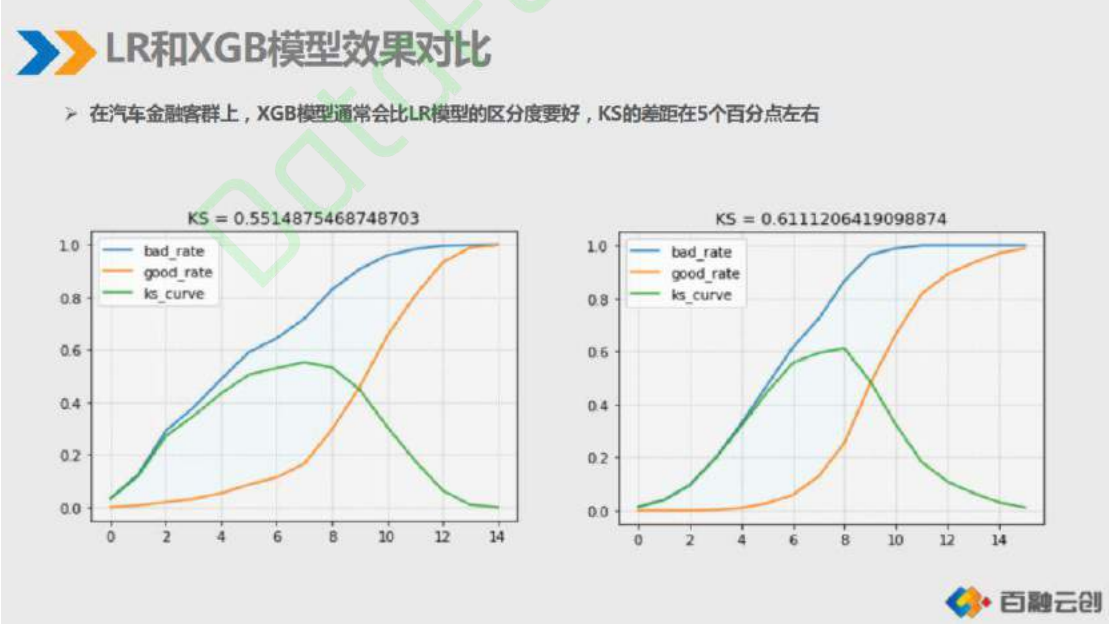
在 loss 函数里加入了正则项，提高了模型稳定性。

在更新的过程中使用了二阶梯度，来保证训练更快、精准。

缩减系数和列采样，提高泛化能力。

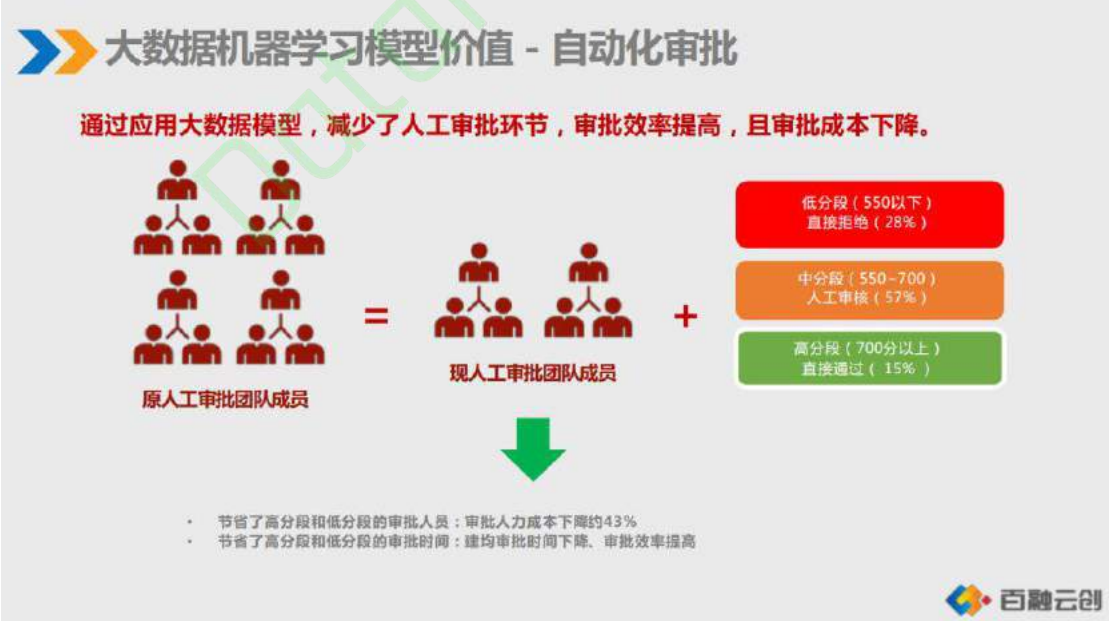
分裂节点搜索，这里并没有采用遍历的算法，而是采用基于样本的梯度，来做分裂树的划分，作为搜索的节点，这样提高了搜索算法的性能。

除了 XGBoost，还有其他的一些改进算法，包括 LightGBM 和 CatBoost。如果数据量比较大，可以采用 LightGBM，如果需要在模型中处理类别变量，可以采用 CatBoost。



模型效果对比如上。

效果对比，使用传统模型和机器学习模型效果的对比。



大数据机器学习模型的价值：通过应用大数据模型，减少了人工审批环节，审批效率提高，且审批成本下降。

上图为，汽车金融大数据风控发展的趋势，主要包括如下几方面：

- 业务扩展
- 客群下沉
- 数据复杂度上升
- 自动化审批
- 互联网大数据的使用
- 机器学习

03. 自动化机器学习平台

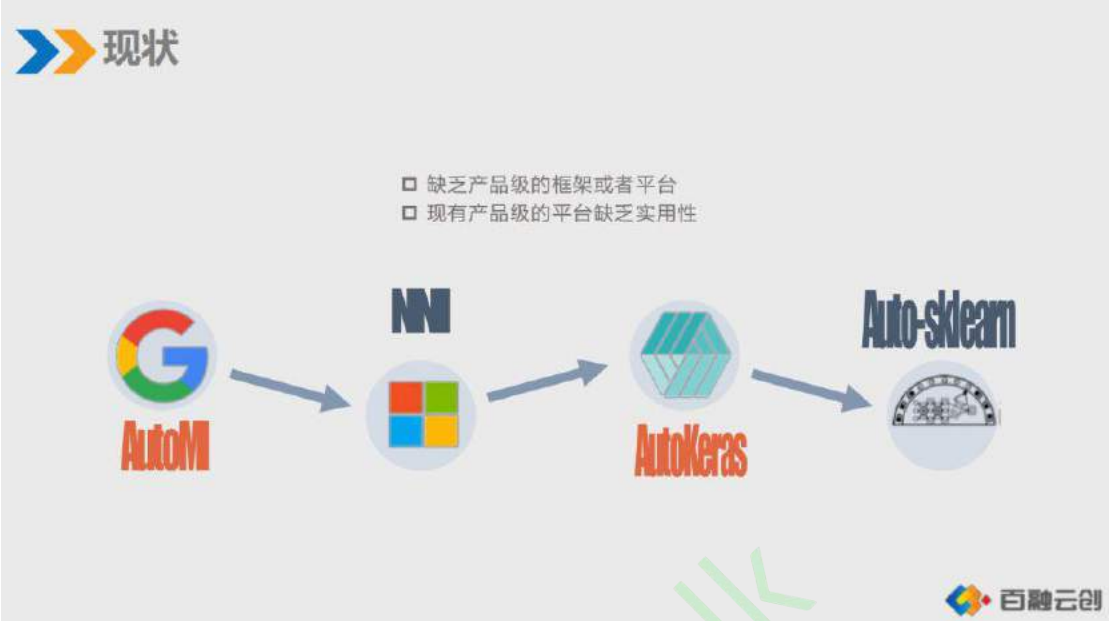
当前风险建模所面临的困难：

门槛高：尤其是机器学习模型。

手动调参的效率低下：模型复杂度高，需要调整的参数非常多。

周期长：尤其是逻辑回归模型的周期非常长，需要对数据做非常多的处理，包括分箱，调整等；然后机器学习虽然建模时间没有那么长，但是调参会特别耽误时间，并且需要经验的支撑。

建模环境和生产环境的割裂：比如一个非常复杂效果非常好的模型建好之后，怎么部署，怎么应用比较麻烦，会涉及到各方面的开发。



基于上述痛点，我们做了一个集成到平台的尝试。上图为市面上，比较流行的框架，不算是成熟的商业产品，这些平台通常是工具型的，算法层面上，并不能提供直接的商业上的使用，只包含建模一个环节，并不包含整个流程。

基于上述原因，百融做了自己的自动机器学习平台，主要包含如下功能：

数据方案、样本方案、清洗方案、处理方案、模型方案、调参方案，并且这些方案都可以做到复用。在数据方案，不但可以上传本地的数据，还可以通过平台的方式对接各方的数据，并且针对不同的样本或者数据源，可以进行独立的清洗方案和处理方案，当模型建完之后，可以通过一键部署。

这是我们针对上述建模痛点，做的一些改进，首先是可交互环境。将建模过程中涉及的编程、数学、算法等细节进行封装，提供交互友好的图形化界面，简化建模流程，提高建模效率，降低建模门口。

工具链的高度整合。将数据分析、数据可视化、建模等工具整合在一起，实现数据挖掘平台、数据可视化平台等模型训练平台于一体。

最后，是把建模环境和生产环境无缝连接，将整个数据分析到模型部署全流程覆盖，降低建模复杂度，提高开发效率，缩减开发周期。

04. 总结

本次分享主要是为大家分享下百融在汽车金融风控领域的一些设计经验，以及我们在风控模型搭建中的技术积累和尝试的一些经验，然后总结了建模过程中遇到的问题，以及百融在解决这些问题中的一些尝试。

DataFunTalk

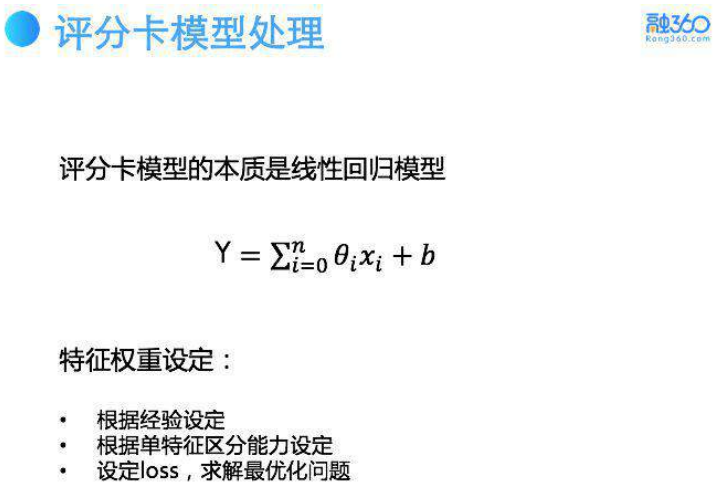
机器学习在信贷风控建模中的优势和挑战

嘉宾：罗灿　编辑：郑智献

在国内国外金融风控领域大致分为两个流派，其中一派为具有统计学背景的人，分布在银行、金融消费公司等传统的金融领域，偏好评分卡进行建模。另外一派则是具有互联网背景的新兴探索者，将机器学习、深度学习等方法运用于金融风控领域。融 360 是 ToB 的金融风控企业，不仅跟传统的金融公司有合作，还和京东、百度金融、滴滴等互联网背景的公司有交集。本次演讲内容为这两种流派优劣势对比及机器学习、深度学习在金融风控领域的实践应用。

一、信用评分卡模型

评分卡模型虽然是一个简单的权重线性加和的回归模型，但在传统金融风控领域使用了上百年。



1、评分卡模型中的特征筛选、参数求解

评分卡模型对特征筛选有以下偏好：

1）特征覆盖率高，通常达到 70%以上；

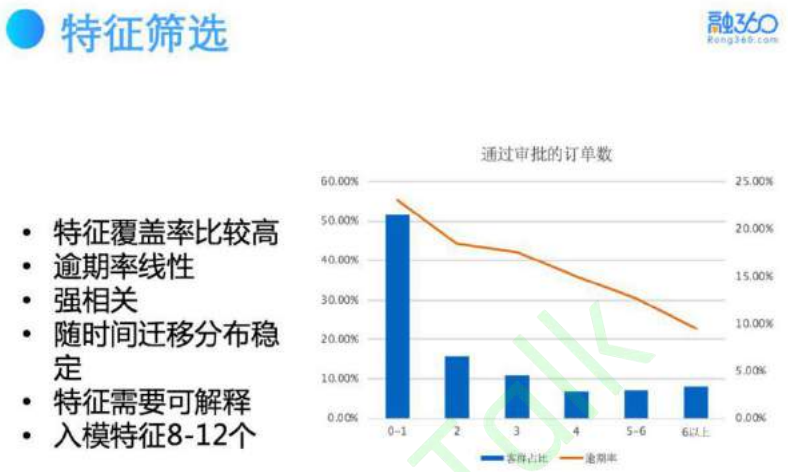
2）特征与逾期率为线性强相关；

3）特征随着时间迁移，其分布保持稳定；

4）特征变量与风险趋势的相关性有明显的可解释性，并且入模变量少，通常为 8-12 个。

以上特征筛选的偏好是为了保证入模的特征变量的稳定性、有效性，比如会有黑名单、负债、资产等特征变量。总的来说，在少样本量及强特

征的情况下，使用评分卡这种简单的模型能够很好的保证稳定性和有效性。



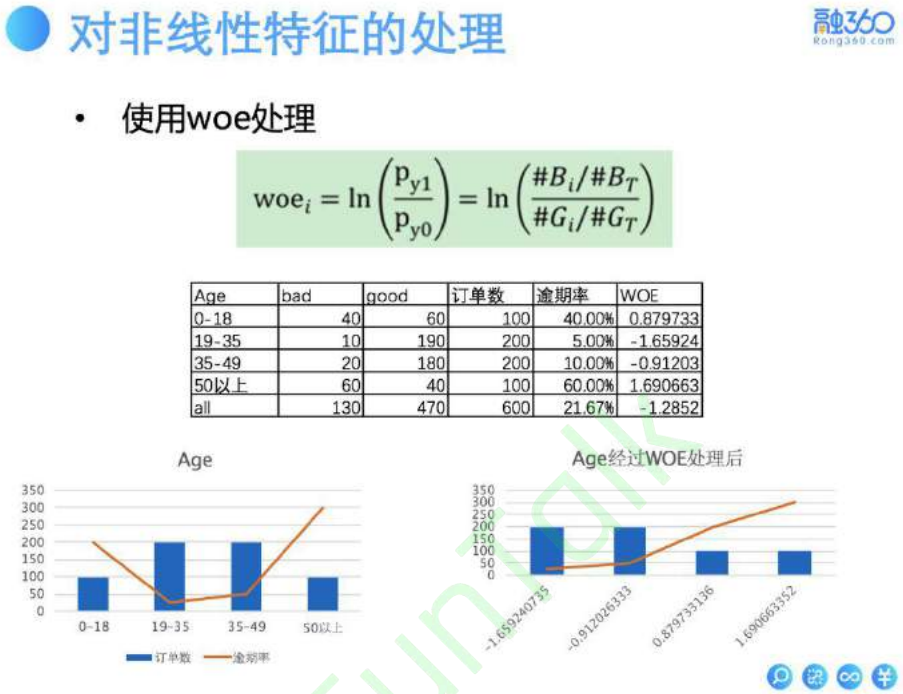
关于参数求解，在机器学习中通常设定损失函数，将其转化为凸问题，再进行求解。但是在评分卡模型中，若行业样本非常稀缺，对参数训练收敛有一定的困难。若当前样本量为 0，可根据专家的历史经验进行权重设定。若当前样本为几百个，可根据单特征区分能力比如 KS / IV 值等进行权重设定。

2、评分卡模型的非线性 / 交叉特征处理

线性回归模型有一定的劣势，比如不能解决非线性相关特征及交叉特征的问题。

2.1 非线性特征处理 — woe / 分桶

对于非线性特征线性化有 woe 处理和分桶两种方式。

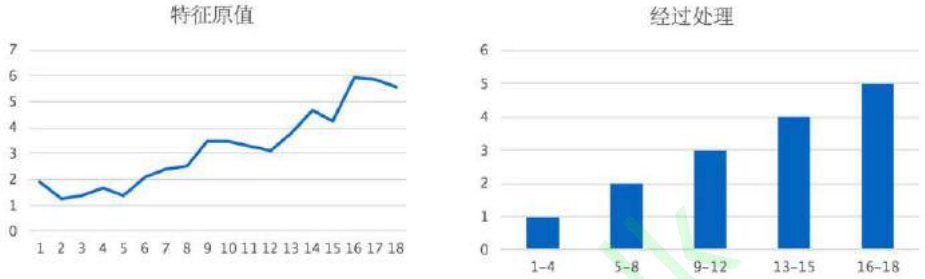


Age	bad	good	订单数	逾期率	WOE
0-18	40	60	100	40.00%	0.879733
19-35	10	190	200	5.00%	-1.65924
35-49	20	180	200	10.00%	-0.91203
50以上	60	40	100	60.00%	1.690663
all	130	470	600	21.67%	-1.2852

1）左图中年龄与逾期率呈非线性相关性，对年龄特征做 woe 特征变换后，在右图中年龄与逾期率呈现了一定的线性相关性。再将变换后的年龄特征带入线性回归模型，就能比较好地体现和刻画年龄特征的价值。

对非线性特征的处理

- 使用分桶的方式处理
  - 根据分布分桶
  - 根据odds分桶
  - 根据IV值分桶



2）对有一定跳变的连续值特征进行分桶，其优势在于弱线性特征转化为强线性特征，并且可以增加模型的鲁棒性。

2.2 交叉特征处理 — 分群

分群

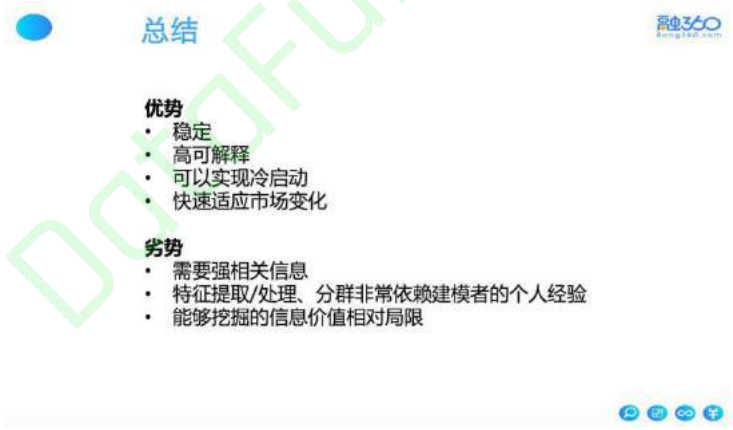
根据各种客群的基本属性对客群进行细分，在每一个细分客群上单独建模

- 提升特征的区分能力，从而提升模型的区分能力，本质是一种交叉特征的体现
- 提升特征的稳定性，从而提升模型的稳定性

对于交叉特征的问题，可使用客户分群的方式。根据基本客群的基本属性划分，比如是否有征信，是否为 VIP 等基本属性划分，再对细分客群进行单独建模，拟合风险。以上在一定程度上相当于做了交叉特征，该过程类似于决策树的建立，根据经验和业务指标建立决策树，将人群的划分作为不同的叶子节点。

最后，结合以上特征处理的技巧，在小样本量上进行线性回归就可以得到稳定且区分度高的模型。

3、总结



评分卡模型的优势有稳定、高可解释性、可实现冷启动以及快速适应市场变化。其中冷启动优势的表现在于不论初始样本量的多少就能快速建模。其劣势为特征要求强相关，且非常依赖建模的个人经验，挖掘的信息价值相对有限。

二、机器学习的出现

随着机器学习的出现，传统金融逐渐向互联网金融转变。

1、传统金融和互联网金融的差异及新要求

传统金融的订单数少、订单金额高、贷款期限长、客群资质好、风控预算高。互联网金融订单数多、订单金额低、贷款期限短、客群资质差、风控预算低。风控的预算高低主要体现在订单金额的高低和期限的长短，其共同决定了贷款利率的高低。每一笔订单金额越高、期限越长，每一笔收获的利息越高，即风控预算越高。

针对互联网金融独有的特性，对风控提出了新的要求。

1）目前具有央行征信的个体覆盖占全国的 50%左右，对于剩下 50%客群资质较差的个体，央行征信的风控系统无法进行覆盖。互联网金融的服务客群就是这剩下的 50%。所以要使得互联网金融信贷用户下沉，就得要求风控模型覆盖的人群更广，准确地说覆盖大量传统风控不能评估的个体。

2）互联网金融借贷的期限短，模型要求有较短预测有效期。

3）互联网金融风控成本低，要求风控数据价格低。

2、金融风控数据

基于以上风控的新要求，对风控数据进行如下整体分析。

数据是根本，建模方法要根据数据进行不断地调整。将风控数据分为四部分，资质数据、信贷数据、消费数据、行为数据。其与逾期率的相关性从前往后逐渐减弱。

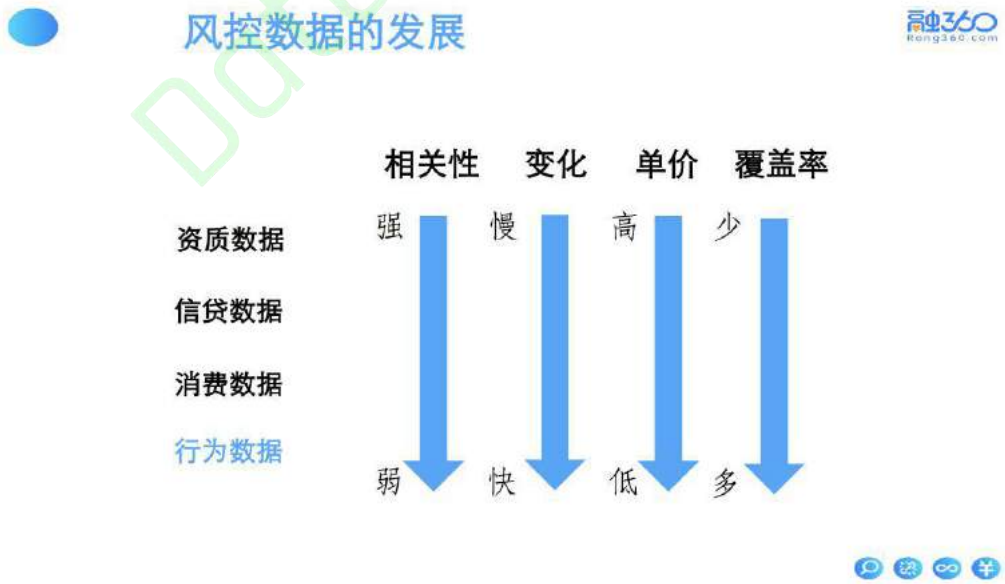
1）资质类数据包括该个体是否有房车、教育程度、工资水平等个人属性相关的数据；

2）信贷类数据包括历史的借贷额度、借贷次数、还款情况、是否逾期等借贷相关的数据；

3）消费数据包括在电商平台的消费记录等流水相关的数据；

4）行为数据包括个体之间的联系、手机 App 的使用列表、手机型号、浏览内容类别等个人行为数据。

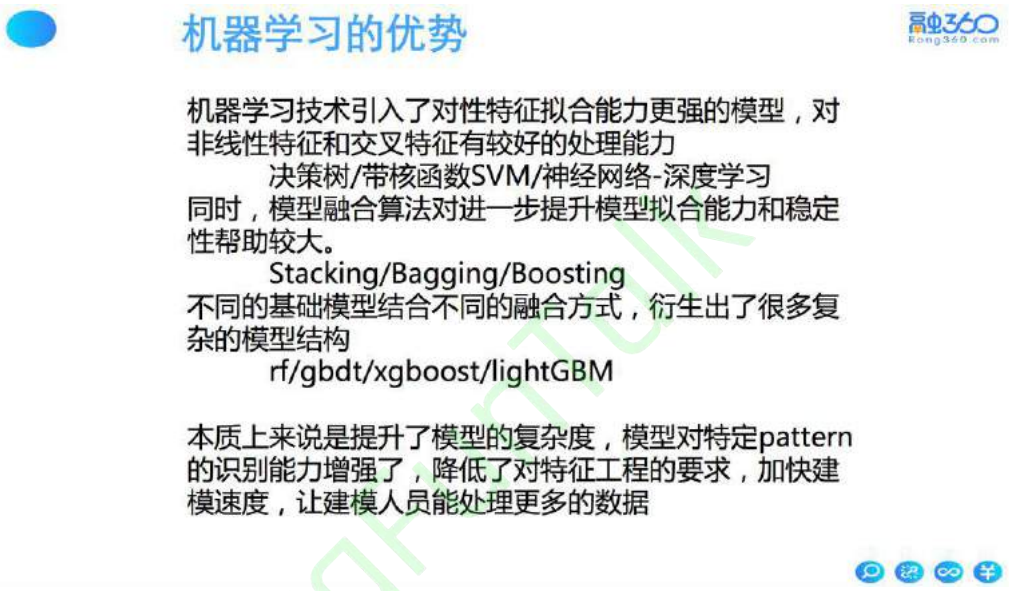
互联网金融中资质较差的客群基本没有资质数据以及合规的信贷数据，虽然有一定的消费数据但是该类数据大多掌握在巨头电商平台手中，所以所能获取到的就是开放度高的互联网行为类数据。行为类数据的优势为覆盖广、成本低，缺陷为相关性弱、变化快。



3、机器学习的优势及问题

互联网金融的数据特征相关性弱，那么要求模型的挖掘能力要强，所以机器学习的重要性就显现出来了。

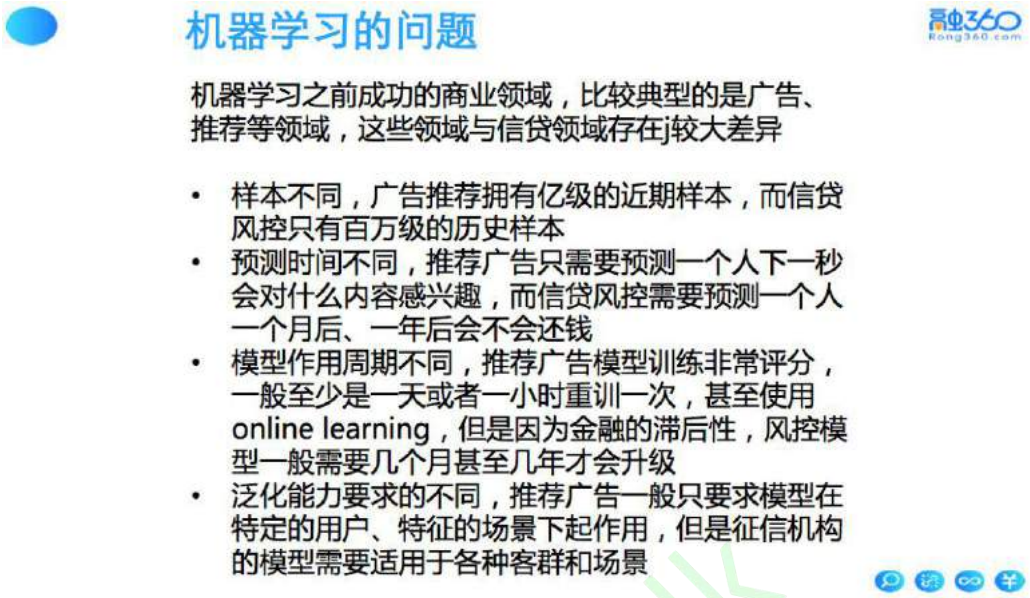
机器学习有以下优势：



1）特征的拟合能力更强，对非线性和交叉特征有更好的能力；

2）能够基于不同的基础模型融合得到拟合更强和更稳的模型，比如 rf / gbdt / xgboost / lightGBM 。本质上，提高了模型的复杂度，模型对固定 pattern 的识别能力增强。总的来说，机器学习能从海量的、贫瘠的数据中挖掘出数据的价值。

机器学习在商业上应用比较成功的领域有广告、推荐等，但是与信贷领域有较大的差异：



1）样本量级不同，广告推荐亿级近期样本，信贷风控只有百万级历史样本；

2）预测时间不同，广告推荐实时预测，信贷风控是未来长周期的预测；

3）模型更迭周期不同，广告推荐能短时间内进行模型更迭，能捕获到短时间内的样本特征回馈。信贷风控模型具有滞后性，样本回馈需要较长的周期，建模的训练数据与预测数据存在着较大的时间跨度；

4）泛化能力不同，广告推荐面向的客群比较特定，信贷风控模型需要适用于广泛的客群和场景。

想要在信贷风控的场景成功地利用机器学习，必须要做一些迁移和适应。

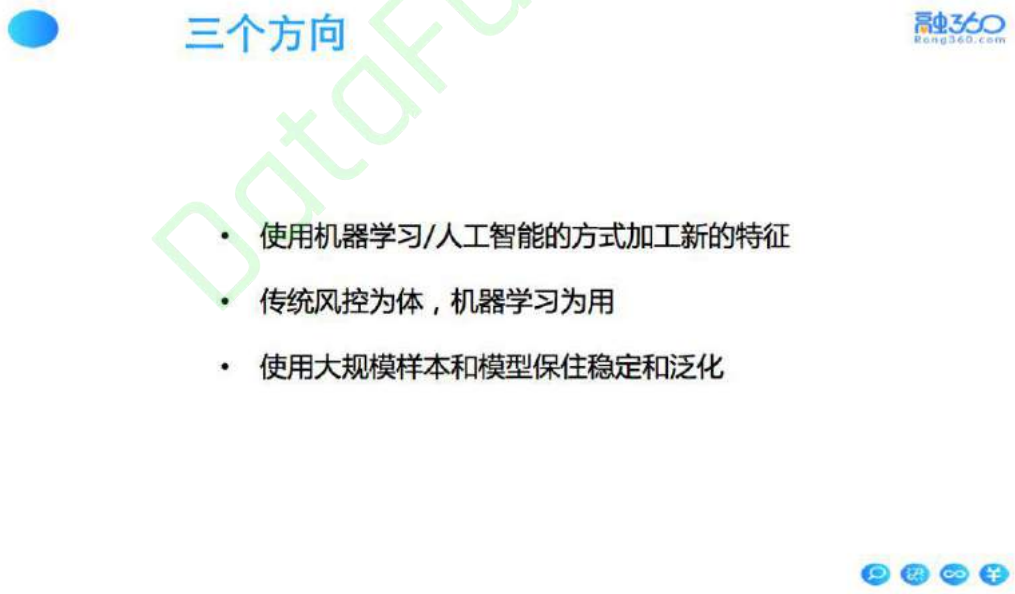
三、机器学习技术的突围

在信贷风控领域关于机器学习技术的探索主要分为三个方向。

第一个方向，既然构造复杂模型存在着不稳定的风险，最稳妥的方式为使用机器学习 / 人工智能增加新特征，再使用评分卡模型。

第二个方向为传统风控为体，机器学习为用。即特征筛选的标准和规则仍不变，仅替换评分卡模型为复杂模型比如 xgboost 等。

第三个方向为大规模样本结合机器学习，保证模型的稳定和泛化。

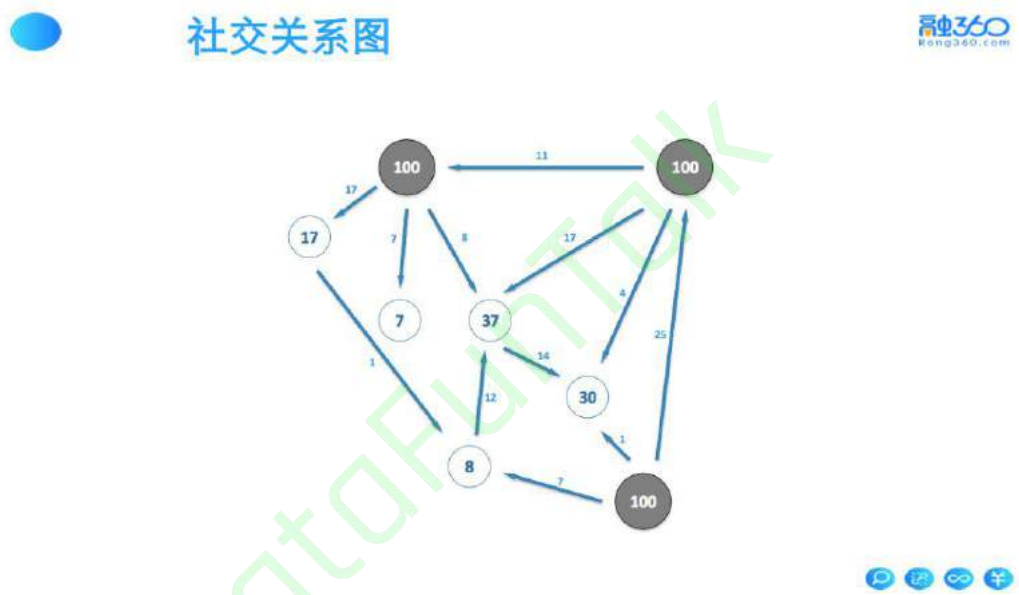


1、使用机器学习 / 人工智能加工特征、风控为体-机器学习为用

关于机器学习 / 人工智能加工特征，做了以下的实践尝试。

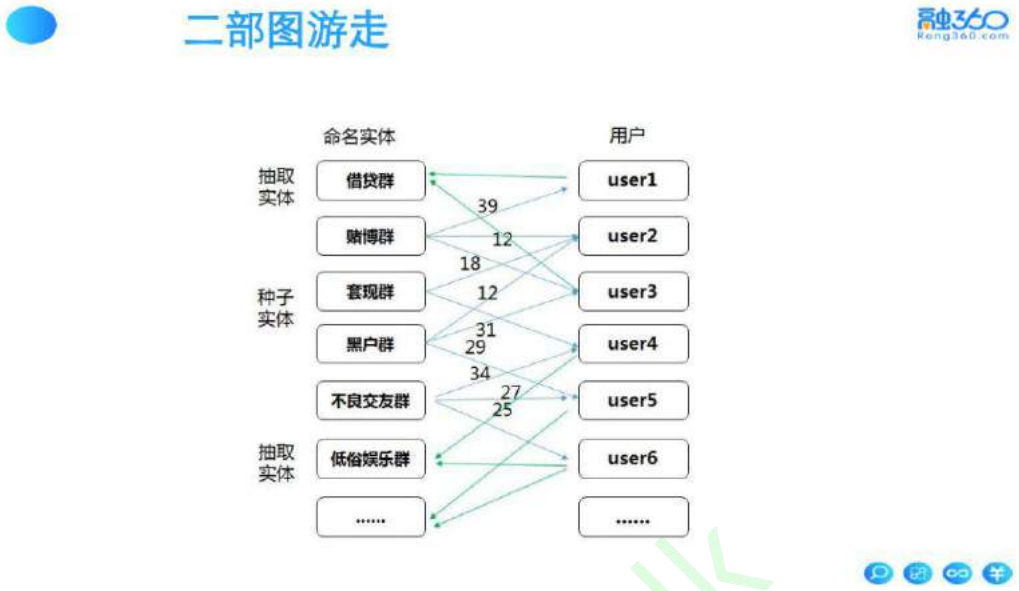
1.1 社交关系图 / 知识图谱

将用户看为一个点，用户之间的联系为边。基于图结构构造特征，比如图的出 / 入度 ，度中黑名单的个数，黑用户 randomwalk 将度权重传导至相邻的边等衍生特征。



1.2 二部图游走

以下为通过二部图来衍生特征，首先建立用户与实体的二部图，实体可以为 QQ 群，App ，浏览链接等等。用户与实体特性相关则连接一条权重边，将有反面特性的实体作为种子实体，向用户侧游走，用户侧又向实体侧游走，直到游走平衡进入稳态可以得到稳定的权重。将稳定的权重作为特征，来标识反面特性的程度。



1.3 word2vec

word2vec 主要应用于 NLP 领域，在风控领域中的运用鲜少。目前该方法应用于页面埋点动作的特征提取，将点击和浏览动作看作词，用户完整周期内的所有动作行为组成了段落和文章，再使用 word2vec 方法，将用户的点击浏览行为隐含表征为一定维度的向量，再将向量作为用户的特征输入 xgboost 进行建模，单个特征有 1-2 个 ks 点的提升，效果还是不错的。

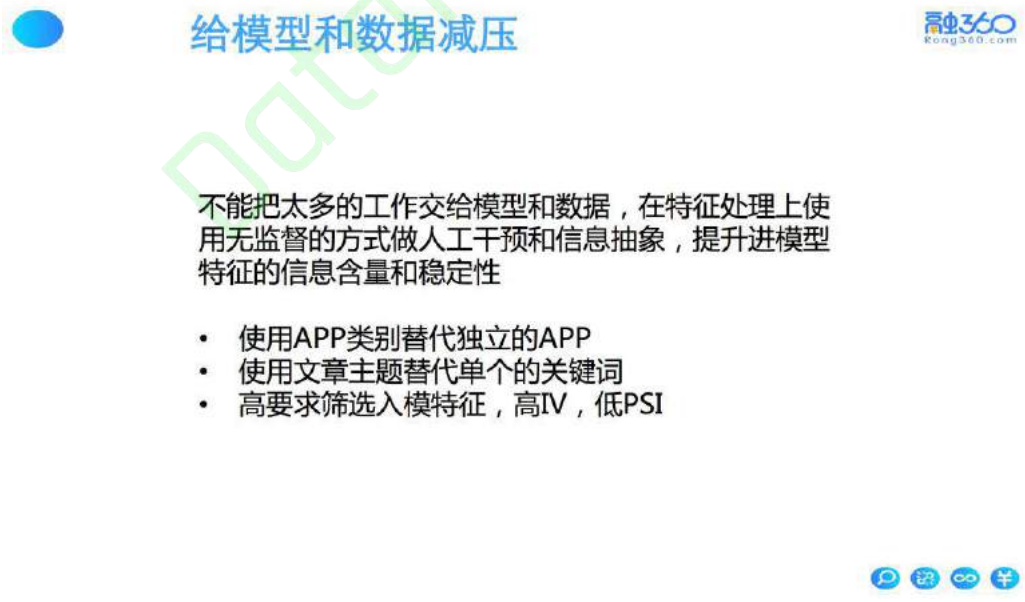
1.4 订单序列模型

用时序的模型如 LSTM 处理用户某一个订单的相关特征，如期限、金额、时间、到期账单等，其中一个订单表示一个时序事件，用户生命周期所有订单构成完整的时序链。将历史用户周期内的订单特征灌入

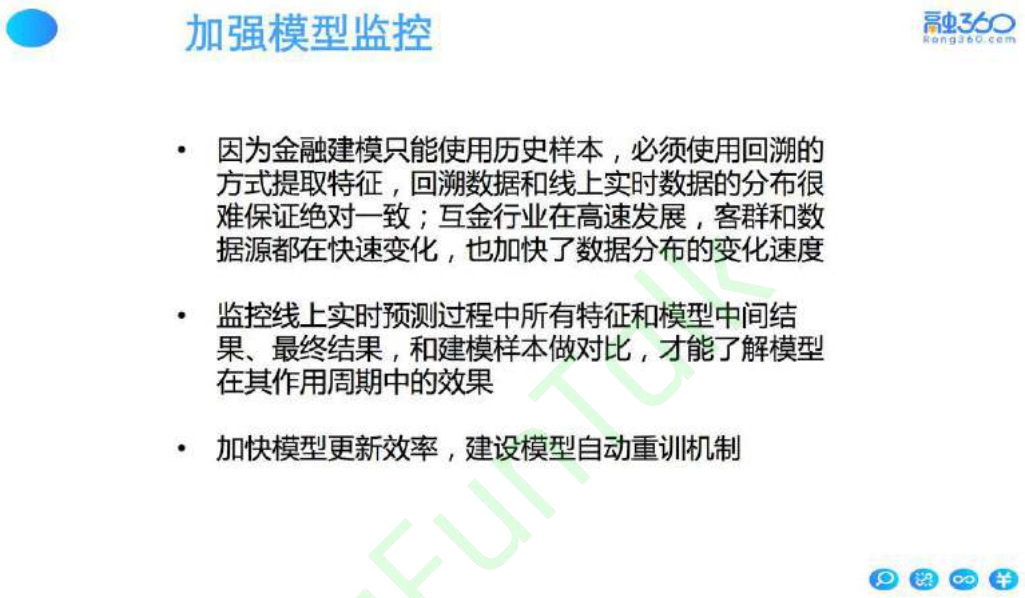
LSTM 得到（1，N）维特征，再将这些特征输入 LR 模型进行训练。该模型对于跨时间不是太稳定，但是相较之前还是有较大的提升，后续还需要持续优化。

2、模型和数据的减压、模型监控

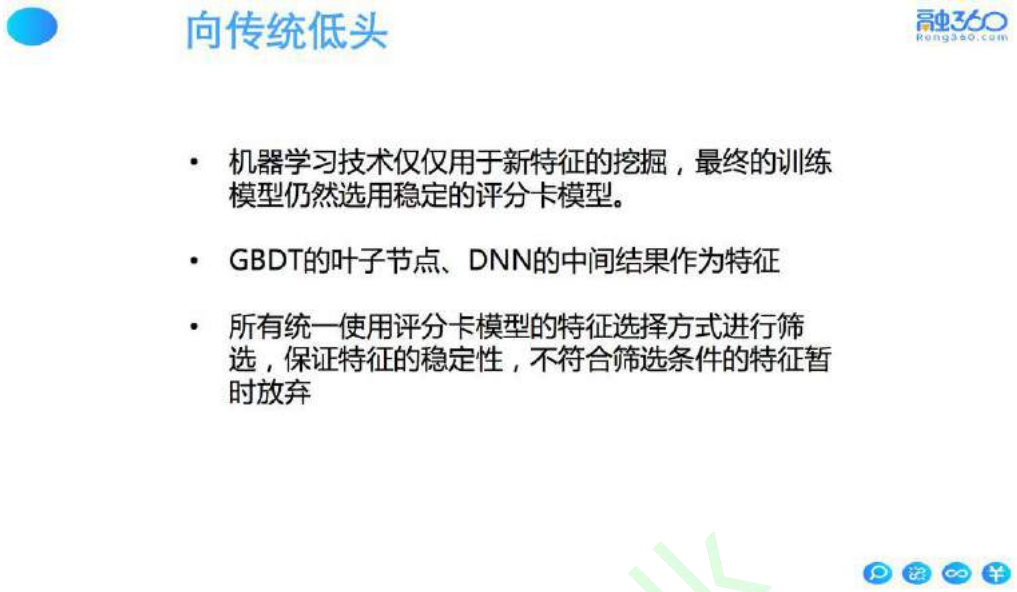
以传统的风控为体，机器学习为用的方式，虽然使用了机器学习方法但还是向传统靠拢。该方式强化了人工干预和信息抽象，弱化模型的复杂度和数据，以此来提升模型特征的信息含量和稳定。比如以高要求（高 IV 低 PSI）筛选入模特征，或者 LDA 文本主题词浓缩大量文本信息等方式来给模型减压。



为了保证模型的稳定性，可加强对模型的监控即加强数据和特征的分布及中间结果的监控。同时可加快模型的更新效率，建设模型自动重训练机制。



以传统风控为体，机器学习为用的方式能够使得模型构建过程更加透明，特征构建更加灵活。也就是机器学习技术仅仅用于新特征的挖掘，最终仍以高要求筛选特征，入模评分卡。



3、大规模机器学习

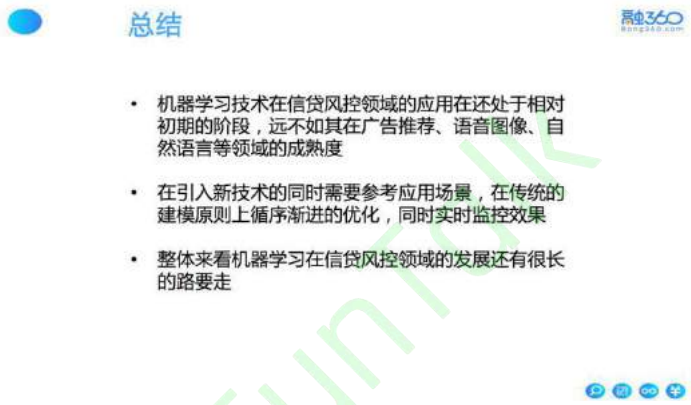
在去年 9 月份开始尝试大规模机器学习模型 — XGB-LR / deepFM / Deep&wide ，大规模机器学习有以下要求：

1）百万级别的样本。对样本的一致性要求不是很高，可将时间跨度大的订单样本作为输入。

2）数万维的特征，数十万参数的模型。大容量模型意味着多参数，多特征。其中输入特征为所有特征 ,即不经过高要求（ 高 IV 低 PSI）筛选入模。其主要风险点为线下线上模型效果差异大 ,即线下建模效果佳，完全拟合了线下分布，但是上线后可能由于过拟合等原因导致效果不佳。所以到目前为止，该模型一直处于线下陪跑测试阶段，监控其稳定性。

传统风控为体-机器学习为用方式建模的模型效果，对比大规模机器学习的模型效果，大部分时间内，其效果要优于传统风控为体-机器学习为用方式，并且有一定的稳定性。

4、总结



在个人技术成长的角度有以下总结：

1）机器学习技术在信贷风控领域远不如广告推荐、语音图像、自然语言等领域成熟，成长期大概在 5-6 年，还有非常多的机器学习的点值得探索。

2）在引入新技术同时需要参考应用场景，比如新技术在广告领域 auc 达到了 0.99 等，但是在风控领域其实不然。所以需要充分考虑风控领域的特性，将机器学习的优势迁移过来。

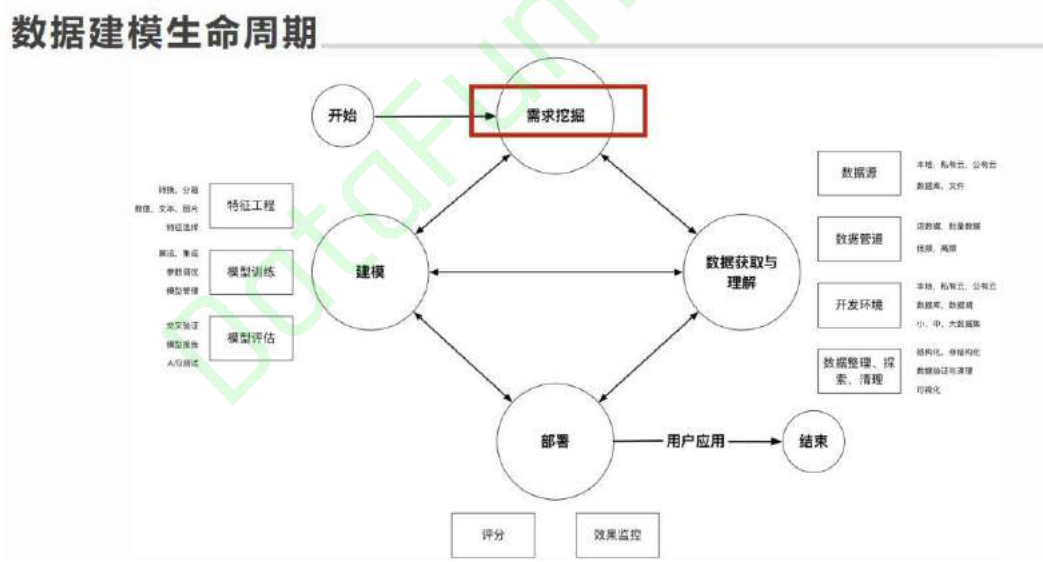
3）机器学习技术在信贷风控领域还有广阔的发展空间和前景。

风控建模流程：以京东群体感知项目为例

编辑：曾军崴　　整理：白云飞

导读：本次分享将以群体风险感知为例，从**需求挖掘、数据挖掘、建模再到最终的模型部署应用**，详细介绍全流程的风控建模方案。下面将从这几个方面出发，详细的讲解具体流程中的概念，同时会给出每个流程中的**目标、实现方法、交付物**，让同学们在具体实践的时候有目标，有方法。

业务需求挖掘



任何项目的起源，都是从需求挖掘开始的，不可能一蹴而就，会经过多次循环。这体现了作为数据分析师和建模人员的功力，可能有的业务要 A，你就给他 A，而没有往下深入几个层次进行考虑，**因为有的时候，可能业务的需**

求自己也没有想清楚需要的是什么，这和传统的软件工程是一样的，所以宁可在这里多花一点时间，也好过到后面踩坑之后，在回来重新挖掘需求。

① 目标

找出关键变量（与判断项目成功与否的指标相关，最好的方法就是定义几个场景，比如做风控，是坏账率下降，还是机注量下降了，或者其他指标），并把关键变量作为模型目标

识别业务已经使用或者需要获取的数据源

② 实现方法

定义目标，与用户以及其他利益相关方一起理解，识别业务问题，并构建多个问题(这些问题要能定义出业务目标同时也能作为建模的目标)

识别数据源：基于问题找出能帮助回答的数据源，越多越好

③ 交付物

项目需求文档（比如 A、B、C 三个场景，机注量下降 2%，或者坏账率下降 1%等等）

数据源梳理，有几张宽表，有几个数据源，都需要弄清楚

④ 业务方的诉求与抱怨

业务方的诉求与抱怨，抽象来说就是多、快、准、省：

多：在风控反欺诈领域，对于抓黑产，召回率越高越好。

快：实时反应，不能等到 T+1 时间过去了才反应，比如拼多多的风控事件，不仅仅需要做到实时，最好做到事前，并且可以发出告警，提示哪个环节被攻击。

准：准确率（auc，roc 曲线）大家都比较熟悉，尤其对于金融风控，准确率要求更高。

省：省心，傻瓜式操作，不需要告诉具体指标的含义。对于模型，只需提供给业务方 0 或 1 即可。

所以，对于业务的诉求和抱怨，就在这四个字之中，当我们接到群体风险感知这个 Case 时，我们首先需要做的就是：定义群体风险感知（为什么叫群体而不叫团伙？），明确业务场景，找出业务痛点。

两个场景：

- 一堆人来买我的货，我不会认为他们是坏的人；
- iphone 新品首发，黄牛带货，对我的品牌或者收益造成损失，这时，我们需要进行控制打压。

所以我们交付的东西，需要是中性的，**把模型看作一把刀，具体刀用来做什么，由业务来决定。**

⑤ 需求分析

风险未知可能出现的状态：

少量交易造成巨大损失（例如：网络舆情事件，诈骗，网络攻击等）

大量交易造成巨大损失（例如：拼多多事件，上万个账号薅优惠券，然后快速的变现）

黑灰产的主要特征：

逐利性(不断升级，不断寻找漏洞）：由于他们是在为自己牟利，没有 kpi 没有 996，甚至一天可以更新迭代 4 个版本，他们自己干的很起劲。

群体性（短期，快速，大量获利）目前成熟的电商，基本上不会让一个账号短时间内下很多单了，现在的黑产为了达到短期快速大量获利，一般使用群控手机，即用软件控制几百台手机，尤其在 6·18 和 11·11 时，给我们造成的压力也是比较大的。

项目的定义：

以黑灰产群体作为特征指标，发现交易链条上可能存在的漏洞增强防控能力，提高黑产成本（黑产本质上是生意，风控人员不是为了消灭黑产，是为了让他们这门生意做不下去，明面上是技术的对抗，实际上**是成本的对抗**）。

⑥ 项目目标

监控指标体系与异常发现（输出结果形式）：

线索账号的线索：某个特定群体的行为（观察某个小白鼠在迷宫里的行为特征）

群体维度线索：多个群体聚集情况（观察一群小白鼠在迷宫里的群体共同行为）

监控指标体系与异常发现（输出结果内容）：

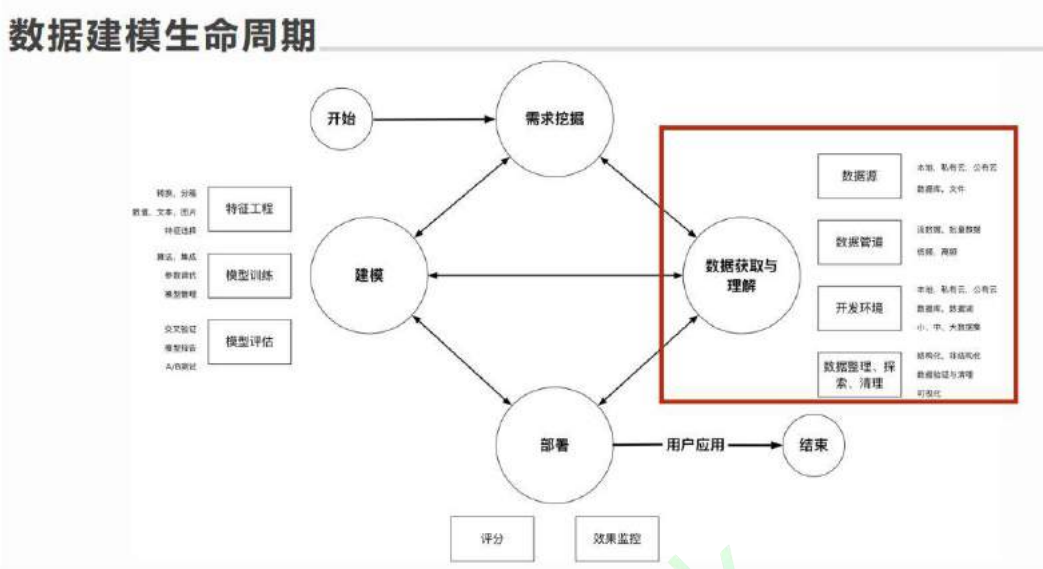
聚集性指标，灵活制定规则（告诉业务方，商品、库存、优惠券是不是被攻击了）

提前发现风险点，预警后续环节（黑产领券之后，会不会有什么大动作，需要提前预警）

SKU，优惠券，订单，相关账号（我们有一个评分，一旦被群体模型识别，就打一个黑标签，让风险值变高）

新的套利，作弊方式，交易漏洞（将黑产规律输出给分析师，分析黑产的作弊方式以及交易环节的漏洞）

数据获取与理解



数据获取与理解，主要分四个部分：数据源（在哪分布，可能需要做数据同步，也有可能需要买一些服务，定期做 API 的推送，需要把数据源定义清楚），数据管道（如何做清洗、加工，如何接入到模型中），开发环境（根据数据规模，选择开发环境），数据整理、探索、清理：结构化、非结构化，数据验证与清理，可视化等等。

① 目标

产出一个干净、高质量的数据集，充分的理解数据集和项目目标变量的关系

把数据集放置在一个适合开发环境以便进行后续建模

开发数据管道的解决方案，定期对数据进行刷新

② 实现方法

- 将数据导入目标分析环境

- 研究数据以确定数据质量是否足以回答项目要解决的问题（PS：前段时间比较火的生物探针技术，可以获取一些黑灰产的特征，但这项技术落地的时候，回收的数据缺失率很高，很难得到应用）

- 设置数据管道来抽取新的数据或者定期刷新数据

③ 交付物

- 数据质量报告

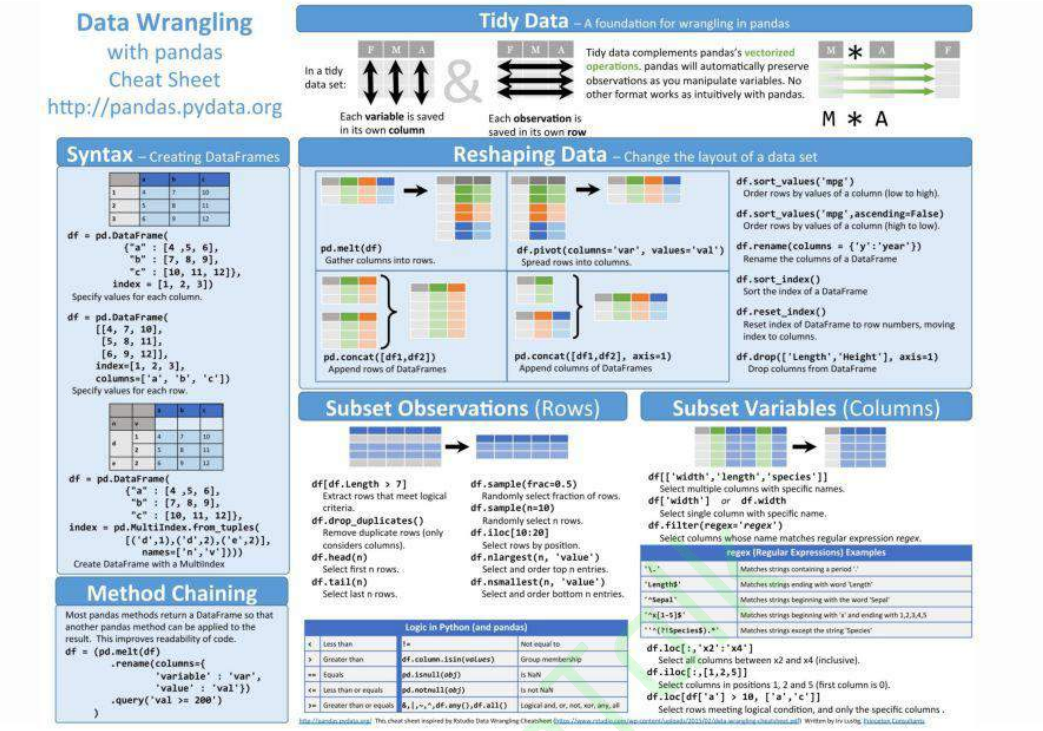
- 数据管道架构

- 可行性评估

④ Tity Data（with Pandas）

具体地址：http://pandas.pydata.org 包含了语法的介绍，重塑，聚合，缺失数据处理，窗口函数，可视化等等。

DataFunTalk


Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org
Tidy Data – A foundation for wrangling in pandas
In a tidy data set:
Each variable is saved in its own column
Each observation is saved in its own row
Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.
M * A
Syntax – Creating DataFrames
df = pd.DataFrame({"a" : [4 ,5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = [1, 2, 3])
Specify values for each column.
df = pd.DataFrame([[4, 7, 10], [5, 8, 11], [6, 9, 12]], index=[1, 2, 3], columns=['a', 'b', 'c'])
Specify values for each row.
df = pd.DataFrame({"a" : [4 ,5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = pd.MultiIndex.from_tuples([('d',1),('d',2),('e',2)], names=['n','v']))
Create DataFrame with a MultiIndex
Method Chaining
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.
df = (pd.melt(df).rename(columns={'variable' : 'var', 'value' : 'val'}).query('val >= 200'))
Reshaping Data – Change the layout of a data set
pd.melt(df) Gather columns into rows.
df.pivot(columns='var', values='val') Spread rows into columns.
pd.concat([df1,df2]) Append rows of DataFrames
pd.concat([df1,df2], axis=1) Append columns of DataFrames
df.sort_values('mpg') Order rows by values of a column (low to high).
df.sort_values('mpg',ascending=False) Order rows by values of a column (high to low).
df.rename(columns = {'y':'year'}) Rename the columns of a DataFrame
df.sort_index() Sort the index of a DataFrame
df.reset_index() Reset index of DataFrame to row numbers, moving index to columns.
df.drop(['Length','Height'], axis=1) Drop columns from DataFrame
Subset Observations (Rows)
df[df.Length > 7] Extract rows that meet logical criteria.
df.drop_duplicates() Remove duplicate rows (only considers columns).
df.head(n) Select first n rows.
df.tail(n) Select last n rows.
df.sample(frac=0.5) Randomly select fraction of rows.
df.sample(n=10) Randomly select n rows.
df.iloc[10:20] Select rows by position.
df.nlargest(n, 'value') Select and order top n entries.
df.nsmallest(n, 'value') Select and order bottom n entries.
Logic in Python (and pandas)
< Less than
> Greater than
== Equals
<= Less than or equals
>= Greater than or equals
!= Not equal to
df.column.isin(values) Group membership
pd.isnull(obj) Is NaN
pd.notnull(obj) Is not NaN
&,|,~,^,df.any(),df.all() Logical and, or, not, xor, any, all
Subset Variables (Columns)
df[['width','length','species']] Select multiple columns with specific names.
df['width'] or df.width Select single column with specific name.
df.filter(regex='regex') Select columns whose name matches regular expression regex.
regex (Regular Expressions) Examples
'\.' Matches strings containing a period '.'
'Length$' Matches strings ending with word 'Length'
'^Sepal' Matches strings beginning with the word 'Sepal'
'^x[1-5]$' Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*' Matches strings except the string 'Species'
df.loc[:,'x2':'x4'] Select all columns between x2 and x4 (inclusive).
df.iloc[:,[1,2,5]] Select columns in positions 1, 2 and 5 (first column is 0).
df.loc[df['a'] > 10, ['a','c']] Select rows meeting logical condition, and only the specific columns .
Summarize Data
df['w'].value_counts() Count number of rows with each unique value of variable
len(df) # of rows in DataFrame.
df['w'].nunique() # of distinct values in a column.
df.describe() Basic descriptive statistics for each column (or GroupBy)
pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:
sum() Sum values of each object.
count() Count non-NA/null values of each object.
median() Median value of each object.
quantile([0.25,0.75]) Quantiles of each object.
apply(function) Apply function to each object.
min() Minimum value in each object.
max() Maximum value in each object.
mean() Mean value of each object.
var() Variance of each object.
std() Standard deviation of each object.
Group Data
df.groupby(by="col") Return a GroupBy object, grouped by values in column named "col".
df.groupby(level="ind") Return a GroupBy object, grouped by values in index level named "ind".
All of the summary functions listed above can be applied to a group. Additional GroupBy functions:
size() Size of each group.
agg(function) Aggregate group using function.
Handling Missing Data
df.dropna() Drop rows with any column having NA/null data.
df.fillna(value) Replace all NA/null data with value.
Make New Columns
df.assign(Area=lambda df: df.Length*df.Height) Compute and append one or more new columns.
df['Volume'] = df.Length*df.Height*df.Depth Add single column.
pd.qcut(df.col, n, labels=False) Bin column into n buckets.
pandas provides a large set of vector functions that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:
max(axis=1) Element-wise max.
min(axis=1) Element-wise min.
clip(lower=-10,upper=10) Trim values at input thresholds
abs() Absolute value.
The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.
shift(1) Copy with values shifted by 1.
shift(-1) Copy with values lagged by 1.
rank(method='dense') Ranks with no gaps.
cumsum() Cumulative sum.
rank(method='min') Ranks. Ties get min rank.
cummax() Cumulative max.
rank(pct=True) Ranks rescaled to interval [0, 1].
cummin() Cumulative min.
rank(method='first') Ranks. Ties go to first value.
cumprod() Cumulative product.
Windows
df.expanding() Return an Expanding object allowing summary functions to be applied cumulatively.
df.rolling(n) Return a Rolling object allowing summary functions to be applied to windows of length n.
Plotting
df.plot.hist() Histogram for each column
df.plot.scatter(x='w',y='h') Scatter chart using pairs of points
Combine Data Sets
adf
bdf
Standard Joins
pd.merge(adf, bdf, how='left', on='x1') Join matching rows from bdf to adf.
pd.merge(adf, bdf, how='right', on='x1') Join matching rows from adf to bdf.
pd.merge(adf, bdf, how='inner', on='x1') Join data. Retain only rows in both sets.
pd.merge(adf, bdf, how='outer', on='x1') Join data. Retain all values, all rows.
Filtering Joins
adf[adf.x1.isin(bdf.x1)] All rows in adf that have a match in bdf.
adf[~adf.x1.isin(bdf.x1)] All rows in adf that do not have a match in bdf.
ydf
zdf
Set-like Operations
pd.merge(ydf, zdf) Rows that appear in both ydf and zdf (Intersection).
pd.merge(ydf, zdf, how='outer') Rows that appear in either or both ydf and zdf (Union).
pd.merge(ydf, zdf, how='outer', indicator=True) .query('_merge == "left_only"') .drop(['_merge'],axis=1) Rows that appear in ydf but not zdf (Setdiff).

⑤ 数据维度

我们从静态数据和动态行为数据俩个角度来构建立体多维群体行为画像：

静态数据

注册信息画像：时间，设备，ip，渠道

登录信息画像：时间，设备，ip，版本

动态行为数据

假设 A、B、C 三个账号由同一个人控制，那么每次有活动他们就一起来了，可知道他们是一伙的，如何把这三个人每次活动一起来这件事儿，通过数据来描述好加工出来，供后面的模型使用，这是我们当时研究的重点。

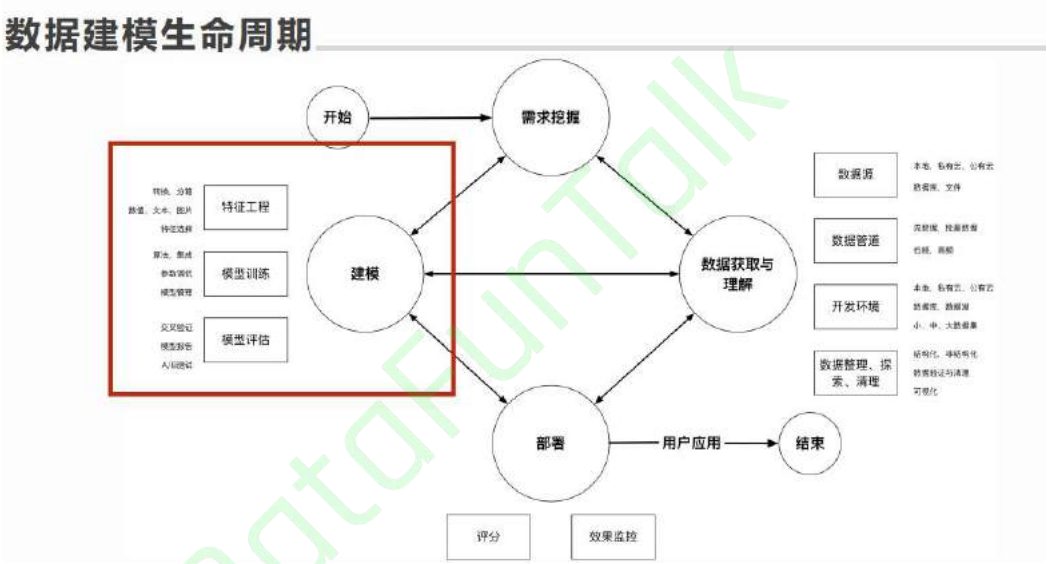
动态行为数据包括：

浏览行为画像：时间，设备，ip，渠道（同静态相同，但是是账号之间的比较）

下单行为画像：商品信息，时间信息，设备，ip，地址，优惠券，支付

领券行为画像：频率，使用周期

建模



建模的三个主要模块：

特征工程：特征处理，数值，文本，图片，特征选择

模型训练：参数调优，模型管理

模型评估：交叉验证，模型报告，A/B 测试

① 目标

- 确定要输入模型的数据特征
- 选择一个最能满足项目要求的模型（如果这个项目以准确率为最核心的 KPI，可能 A、B、C 三个模型都各有所长，有的效率高、有的速度快、有的召回率高，这时就需要按要求选择）
- 模型参数调优，针对生产环境做优化（A/B TEST）

② 实现方法

- 特征工程
- 模型选择（分类、聚类、有监督、无监督、深度学习、强化学习）
- 模型训练（如何调优）
- 评估模型在生产环境的适应性

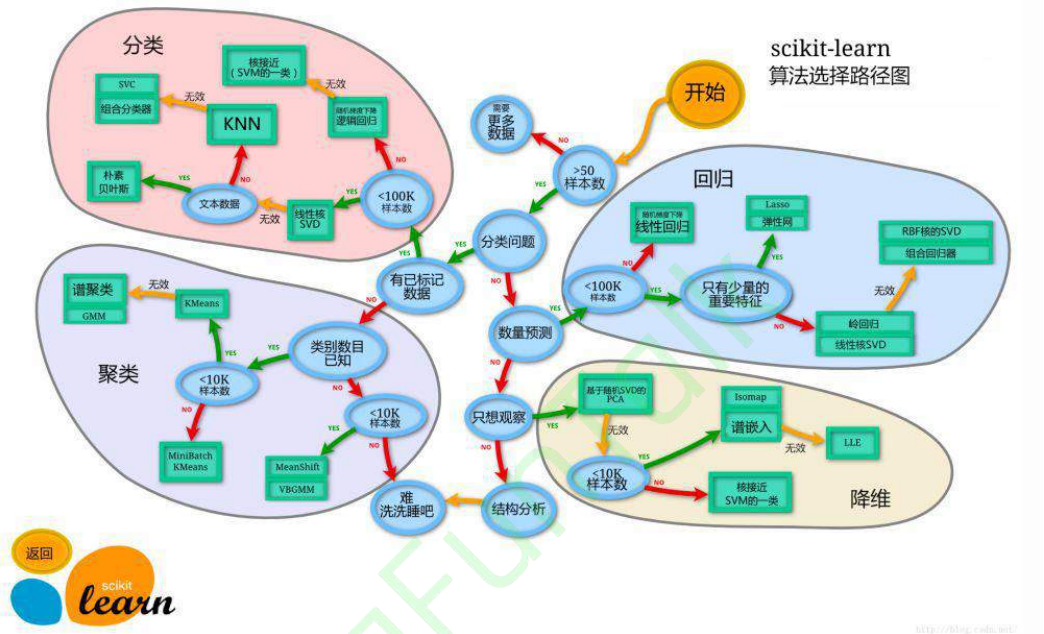
③ 交付物

特征集

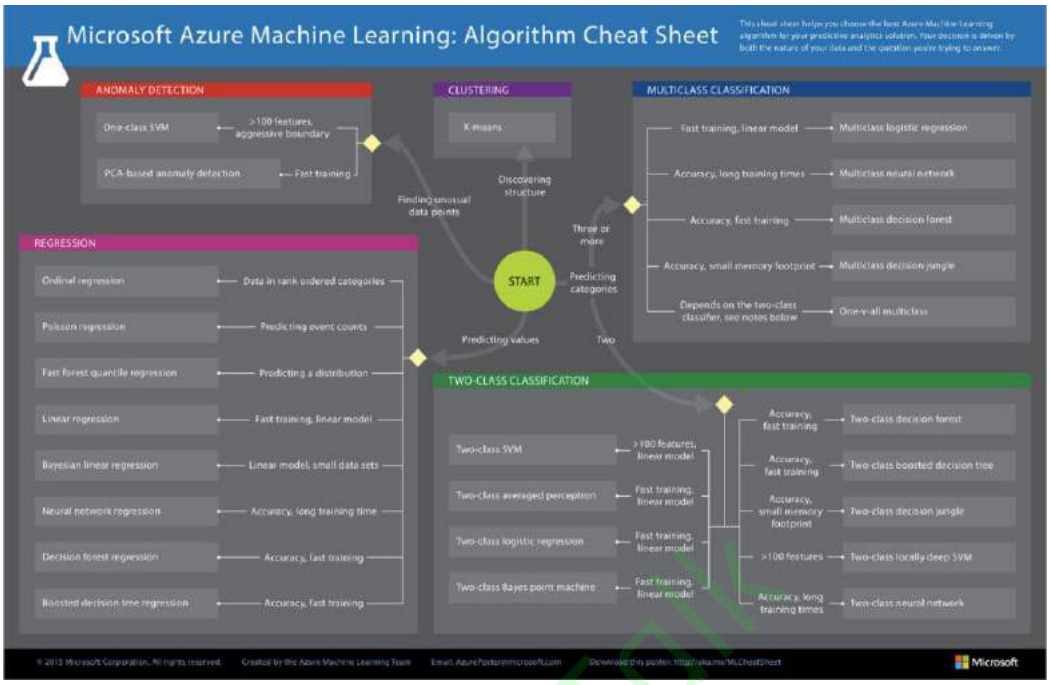
模型报告

④ 算法模型路径选择

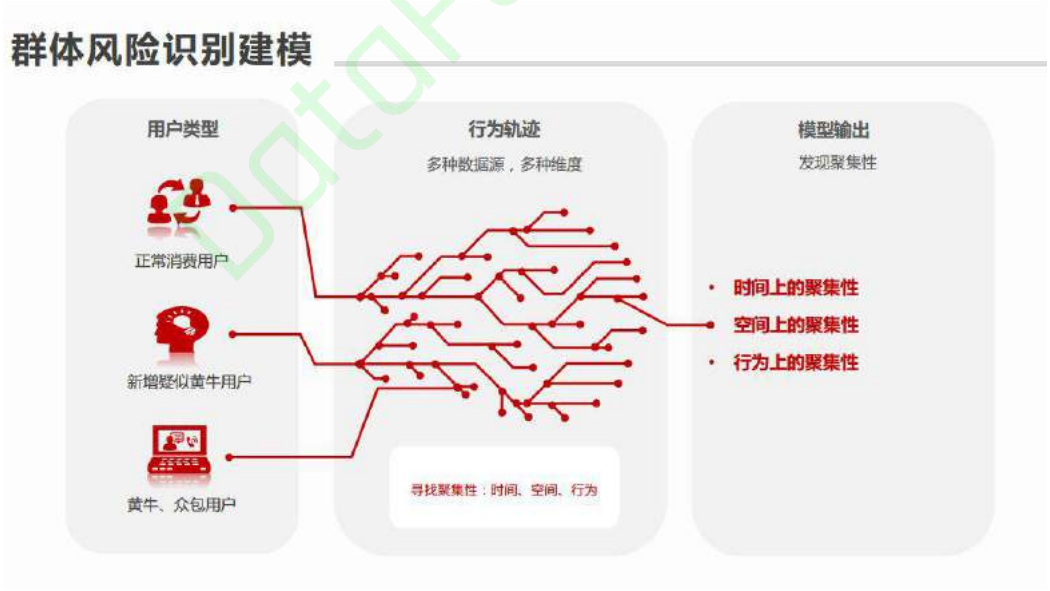
Sklearn 算法选择路径图（这是目前网上比较流行的模型选择路径，大家可以根据业务的具体需求来选择模型）；



微软的模型选择路径：



⑤ 群体风险识别建模

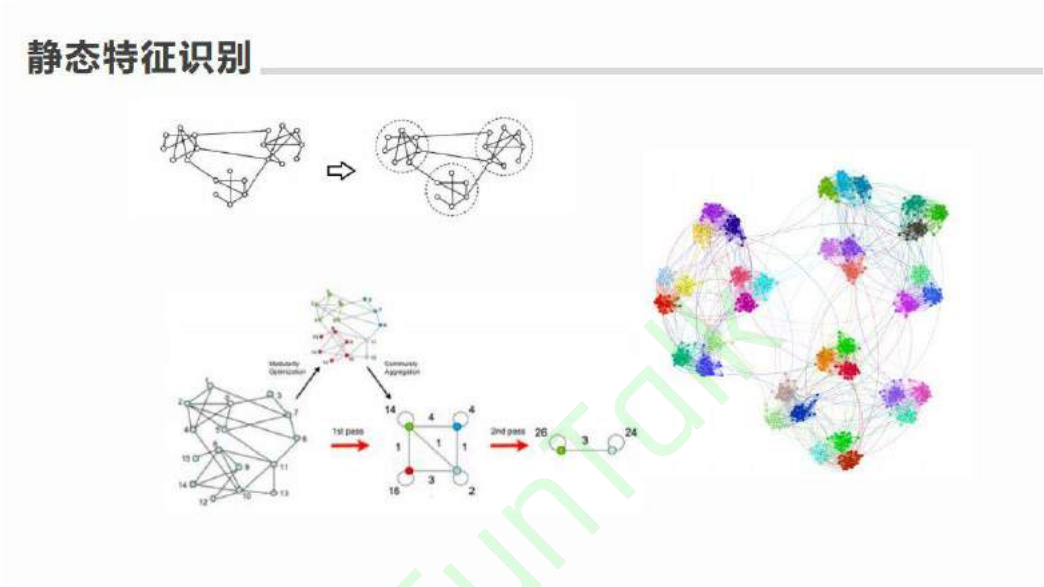


用户类型：正常消费，新增疑似黄牛用户，黄牛&众包用户

行为轨迹：多种数据源，多种维度，寻找在时间，空间，行为的聚集性

模型输出：发现时间，空间，行为上的聚集性

⑥ 静态特征识别



关系图，聚类：Louvain，Fast Greedy，Markov Cluster，Info Map，Walktrap。

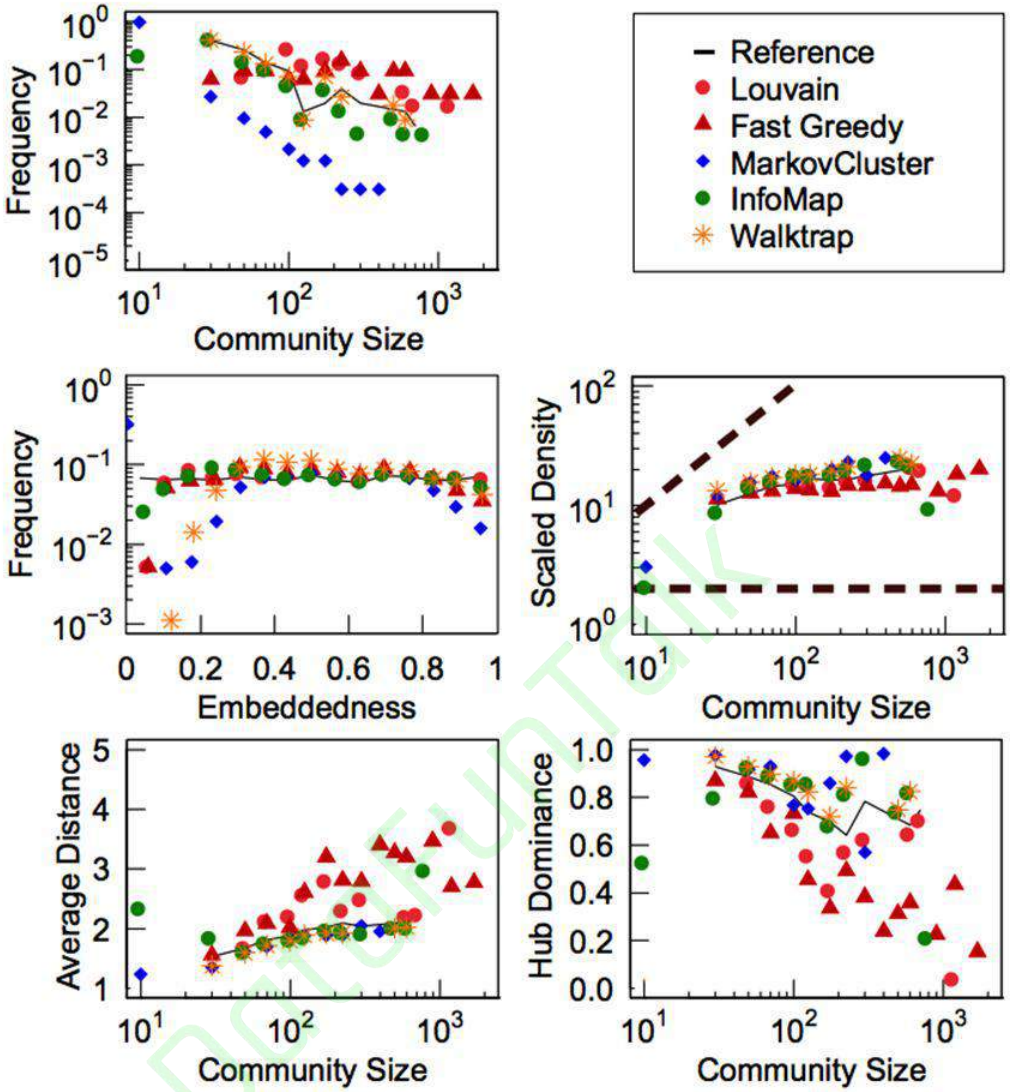
图计算的社区发现（前人的一些经验）：

Algorithm	Year	Complexity
CFinder	2005	
LPA	2007	$O(m)$
LFM	2009	$O(n^2)$
EAGLE	2009	$O(n^2 s)$
GIS	2009	$O(n^2)$
HANP	2009	$O(m)$
GCE	2010	$O(mh)$
COPRA	2010	$O(vm \log(\frac{vm}{n}))$
NMF	2010	$O(Kn^2)$
Link	2010	$O(nk_{max}^2)$
SLPA	2011	$O(Tm))$
BMLPA	2012	$O(n \log n))$

常用算法比较：

Algorithm	$n = 10^4$	$n = 10^5, k_{max} = 10^3$	$n = 10^5, k_{max} = 3 \times 10^3$
Louvain	0.80	0.78	0.80
Fast Greedy	0.59	0.66	0.67
MarkovCluster	0.83	0.87	0.80
InfoMap	0.88	0.93	0.91
Walktrap	0.77	0.79	0.78

DataFunTalk


Frequency
Community Size
Reference
Louvain
Fast Greedy
MarkovCluster
InfoMap
Walktrap
Frequency
Embeddedness
Scaled Density
Community Size
Average Distance
Community Size
Hub Dominance
Community Size

⑦ 动态特征识别

动态特征识别

时间	活动场景	下单账号
2019-05-21 11：00-12：00	A品类B商品C优惠	pin1,pin2,pin3,pin4,pin5,pin6,pin7
2019-05-21 12：00-13：00	A品类B商品C优惠	pin0,pinA,pinB,pinC,pinD
2019-05-21 13：00-14：00	A品类B商品C优惠	pinF,pinE,pinH,pinG,pinB,pinM,pinL,pinX
2019-05-22 08：00-09：00	X品类Y商品Z优惠	pin3,pinE,pinB,pinH
2019-05-22 09：00-10：00	X品类Y商品Z优惠	pin1, pinC,pinX,pinV,pinZ,pinR,pinU,pinI
2019-05-23 00：00-01：30	M品类L商品N优惠	pin4,pinC,pin3,pinX,pinD,pin9,pin4,pinF
2019-05-23 01：30-02：30	M品类L商品N优惠	pin23,pin5,pinG,pinH,pinJ
.....	.....	.....

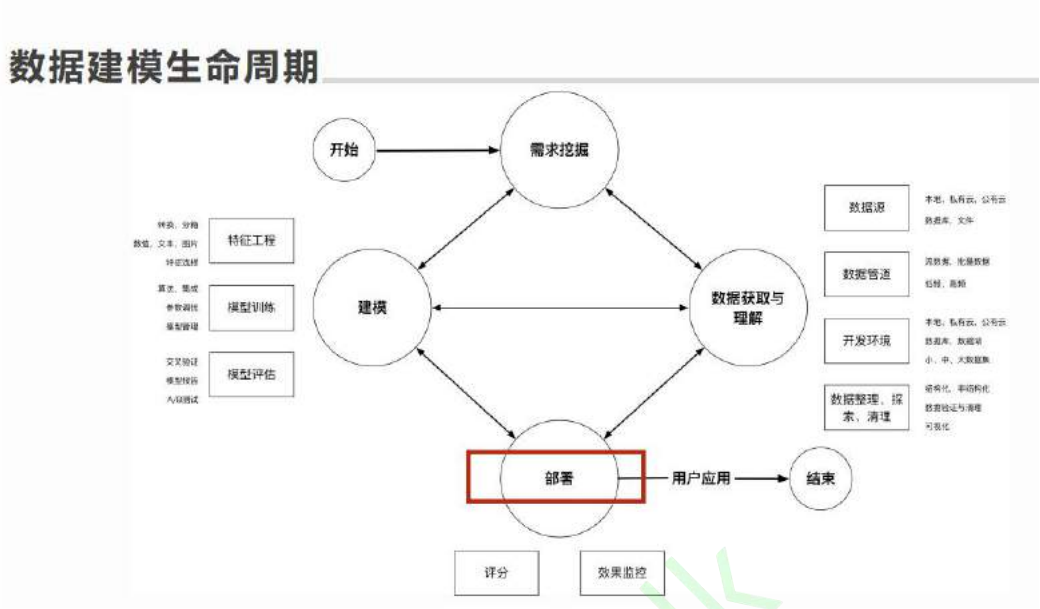
这是我们归结出来的数据底层形式，包括：时间，活动场景，下单账号几个维度。

群体划分(聚类问题) ：距离的选择对聚类的结果影响巨大

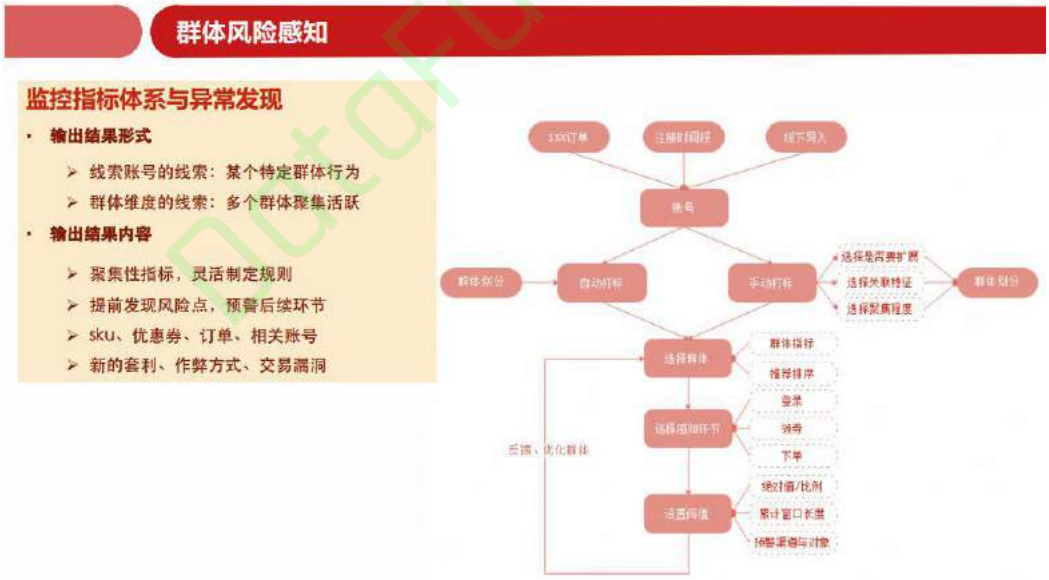
动态特征识别（群体划分）

算法名称	算法参数	范围	距离度量
K-means	簇数	大样本、聚类数目中等	点之间的距离公式
spectral clastering	簇数	样本数中等、聚类数目小	图距离
ward hierarchical clustering	簇数	大样本、聚类数目大	点之间的距离
agglomerative clustering	簇数、链接类型、距离	大样本、聚类数目大	任意成对点线图之间的距离
DBSCAN	半径大小、最低成员数	样本数中等、聚类数目中等	最近点之间的距离
birch	分支因子、阈值、可选全局集群	大样本、聚类数目大	点之间的欧式距离

部署



部署的时候需要考虑的两个关键因素：**评分和效果监控。**



目标：把模型和配套的数据管道部署到生产环境

实现方法：**推荐 api 接口形式部署模型**

交付物：监控报表，模型说明文档（输出和部署细节），数据管道说明文档

用户应用

① 目标

完成项目交付

② 实现方法

- 验证部署好的系统的功能性和准确率，满足业务需求
- 所有文档都要完成并进行审阅
- 项目交接工作可能是交给 IT 运维部分或者是用户的数据团队甚至可能是业务团队

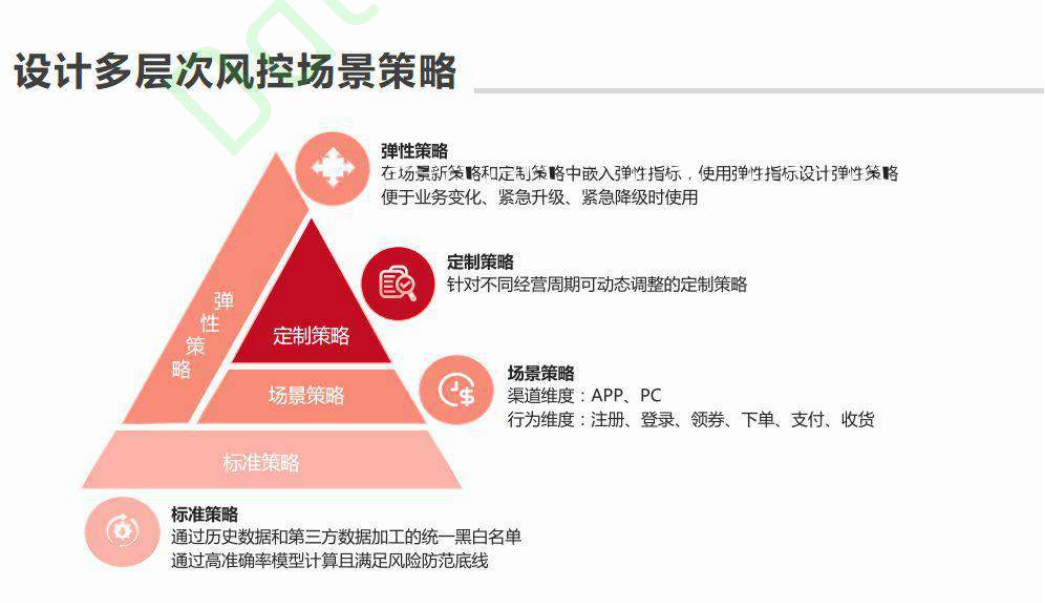
③ 交付物

项目交付报告

④ 业务应用架构：

用规则引擎（实时计算、多重规则、实时指标），模型引擎（模型管理、模型部署、模型运算），决策引擎（策略管理、策略可视化、策略路由）连接数据中心和用户行为。

⑤ 设计多层次风控场景策略



标准策略：标配，黑白名单，评分

定制策略：针对不同的经营周期动态调整策略

场景策略：针对渠道，行为维度

弹性策略：场景策略和定制策略中增加弹性指标，用于业务变化，紧急升级和降级

今天的分享就到这里，谢谢大家。

DataFunTalk

信贷业务风控策略简介

嘉宾：韩士渊　编辑：张祥

导读：大家好，今天分享的主题是信贷业务风控策略。风控业务主要经历了几个阶段：

规则：直接判断通过，或不通过。

数据：可以通过客户的资产，流水，来判断客户的资质优劣。

模型：通过数据分析、数据挖掘，找到相应的规律，识别出人工难以找到的部分人群。

但是，数据是有限的，成本很高，会限制风控的上限；同时，如何有效的结合数据、规则、模型，来实现业务目标，这就需要风控策略来完成。

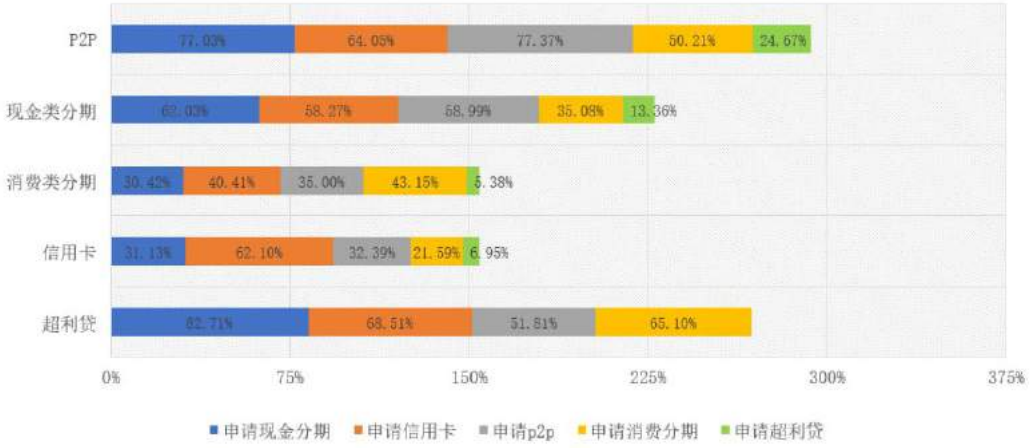
本次分享，将介绍如何在信贷业务中利用数据、规则、模型等完善风控策略，包括原有风控流程及规则优化、定价策略、额度策略等内容。

背景

1.1 消费信贷行业背景

首先给大家普及下消费信贷行业的情况，目前的消费信贷市场和 2 年前的不太一样。2 年甚至 3 年以前，它是一个蓬勃发展的阶段，但是现在出现了一个情况，就是消费降级，社会消费品零售总额、汽车零售总额增长放缓，居民狭义的负债率攀升，以及银行不良贷款率增长态势。信贷市场相较前几年的市场发生了变化，前几年属于资产红利，风控的价值本质上是资产获利的多少；目前而言，风控的价值是有效降低风险，减少亏损。因此，对于成本的控制，对于风控的效果，需要有更精细化的应用和实践。

1.2 消费信贷行业客户渗透



横轴表示渗透占比，横轴越长，渗透率越高；单个颜色越长，表明用户群重合也越高。当某个产品受到监管，会导致其客户流失，随之带来金融风险问题。以 P2P 和超利贷为例，正常情况下，P2P 的客户应该比超利贷好，但是由于市场监管的变化，二者的客户质量已经差不多了。所以，当市场规则发生变化的时候，原来的规则和模型都失效，因此，风控策略就成为了最后一道关卡。

1.3 传统评分卡的开发流程

① 目标定义

定义风控业务的目标，要求熟悉业务逻辑，好坏客户的定义。

② 数据的整合加工

数据：包括用户的姓名、身份证号、手机号、银行卡号，购物记录等

清洗：错误值、缺失值、离群值

衍生变量的处理：引入其它维度的数据，以及做特征组合

③ 特征选择、调优、效果评估

特征选择：利用统计显著性、变量重要度、IV 指标，变量聚类等算法来挑选重要的变量，通常还需要采用分箱的方式，将连续值离散化，增加变量的鲁棒性，增加了变量的整体稳定性。

模型调优：在训练集上调整模型参数，对比 KS 值的变化，选择 KS 最大的模型。

效果评估：对比训练集、测试集、样本外数据集的 KS 值，查看模型是否稳定。

共线性：此外，还需要检验模型变量之间是否存在共线性问题，一般计算 VIF 值，如果 VIF > 5，表明存在共线性问题，该变量不可用。

模型稳定性：计算 PSI 值，如果 PSI < 0.1，则模型稳定性较好，如果 PSI>0.2，则模型稳定性较差，不可用。

④ 模型打分、部署、监控

对变量每个分箱都打一个分值，形成一张评分卡（相较于机器学习，优点是可解释性强），最后上线部署。

1.4 机器学习模型开发流程

相较于评分卡开发流程，差异如下：

① 机器学习中人工介入少；

② 可解释性差；

③ 重点是调整参数，避免模型过拟合。

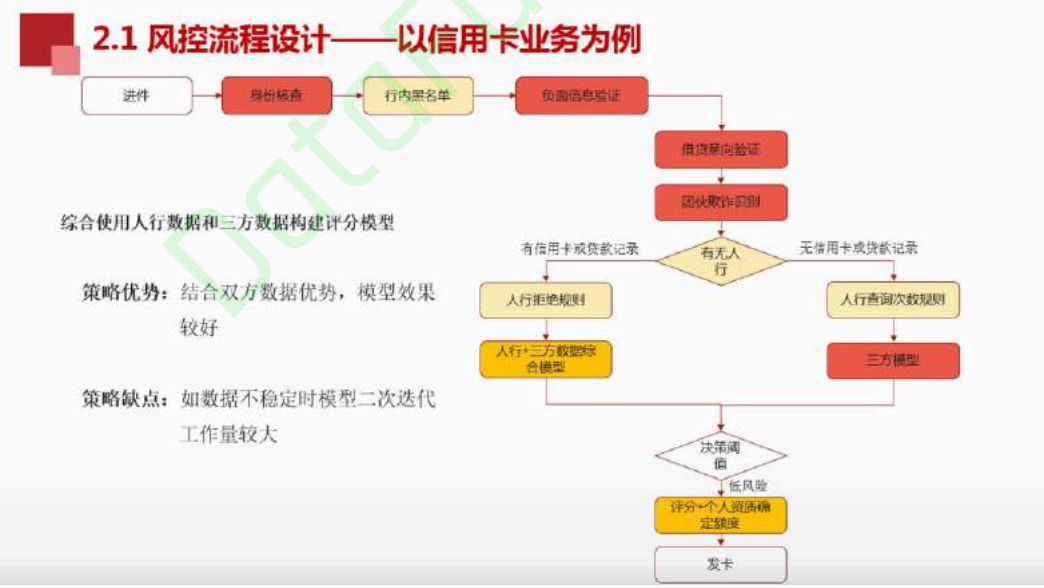
贷前风控流程与策略

2.1 风控流程设计

目标是发现风险点（包括：信用卡欺诈、团伙窝案、高危用户等），降低风险；同时降低成本、提升效率。

银行的风控流程，以某四大行信用卡业务为例。

案例 1：



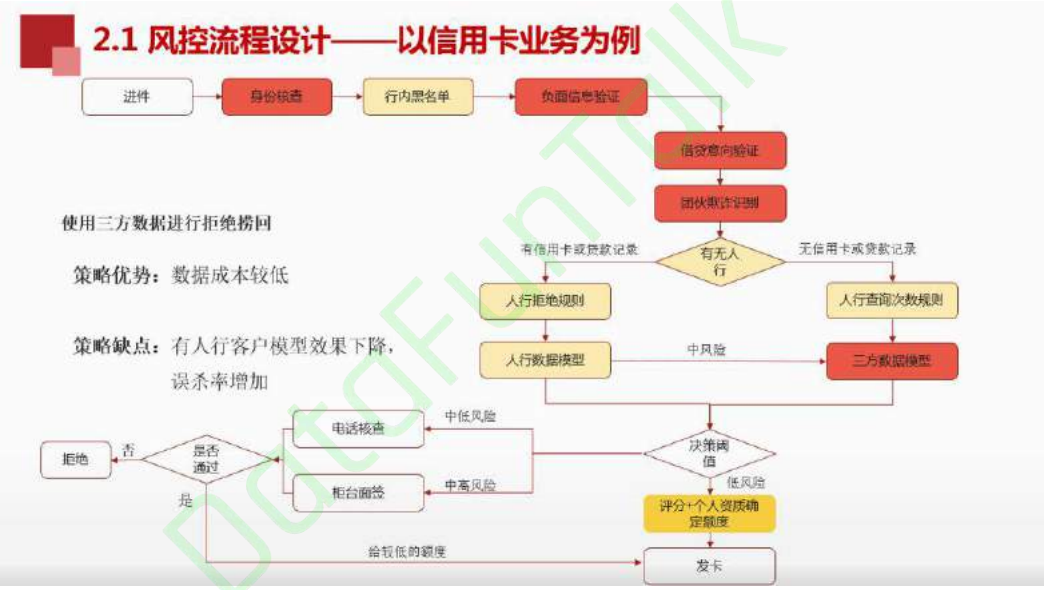
① 身份核查：验证身份的真实性，是否为本人、是否有欺诈等行为。

② 剔除其它点影响：行内黑名单、负面信息验证。

③ 将成本低的借贷意向验证放在成本高的团伙欺诈识别前面，达到节省成本的作用。

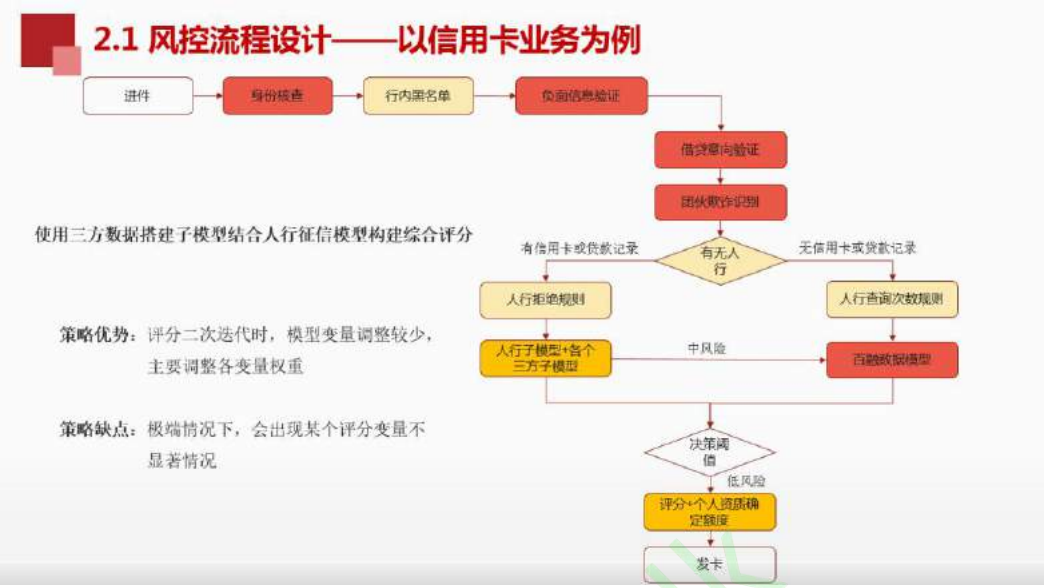
④ 结合人行拒绝规则、人行数据以及第三方数据建模，其优点是结合双方数据优势，模型效果会较好。但数据不稳定时，模型二次迭代工作量较大。

案例 2：



相较于案例 1，案例 2 较为保守，没有将人行数据和三方数据结合起来做来模型，对于拒绝的用户再次做人工审核，将符合的用户再次捞起来。这样做数据成本低，但是有人行客户模型效果下降，误杀率增加。

案例 3：



同案例 1 的区别：将各个数据构建子模型，然后整合为一个综合模型，如收入数据、支出数据建模。差异在于客群建模的差异，将不同的客群用户分开做模型。这样在评分二次迭代时，模型变量调整较少，主要调整各变量权重即可。但是在极端情况下，会出现某个评分变量不显著的情况。

综合上述三个案例，风控策略并不是完全依赖于成本，还要依赖于实际业务情况和业务目标。很多时候，风控流程是根据业务情况来进行的。

2.2 利率策略和额度策略

2.2 利率策略&额度策略

在完成模型构建后，对每位客户进行打分，统计每个分段内的坏账率，可得到以下决策表：

评分区间	通过率	人数	区间坏账率（pi）
<551	100%	3000	28.50%
[551,582)	95%	3000	17.72%
[582,601)	90%	3000	13.23%
[601,617)	85%	3000	10.67%
[617,631)	80%	3000	8.76%
[631,644)	75%	3000	7.08%
[644,655)	70%	3000	6.93%
[655,667)	65%	3000	6.18%
[667,679)	60%	3000	5.04%
[679,691)	55%	3000	4.71%
[691,701)	50%	3000	3.69%
[701,717)	45%	3000	3.36%
[717,730)	40%	3000	3.03%
[730,745)	35%	3000	2.46%
[745,760)	30%	3000	2.28%
[760,777)	25%	3000	1.80%
[777,794)	20%	3000	1.44%
[794,818)	15%	3000	1.14%
[818,849)	10%	3000	0.90%
>=849	5%	3000	0.51%

完成模型构建后，对每位客户打分，统计每个分段内的坏账率，以控制收益与成本。怎么给合适的利率和额度呢？

① 利率策略：

假设：

- 预期收益率：r_0
- 利率在区间中变化时，不同分段的坏客户率变化不显著
- 评分有较好的排序能力，每一分段对应的坏客户占比有显著差异，第i段评分区间对应的违约率为p_i

可能的收益-可能的损失=预期收益

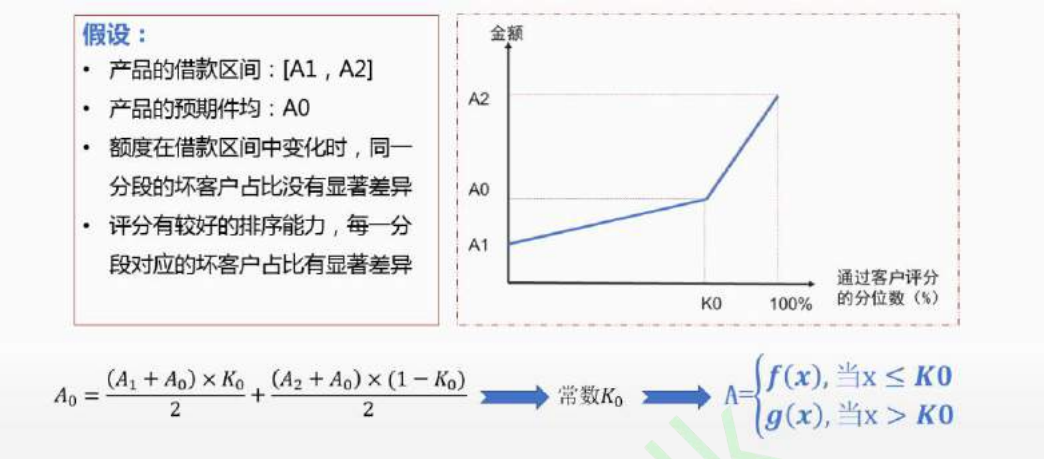
$$A_0 \times r \times (1-p) - A_0 \times p = A_0 \times r_0$$

$$A_i \times r_i \times (1-p_i) - A_i \times p_i = A_i \times r_0$$

$$r_i = \frac{r_0 + p_i}{1 - p_i}$$

A 表示额度，r 表示预期收益率，p 表示坏账率，对每个评分段分别计算预期收益 r_i，但通常情况下，利率是固定的，当分数在某个阈值时，就直接拒绝掉。

② 额度策略：



额度策略本身受限于产品设计、客户需求及竞品情况，结合自身成本和风险偏好，可初步确定产品的额度区间 [A_1，A_2] 和件均 A_0。由于右图中俩个梯形的面积应该是相等的，因此，可以得到关于 A_0 的计算表达式，由于 A_0、A_1、A_2 都是已知的，因此我们能计算出 A_0 对应的常数 K_0，这样就可以把右图中蓝色的折现拟合出来，即相对最优的一个解。可以实现，根据不同的分位数，给不同的额度。那么，这么做合理吗？它是需要满足一定的假设条件的：

额度在借款区间中变化时，同一分段的坏客户占比没有明显差异；

评分有较好的排序能力，每一分段对应的坏客户占比有显著差异。

③ 额度策略优化：

基础风险额度	月收入（千）	0	1	2	3	4	5	6	7	8	9	...
	年收入（万）	0	1.2	2.4	3.6	4.8	6	7.2	8.4	9.6	10.8	...
1.5	[600,630]	1.5	1.2	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	...
3.1	[630,660]	1.9	1.9	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	...
5.4	[660,690]	3.3	3.2	3.9	4.7	5.4	5.4	5.4	5.4	5.4	5.4	...
8.6	[690,720]	5.3	5.2	5.7	6.3	6.9	7.4	8	8.6	8.6	8.6	...
12.8	[720,750]	7.7	7.7	8.3	8.8	9.4	10	10.5	11.1	11.7	12.2	...
17.2	[750,780]	10.3	10.3	10.9	11.4	11.9	12.4	13	13.5	14	14.6	...
21.1	[780,810]	12.7	12.7	13.2	13.7	14.3	14.8	15.3	15.9	16.4	16.9	...
24.6	[810,840]	14.8	14.8	15.3	15.8	16.3	16.9	17.4	17.9	18.4	18.9	...
27.4	[840,870]	16.5	16.5	17	17.5	18	18.5	19.1	19.6	20.1	20.6	...
29.2	[870,900]	17.5	17.5	18	18.5	19	19.5	20	20.5	21.1	21.6	...
29.9	[900,1000]	17.9	17.9	18.5	19	19.5	20	20.5	21	21.6	22.1	...

采用 sigmoid 来替代分段函数，确定基础风险额度。对于大额借贷，还是考虑用户的偿债能力，即收入，资产，流水等指标，先算出基础风险额度，再结合收入等指标，差异化调整基础额度。

2.3 风控规则有效性诊断

怎么确认规则的有效性和调整？

首先对拒绝客户进行打分，然后对比‘通过客户评分分布’和‘拒绝客户评分分布’，找出异常规则，再把各规则进行分组评分分布对比，最后进行规则调整。

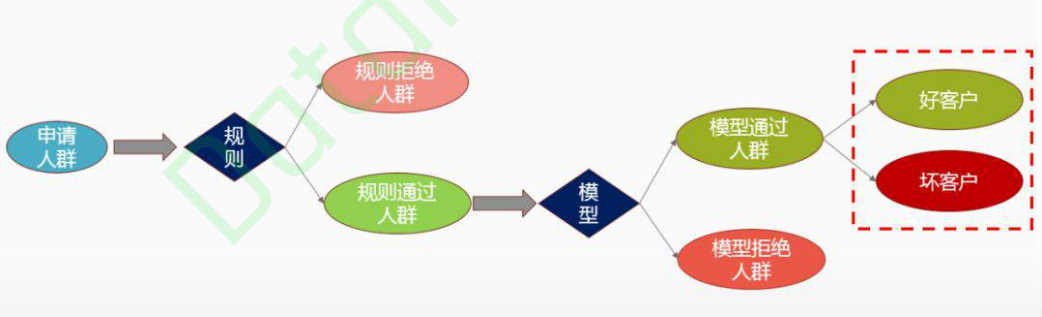
如上图，对拒绝的客户重新打分，与整体样本中拒绝用户评分对比：

规则 1，效果最好，最低分段的用户拒绝率高，高分段的用户拒绝率。

规则 2，效果不明显，与整体数据的分布差异不大，效果不明显。

规则 3，太过于波动，不稳定。

2.4 信用评估模型的构建与优化——模型优化



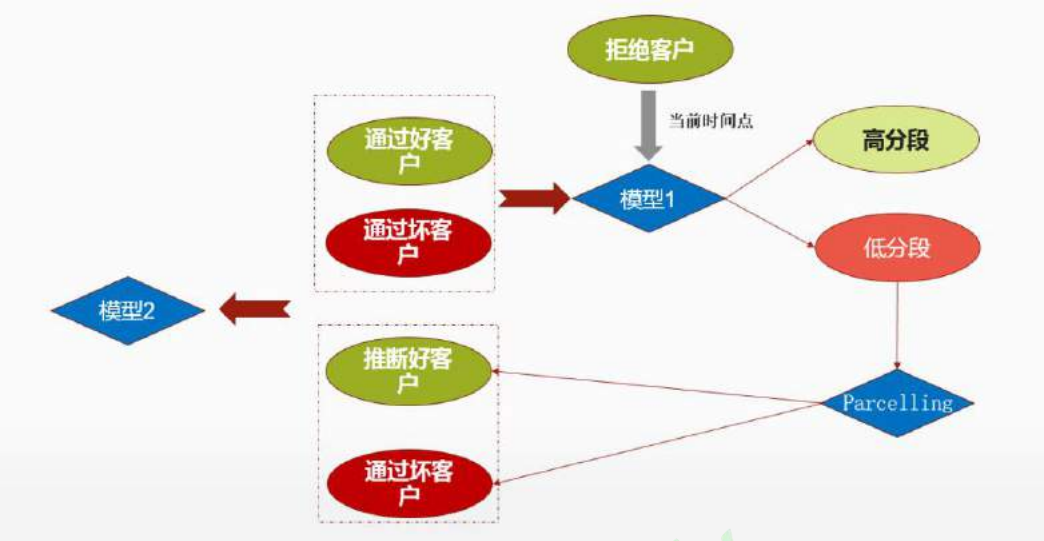
基础的风控建模流程如上。在进行模型迭代时，主要利用通过样本进行模型构建，被迭代的模型拒绝率较高；如不考虑模型拒绝人群，进行拒绝推断，则新模型的应用效果会有明显下滑，且多次迭代后，新模型的效果提升比例会越来越小。因此重要的一点在于，如何做拒绝推断，找到之前被规则淹没的特征。

为了解决样本淹没问题，将规则拒绝的样本加入模型训练阶段，有三种方案：

① 比例分配将拒绝对象随机划分为“好”和“差”的账户，再次带入评分流程中，构建一个模型；

② 简单增强法:对拒绝客户打分,并选取某个 cutoff 点进行区分,cutoff 点的两边分别为“好”和“坏”客户，代入模型中迭代；

③ Parcelling，是结合比例分配和简单增强法，对拒绝客户打分，对每个分段按照比例进行好坏客户划分，再次代入模型进行迭代。



我们的做法是使用通过的样本和通过的坏客户构建模型 1 ,拿当前时间点跑客户评分，分为高分段和低分段，把原来拒绝客户跑完模型后高分段的样本剔除掉，对低分段特征做 parcelling，然后推断好客户和坏客户，合在一起再做模型 2，然后不断的迭代这个模型，这时跑出来的模型，要比刚刚介绍的模型 1 方法要好。但是这种决绝推断是没办法从样本上解决样本有偏的问题，只能从某种程度上解决。

以上就是今天要分享的内容，谢谢大家。

度小满智能获客系统实践

嘉宾：费浩峻　编辑：李婷婷

大家好，今天分享的主题是：度小满金融信贷业务的智能获客系统实践。根据过去两年的工作经验，将其总结沉淀为渠道、效率、数据理解和度小满智能获客系统这四个环节，其中前三个环节是对获客的理解，渠道指明了获客方向，效率好比获客的发动机，数据则是获客的燃料。

渠道

1. 都有哪些渠道？

获客时首先会思考这样的问题：获客的体量有多大？获客的人群是什么样的？获客的渠道有哪些？获客的成本是多少等等。实际工作中，我们应先充分开发公司内部数据，逐步接入外部数据。常见的获客渠道有：直接触达，竞价广告、流量合作和线下渠道。

直接触达，依托公司内部数据，具有高转化率，低接入成本，高可操性，但受限于客户规模；

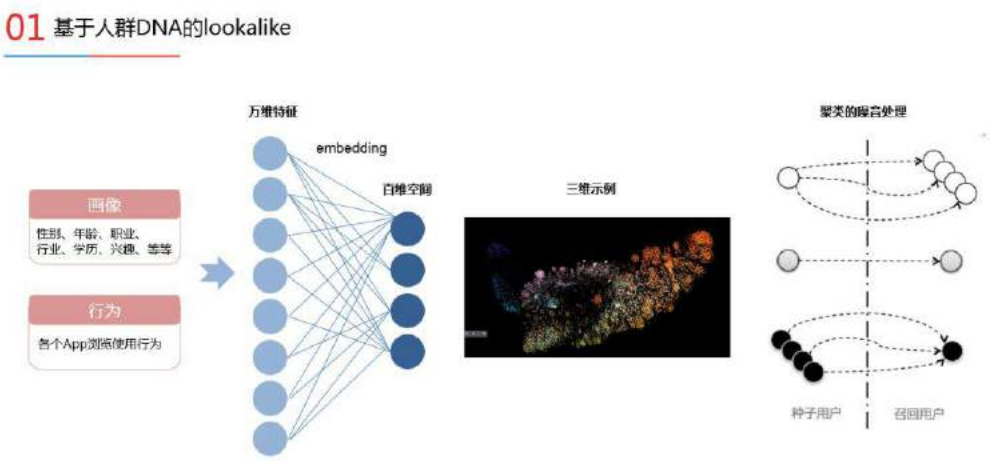
竞价广告，公司外部主要获客渠道，克服了客户规模弊端，但是市场竞争激烈易产生恶劣竞争、市场变动快、影响大，成本高；

流量合作，是一种较为稳妥的方式，客户规模大，增强公司壁垒，但接入成本高，可操作性弱；

线下渠道，复杂多样，各个公司都在尝试，但还未形成成熟的获客环境。

2. 如何选择渠道？

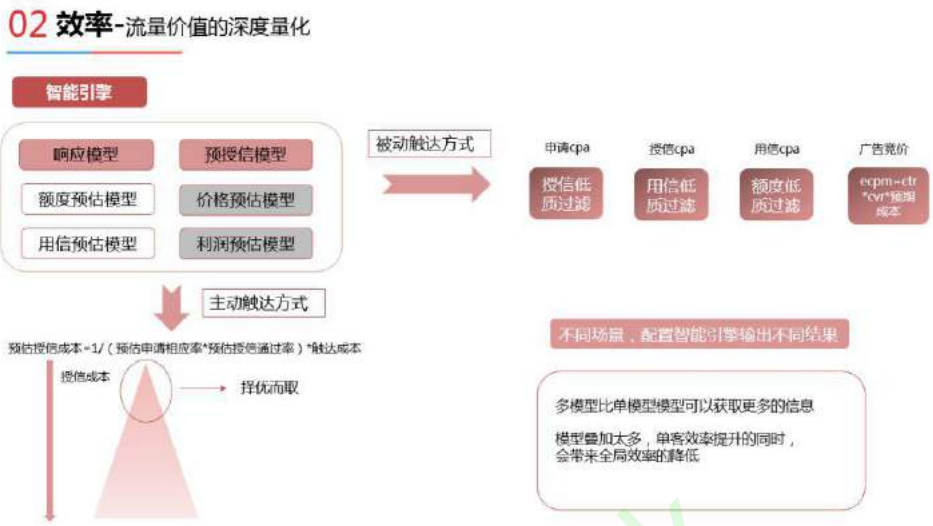
不同渠道各有利弊，如何选择渠道？从认知角度来看，产品初期，以感官认知为主，依赖产品的定位，从而确定渠道；产品发展期，分析认知逐渐重要，用户画像、用户标签帮助我们更好确定目标客户，选择合作渠道；产品成熟期，分析认知很难提高效果，转为大数据驱动的模型认知，模型可以更好地细分客户，具有高信息利用率、判别准确率和鲁棒性。



主要做法是把已知的行为和用户画像，抽象成特征，这时我们的数据集有了万维特征，然后在把特征 Embedding 成百维特征空间里，当需要投放时，会寻找目标客群，以及相似的空间向量（相似客群）进行投放。

PS：在寻找相似人群的时候，需要考虑噪音的问题，会把单目标客群到多目标客群去掉，voting 是必要的。

效率



1. 单客户的效率，如何更精准的获客

智能引擎可以提高单客户的效率，在风控模型调用前使用，它由响应模型、预授信模型、额度预估模型、价格预估模型、用信预估模型和利润预估模型组成。

以主动触达方式为例，对授信成本进行预估，将响应模型和预授信模型拟合在一起，从而得到客户授信成本的排序，在选择客户时，优先选择预授信成本低的客户进行广告投放；

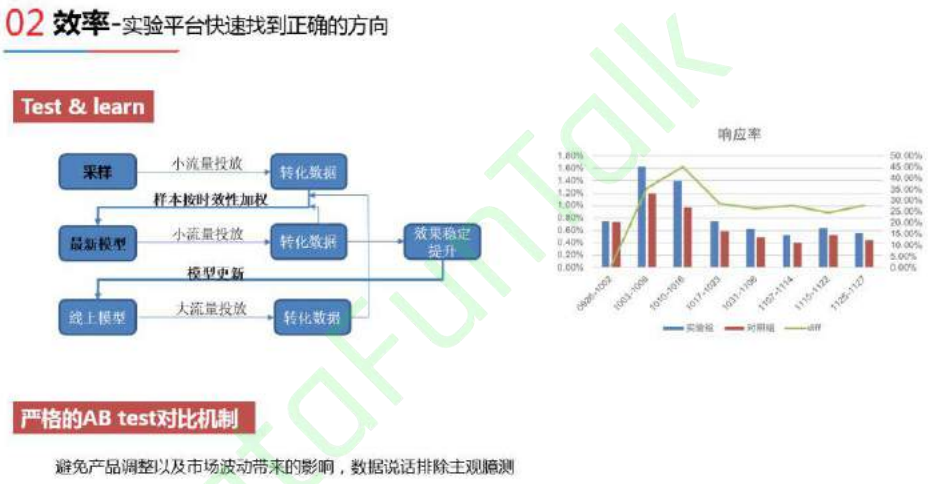
以被动触达方式为例，在授信、用信、额度、广告竞价环节中，均使用模型过滤掉质量低用户，这种方式会尽量最大化单客户效率。

可能大家会疑惑为什么需要建立多环节的模型，原因有二：

一是，不同阶段用户流失是不同的，建立多个环节的模型可以有选择的使用，确保不是一味的提升单客效率而降低了全局的效率；

二是，相同的特征在不同的模型中表现差异性很大，可以更有效使用特征。

2. 整体的效率， 模型如何快速迭代

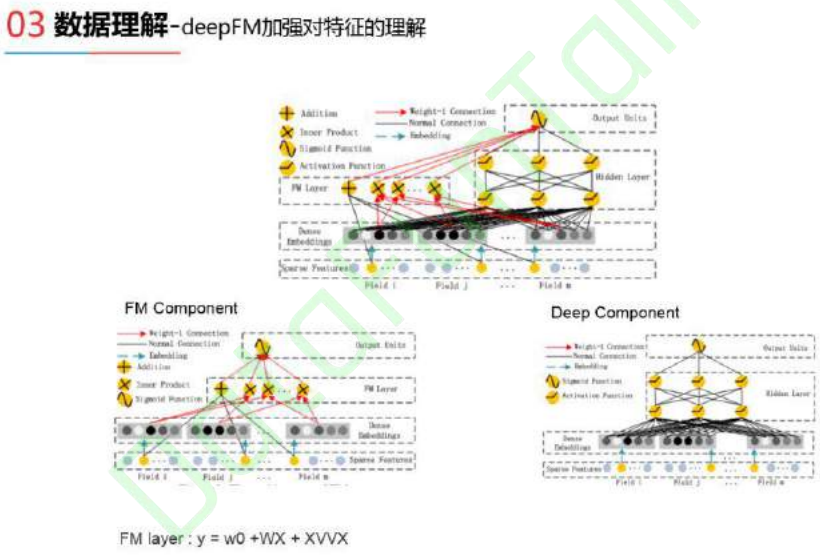


实验和模型的迭代是继续往前走的很重要的保障，以实验流程缩减流程图为例，线上同一时间运行四个模型，① 线上最优模型，② 根据新样本每天更新的模型，③ 一系列的观察模型，④ 随机采样的过程；采样过程可以做很多事情，包括 MAB 的方法或自己设定方向的探索，比如下一版模型中，新样本应占总体样本的比例，样本的时效性处理等等。模型对比应做到严格的 AB test 对比机制，避免产品调整，市场波动带来的影响，排除主观臆测。

数据理解

第三部分是讲数据，我们前面提到，数据是获客的燃料，模型做的好不好，数据理解是最重要的，数据理解主要包括两个方向：特征理解（如何更好的挖掘理解特征）和样本理解（如何理解样本是否为目标样本）。

1. 特征理解，deepFM 更好的理解特征



将不同的特征进行融合，解释这样的特征时，我们一般采用两种方式，一种是从广度上理解，如矩阵式的；另一种是从深度上的理解，如深度神经网络，梳理各个特征之间存在的隐藏关系。DeepFM 是我们去年开始调研部署上线的模型，整体模型结构是非常简洁的，DeepFM 就分两个模块，一个是 FM 模块，一个是 DNN 模块。基本上线上线下在同样的数据特征、评测集要比 XGBoost、AUC 等提升 1~2 个点。

2. 样本理解，建立全生命周期数据构造

构造全链路的转换数据是做好转换归因的第一步，以客户触达的为例，客户前后共经过 feed 流广告展示、短信触达、抖音广告展示、下载 app，搜索广告展示、电话外呼、爱奇艺会员入口展示，用户申请。很难确定哪一种获客方式获得来该客户，在做分析时，尽量避免使用受影响太大的样本。总而言之，我们需要找到用户的关键动作并对之进行分析，但营销场景可能非常复杂，我们不能将用户的关键动作归因到具体的获客渠道策略时，全生命周期的数据管理可以帮助我们解决这样的问题。

度小满智能获客平台

04 度小满智能获客平台

下面介绍度小满获客平台现在的架构，主要分为数据层、技术层、模型层、系统层、功能层和渠道。

① 从数据来讲，主要的数据来源是用户数据，第二个是广告数据、第三个是用户的生态数据；

② 在系统层，主要为智能引擎、创意引擎、广告投放平台、内容辅写平台、实验平台这五个部分：

- 智能引擎主要配置响应模型、授信预估等，使用常规的机器学习模型一般为 xgboost ，deepFM ，stacking 等；

- 创意引擎 ,指的是获客的千人千面的创意 ,实现方式有文本创意和图片创意，文本创意以 GAN 深度学习模型为主，图片创意现在做到提取图片关键因素，还未实现完全图片的自动生成；

- 广告投放平台是主要体现技术的地方，投放时间预估、投放频度等；

- 内容辅写平台以 nlp 技术为主，同时也包括文章质量的预测、热门文章的预测等；

- 实验平台主要是加速实验的进程和舆情的监控。

③ 上面介绍了智能获客平台的底层架构，主要是为了实现上层功能，也就是多目标客群的识别、千人千面创意和快速迭代反馈，从而为客户匹配合适的渠道，更适合的触达方式和触达内容。

DataFunTalk

机器学习在反欺诈中应用

嘉宾：陈德建 编辑：侯美娟

当前机器学习在金融行业已经运用非常广泛，尤其在金融业的信贷领域。在实践中，欺诈与信贷业务强相关，所以，反欺诈变成机器学习在信贷领域的一大应用。反欺诈一般会用到机器学习、深度学习以及图谱关系，其中，机器学习与深度学习多是针对个人欺诈风险，而图谱关系则多用于团伙欺诈风险的识别。因此，本次分享将介绍欺诈风险的特点以及机器学习、深度学习和图谱在实践中的应用。

关于反欺诈

（一）欺诈风险简介

在信贷领域有两类风险，一类是信用风险，一类是欺诈风险。信用风险主要是对借款人还款能力和还款意愿进行评估，而反欺诈则是对借款人的目的是否正当进行判断。

一般借款人出现信用风险，金融机构可通过风险定价和自有备付金进行防范，风险可控性较大。而当借款人在借款时便以骗贷或骗钱为目的，且金融机构未能及时识别欺诈，则会出现未能通过借款赚取利息，反而被骗走本金的情况，尤其无法识别团伙欺诈时，会在短时间内遭受非常严重的后果，金融机构面对此类风险的可控性便会非常小，所以欺诈风险是金融机构零容忍的。

（二）反欺诈生命周期简介

DataFunTalk

信贷反欺诈要从防御开始做起，所以应从用户申请到放款整个生命周期的各个阶段特点进行分析，针对各个阶段采取具有针对性的反欺诈措施。其生命周期可参见下图：

如上图，设备与网络防御是反欺诈的第一步，可通过设备和网络层面的检测，防止用户利用设备对金融机构进行欺诈；在第一层进行防御了部分欺诈用户后，再从用户行为层、业务频次层及业务事件异常层识别用户是否有欺诈行为；再经过前四层之后还会通过复杂网络对团伙欺诈进行识别。

机器学习应用

（一）反欺诈规则的缺点

反欺诈一般通过两种方式，一种是设定规则，另一种是通过算法。

规则在反欺诈实践中应用也较多，但是缺点也明显，主要表现为：1.策略性较强，命中

直接拒绝，而且黑名单本身的误伤性也较强；2.无法给出用户的欺诈风险有多大；3.未考虑用户从信用风险向欺诈风险的转移，尤其是在行业不景气时。以上缺点机器学习可以进行有效的避免，如可计算用户的欺诈概率有多大，从而采取一定的措施争取客户，而不是直接拒绝，同时也可以通过模型计算用户从信用风险转移为欺诈风险的概率，从而金融机构可及时进行风险控制与准备。

（二）机器学习有监督模型

评分卡一般运用在信用风险评估，如：A 卡（申请评分卡）和 B 卡（行为评分卡）等，反欺诈也会运用有监督学习，如评分卡（F 卡），具体如下：

一般有监督机器学习问题：

假设训练数据

X={x1,x2.....,xn},n个样本

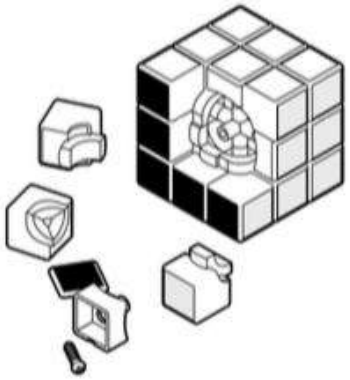
Y={y1,y2...,yn)}

损失函数(loss function)

L(F(X),Y)

目标，寻找一个F

$F^*=\arg\min_F L(Y,F(x))$



做模型的过程中，特征很重要，特征决定模型的效果。反欺诈模型需要从欺诈的角度来做特征，要注意与信用特征区分开，以免与 A 卡和 B 卡的耦合度过高。模型算出的多是概率，一般会将用模型算出的概率映射到分数，具体如下图：

评分卡分数的计算

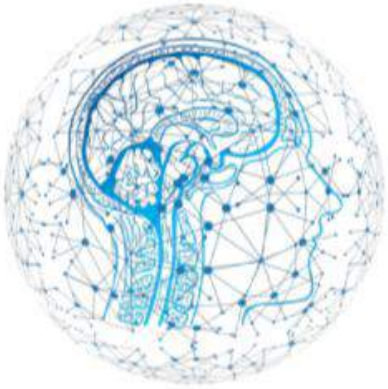
评分卡模型的最终产出是分数，且与违约概率负相关

$$score=BasePoint+\frac{PDO}{Ln(2)}(-y)$$

其中，$y=logit(p)=log(\frac{p}{1-p})$

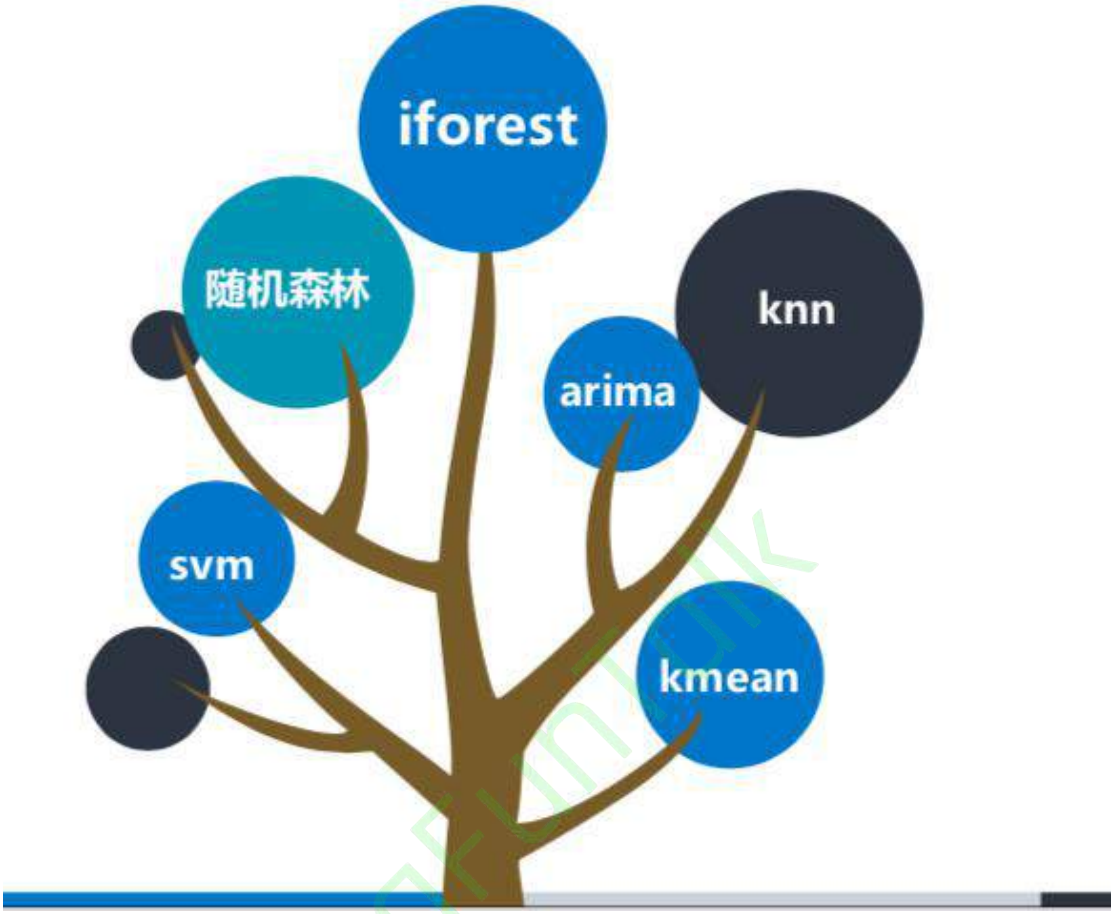
Base Point：基准分，无实际意义

PDO：Point-to-Double Odds，好坏比每升高一倍，分数升高PDO个单位



（三）部分常规机器学习在反欺诈中应用

在反欺诈中用到的机器学习主要有下图几种。其中，iforest 通常用来做数据离群点的异常检测，在应用方面，金融公司可根据自身的规则和算法，将检测出的离群点在评分卡入模的数据方面，进行加权或算法调整。svm 通常也用来做异常检测；arima 则用来作时间序列预测分析；根据现在信贷数据坏样本较少的特点，knn 和 kmean 可以用来做聚类；随机森林则是在做异常检测时进行分类，以上机器学习可通过博客进行更多了解，此处不进行深入讲解。



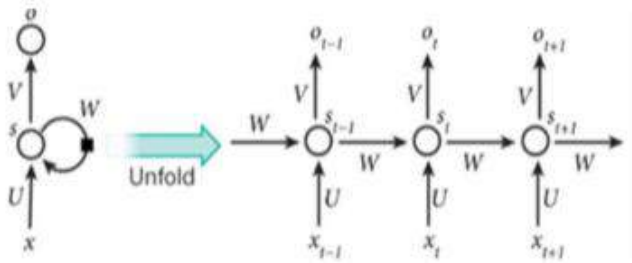
（三）深度学习应用

此部分对人工神经网络（ANN）和时间序列进行简单介绍。

神经网络通常需要大量彼此连接的神经元，每个神经元通过特定的输出函数，计算处理来自其他响铃神经元的加权输入值。神经元质检的信息传递强度，通过加权值定义，算法会不断自我学习，调整加权值。神经网络算法的核心是：计算、连接、评估、纠错和疯狂培训。

时间序列部分介绍 RNN（循环神经网络）和 LSTM（长短记忆循环神经网络）两种算法。LSTM 是 RNN 的优化版，在特征较多时，RNN 计算量会呈指数式增长，其计算复杂度也会增加，如下图：

时间序列预测分析就是利用过去一段时间内某事件时间的特征来预测未来一段时间内该事件的特征

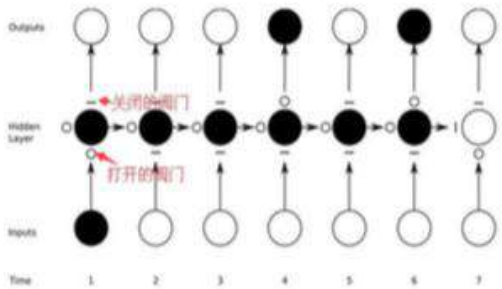


x为输入层，o为输出层，s为隐含层，而t指第几次的计算；V,W,U为权重，其中计算第t次的隐含层状态时为St = f(U*Xt + W*St-1)

特点：依赖事件先后发生的顺序　　缺陷：计算量会呈指数式增长，计算复杂度增加

St = f(U*Xt + W1*St-1 + W2*St-2 + ... + Wn*St-n)

LSTM 则是在 RNN 的结构以外加入遗忘阀门（forget gate）、输入阀门（input gate）和输出阀门（forget gate），其通过这些阀门节点实现记忆功能，改善了 RNN 在计算过程中会出现的问题，如下图：



当阀门打开的时候，前面模型的训练结果就会关联到当前的模型计算，而当阀门关闭的时候之前的计算结果就不再影响当前的计算。因此，通过调节阀门的开关我们就可以实现早期序列对最终结果的影响。而当你不希望之前结果对之后产生影响，比如自然语言处理中的开始分析新段落或新章节，那么把阀门关掉即可。

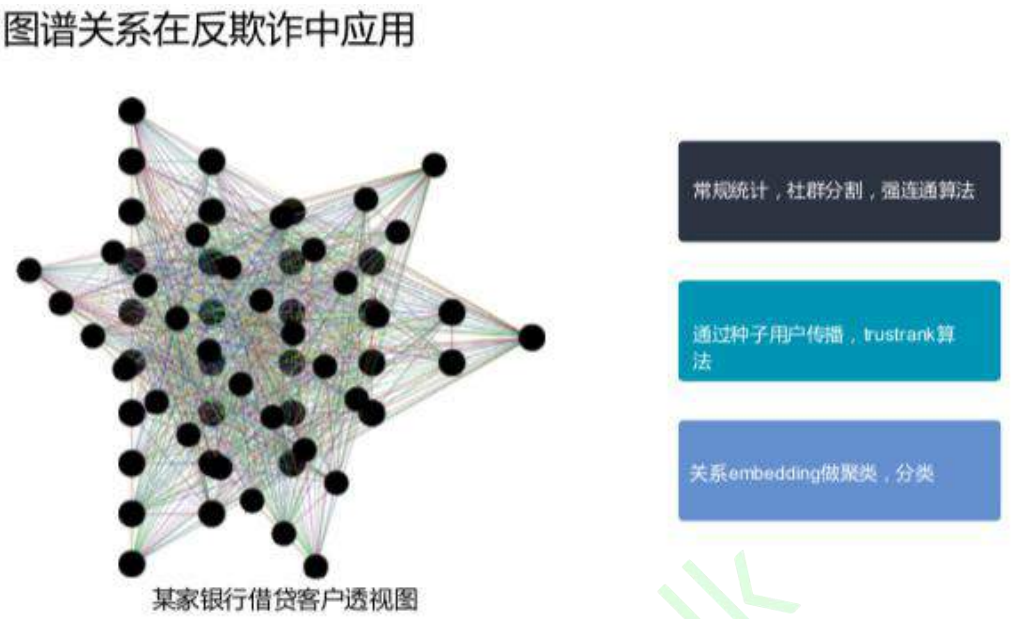
时间序列在信贷中有两个比较重要应用场景，一是 B 卡（行为评分卡），一是异常检测。

我们着重介绍 LSTM 在这两个场景中的应用。在行为评分卡的应用中，当用户在金融机构进行多次借款时，可以将其以往的借款行为通过统计方法或其他相关方法生成 embeding 进行 LSTM 计算。异常检测的应用可参考下图：

另外在使用 LSTM 时需注意 4 点：1.应限制每一个时间序列 embeding 的长度；2.对缺失数据做补 0 操作；3.针对离散变量的 embeding 尽量不要做 onehot 处理；4.样本量少时，应通过仿真模型进行异常检测评估，仿真模型能够有效解决信用风险转欺诈风险的导致模型失效的问题。

图谱相关应用

图谱主要用来防止团伙欺诈，也可以根据用户的周边关系判断用户的好坏概率。此次介绍三种图谱关系在反欺诈中的应用。



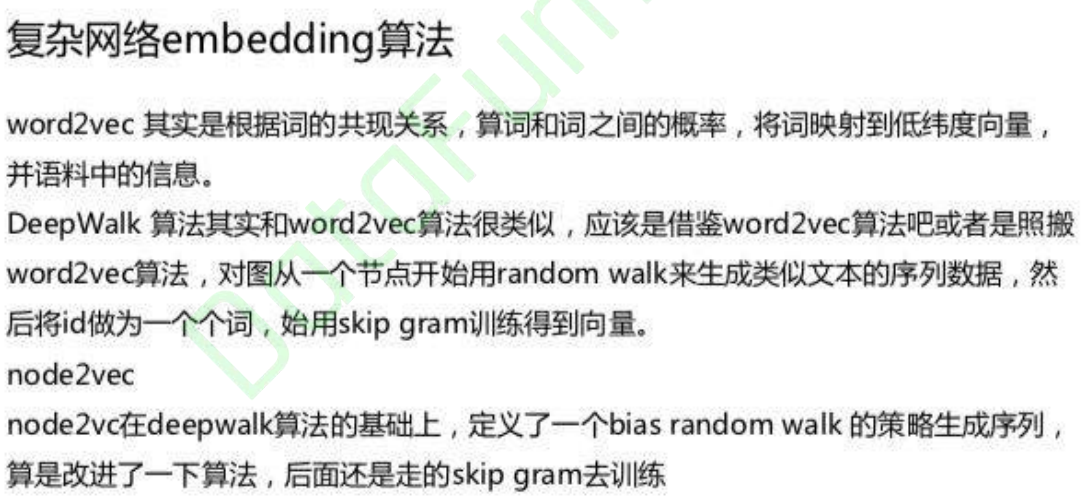
其中，常规统计一般不会直接用规则，而是将规则做成特征，再带入模型进行统计。比如一度联系人中有多少黑中介，一度联系人中的逾期人数有多少，此类特征的 KS 较高且有效。常规统计一般会用到社群分割和强连通算法。而种子传播层面则需要用到 trustrank，关系 embeding 则可以通过衍生变量将关系向量化，将向量化的关系带入模型进行进一步的分析和统计。

（一）常规统计

因为资源限制或提高投资回报率的原因，黑产一般会最大程度的利用已有资源，比如，重复的使用现有设备和信息进行多次贷款申请，这样就会出现同一手机号码、登录 IP 或硬件设备出现在多个申请信息中，形成关联网络。常规统计的运用，是通过将数据进行关联，形成关系网络图，然后使用社会关系网络分析工具，分析关系网络图中是否有大量共用设备等拓扑结构。

（二）复杂网络 embeding 算法

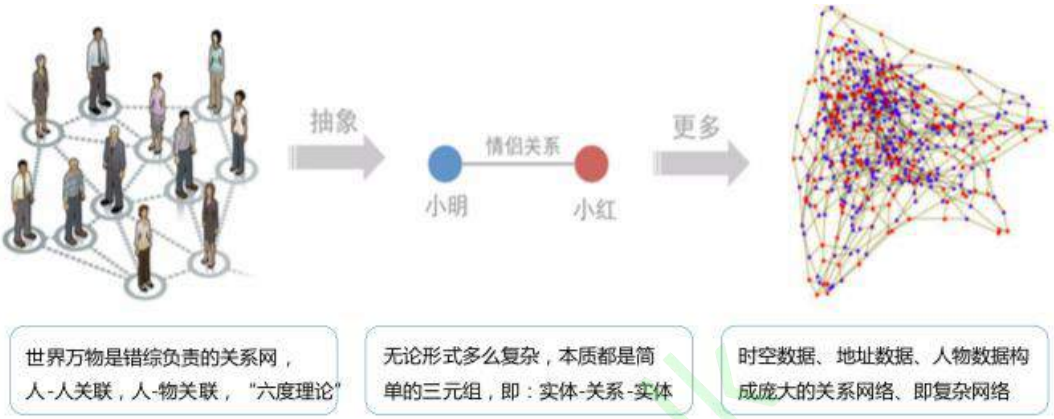
有时候机器无法识别信息，需要将信息向量化（embeding），将信息向量化后才可以做后续的算法操作。embeding 的方法有很多 ,此次仅介绍 node2vec 一种。node2vec 的原理是前端为随机游走（random walk），后端为 word2vec。random walk 则采样，将概率最大的关系采样出来并生成类似文本的序列数据，这类序列数据相当于词的共现性，对词的共线性可以做 word2vec，这里的 word2vec 与 NLP 的 word2vec 无差异。Embeding 后会生成 50 维到 128 维的向量，之后进行聚类和分类的操作，具体如下：



（三）trustrank 算法

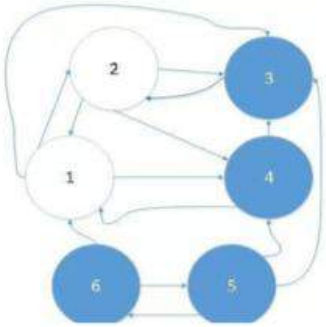
Trustrank 是 pagerank 的升级版，当前我司用的 trustrank 并不是传统的 trustrank，，而是改变其中的某些算法。trustrank 是传播关系的一种算法，根据人与人的关系进行判断和识别。比如，小红和小明是情侣关系，当小红违约时，小明的违约概率会增大，

根据类似传播关系用来做定额和模型的衍生变量。Trustrank 的使用需要建立起图谱关系，数据量小时，spark 的 sparkgragh 对 trustrank 的图谱关系支撑较好。



Trustrank 涉及种子用户（含白种子和黑种子）的定义，当前的大多使用中只有一种传播方法，也就是白种子只传播白用户，黑用户只传播黑用户。但是其实可以进行变量的衍生和算法的改进，比如一个用户既跟黑中介有联系，又跟高净值用户有联系。

以下图为例，trustrank 为种子用户（下图的 1 和 2）定义一个初始值，每次传播后会改变矩阵的值，最后收敛得到 trustrank 的分。



$$r=\begin{bmatrix}\frac{1}{2}\\ \frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{bmatrix},\quad T=\begin{bmatrix}0 & \frac{1}{3} & 0 & \frac{1}{2} & 0 & \frac{1}{2}\\ \frac{1}{3} & 0 & 1 & 0 & 0 & 0\\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{2} & \frac{1}{3} & 0\\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0\\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}\\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0\end{bmatrix},\quad \alpha=0.85,\quad d=\begin{bmatrix}\frac{1}{2}\\ \frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{bmatrix}$$

通过不停的迭代，直到都收敛。公式：$r=\alpha\cdot T\cdot r+(1-\alpha)\cdot d$ 衰减因子，一般取值0.80或0.85，关系矩阵T,d为种子

以上图谱关系可以进行改进和升级，如在传播的过程中可以加入通讯录关系，号码通，老乡、目前所在地、兴趣、职业等，形成以用户画像传播的好坏用户，在不同的用户画像传播中做不同的权重传播。

DataFunTalk

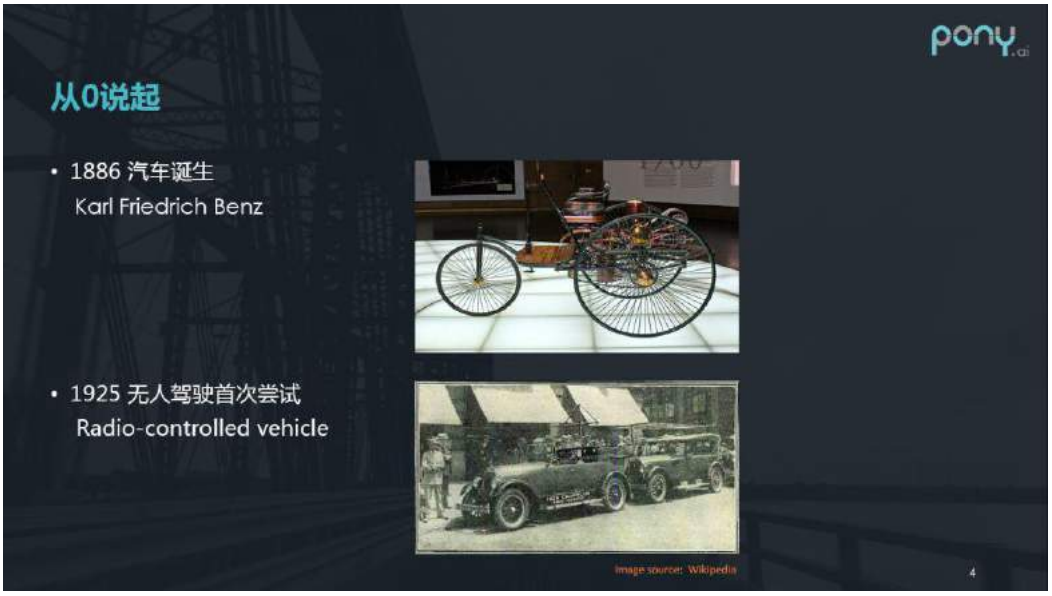
自动驾驶篇

自动驾驶从 0 到 1 之路

分享嘉宾：李衡宇 编辑：Hoh Xil

About Pony.ai：

从 0 说起



汽车是 1886 年诞生的，卡尔本茨先生建造的人类历史上的第一辆汽车，到现在只有短短 130 多年的历史。人类首次尝试无人驾驶的时间为 1925 年，当然这不是为了展示人工智能，而是用来展示无线电技术，它是用后面一辆车的无线电来控制前面一辆车的行为。之后的很多年，技术还是达不到让车辆自主行驶的水平。往往很多新事物的产生，都发端于科幻。比如，小说海底两万里中的潜水艇，成书的时候还没有潜水艇，但是在短短的一段时间之后潜水艇就出现了，所以科幻和科学是有密切关系的。自动驾驶也一样，在现实中还没有出现自动驾驶车辆的时候，科幻先描绘了自动驾驶的未来：

这个是 1982 年的美剧《霹雳游侠》，90 年代引入中国，当时也是比较火热的一部剧，里面就有描绘自动驾驶的片段，如上。

很快过了 20 年以后，在科研上，就出现了这样的汽车，最著名的就是 DARPA Challenge，DARPA 是美国国防部下属的研究机构，现在所使用的 GPS 系统，就是这家机构在 70 年代资助的。DARPA 在 2004 年第一次举办了无人车挑战赛，在一个沙漠路段，让自动驾驶汽车完成行驶，当时没有车辆完成挑战。但是到了 2005 年，有 5 个团队达成了目标，到 2007 年，开始挑战模拟的城市道路，并且需要在限定时间内完成。这个挑战赛的难度越来越大，并且开始有科研机构参与进来，其中最著名的就是斯坦福大学和卡耐基梅隆大学的研究团队，现在很多自动驾驶领域的领导者，都是当时研究团队里的研究生和教授。他们也贡献了自动驾驶领域最有参考价值的文献，如：

- "Towards 3D Object Recognition via Classification of Arbitrary Object Tracks"

- “Tracking-based semi-supervised learning”

- “Towards fully autonomous driving: Systems and algorithms”

所以在科研之后，工业界就开始关注这件事情，在最近的十几年还有很多事情让无人驾驶成为可能：

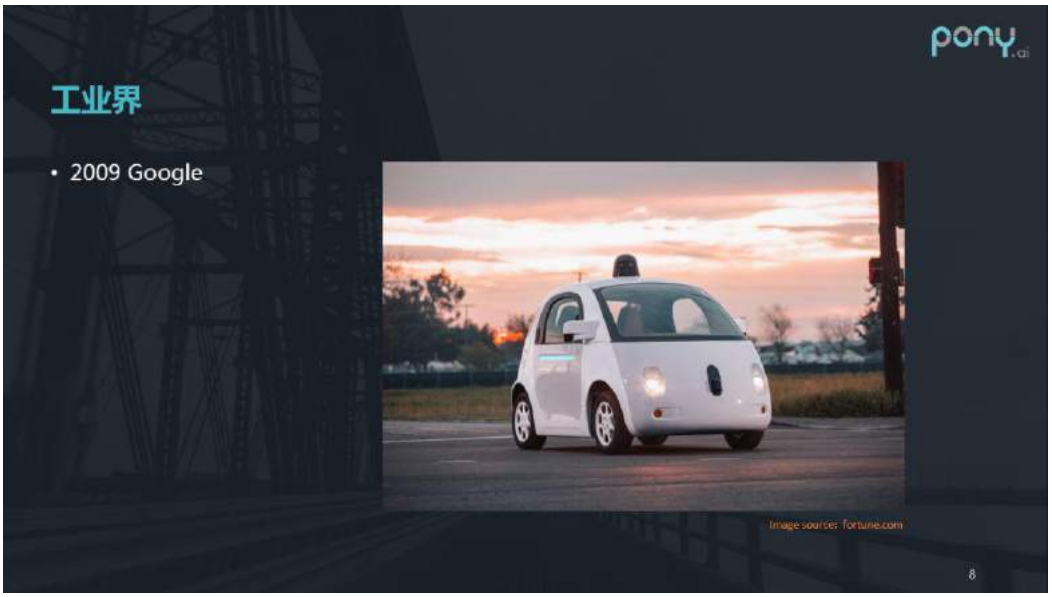
车辆技术：汽车的电子化和电动化。因为自动驾驶很多程序是需要电控系统的，如果是传统的液压或者机械控制方式，在反应速度和控制难度都会大很多。

算力&芯片：算力的提升和芯片技术的进步，最优代表性的就是 CPU。

传感器：比如激光雷达，让自动驾驶能够达到更好的感知效果。

大数据：互联网带来的大数据技术。

所有的这些再结合机器学习算法的进步，让自动驾驶成为可能，并让工业界进入这一领域。



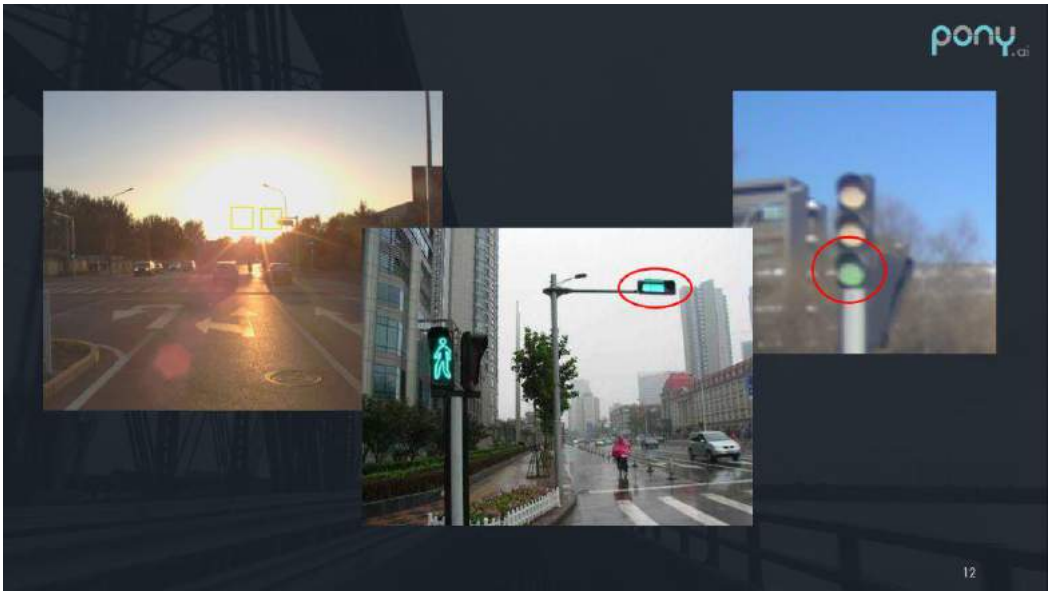
第一个是 Google，在 2009 年进入这一领域，并且收购了上述两个最优秀的大学科研团队。因此 Google 的无人车领先于世界，直到今天。

其他的公司也陆续进入了这一领域：

主要分为三类公司：高科技公司、汽车的一级供应商和主机厂。通过早期的文献，这些公司打造了自己的原型车。

当原型车出来之后，是否就是实现了从 0 到 1？不是的，原因可以参考上面的例子：现实非常骨感，即使我们根据前面的论文做出了原型车，在路上会遇到这样的情况，如逆行的电动车，这在挑战赛里是没有的。

还有这个场景，可以看到晚上会有一辆三轮车，并且上面会有非常庞大的物体，这对人工智能会是一个很大的挑战：如何判断这到底是个什么物体。



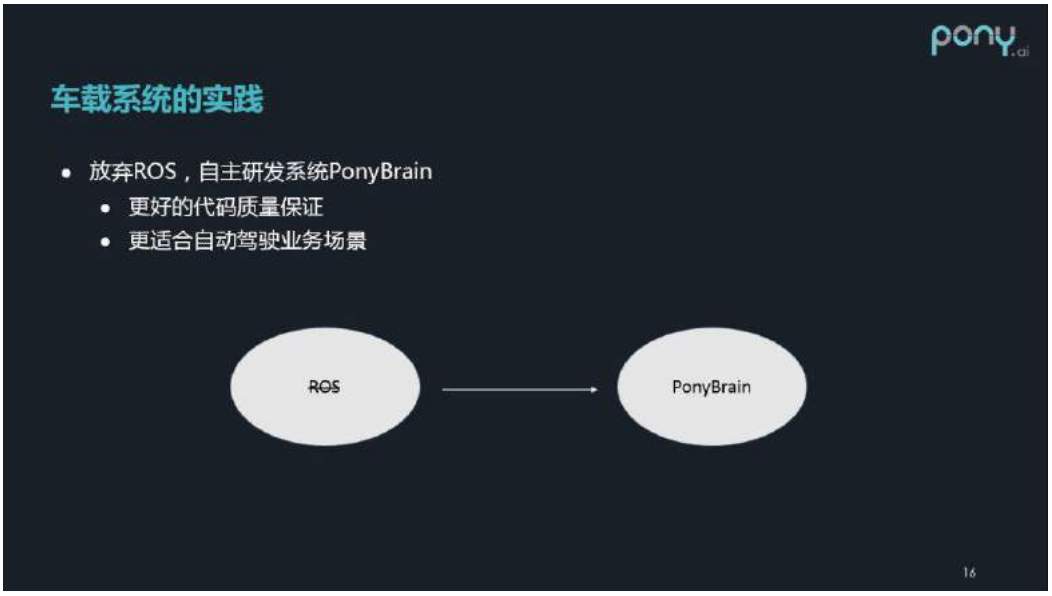
还有红绿灯，我们会发现每个城市，甚至不同的区域，红绿灯都有不同的模式，有各种的形状，以及逆光等等，都需要算法来解决，还有个更可怕的情况，如最右边的红绿灯，大家可以看到是个绿灯，但是真实的情况这个红绿灯坏了是不亮的，但是反绿。如果自动驾驶汽车要安全行驶就必须要解决这些问题，这些都是前面那些论文里不曾提及的。

工业界打造自动驾驶汽车，一定会商业化，并且创造市场价值，这不是做一个研究，所以最终的目标一定是：扩大车辆的数量，覆盖的范围，行驶的里程，运营的时段。所以这就带来了很多工程上的挑战。这里只是列举了其中一些：稳定性、可靠性、可扩展性、自动化、复杂场景处理、Corner case、大规模地图生成、地图增量快速更新、大数据存储和计算、BI、HMI、Control Center，所有的这些都需要我们去面对。

打个比方，就是从第一辆的奔驰车->5、60 年代的奔驰车->到现在的奔驰车。我们需要一次次的迭代。类比一下汽车换代，也许我们做出的第一辆原型车就是左边这辆车，但是我们最终的目标，是达到最右边或者中间的状态，虽然都是自动驾驶汽车，但是有非常大的不同。比汽车换代更难的是，我们的系统在升级换代时，是需要保持同步的测试的。

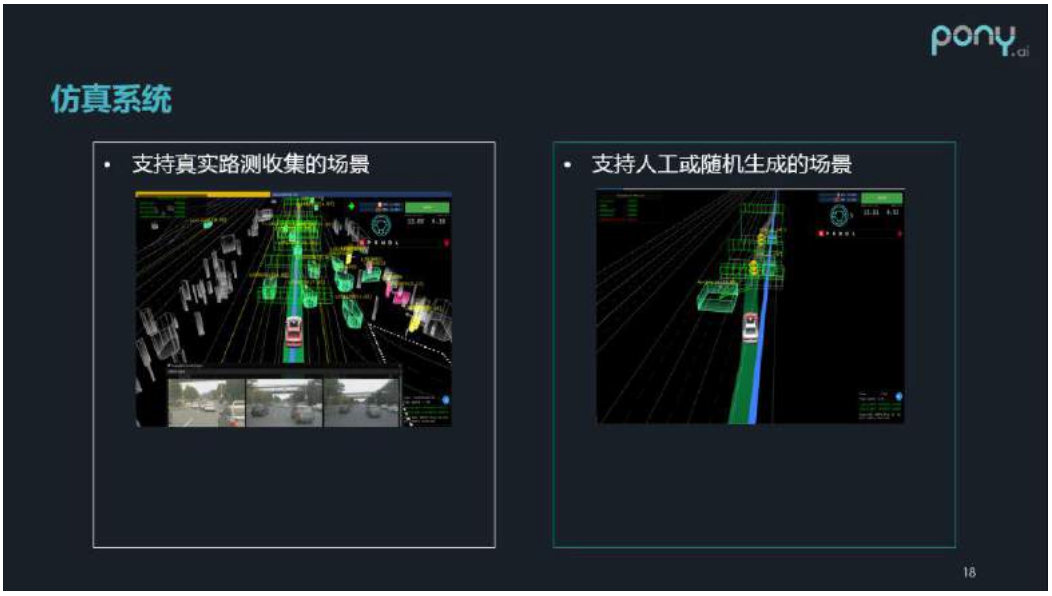
方法是我们会有一个非常快的迭代和流程，一个工程师如果要开发一个功能，或者修复一个 bug，会先在代码上做更新，然后在模拟仿真的环境下测试，测试的环境或者场景来自我们从各个地方采集的数据，如果都完成了之后，我们会进行场地测试或者路测，来验证是否达到预期，中间任何步骤如果不符合预期，会重新回到第一步骤，然后再次验证。实际这个流程是非常快的，一般当天就会有结果。如果测试结果 OK，每周都会同步到我们所有的测试车辆上。并且每个工程师都可以独立、并行的来做这件事情，这就保证了一个非常快的迭代节奏。

这是软件的迭代节奏，硬件上虽然不如软件这么快，但是我们也在保持一个非常快的节奏。



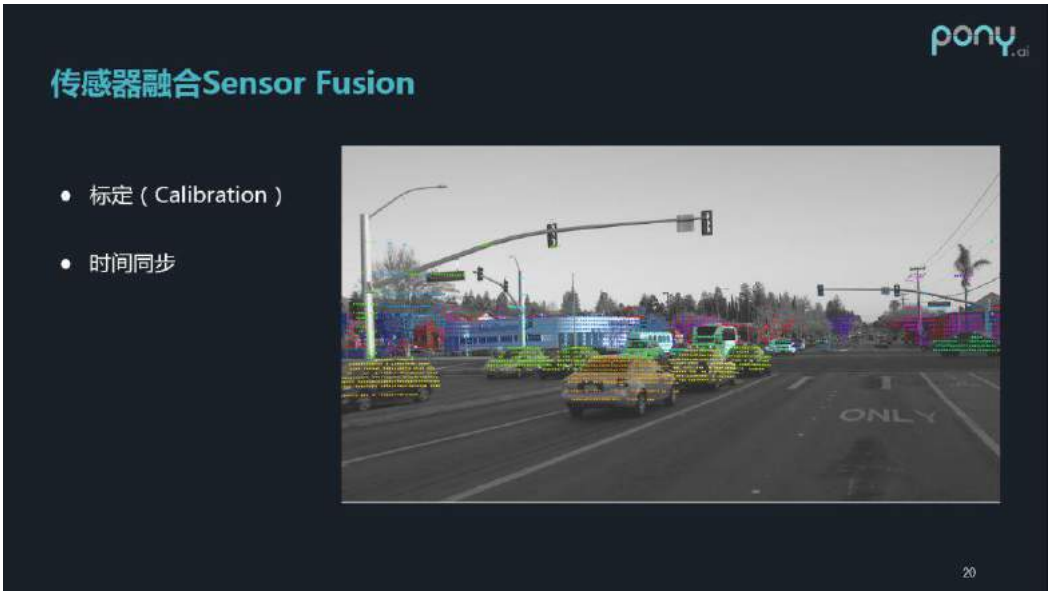
在技术上，我们也做了很多的创新，比如车载操作系统，我们放弃了 ROS，打造了自己的 PonyBrain 操作系统，目的是让系统更好的适应自动驾驶的场景。ROS 是一个分布式系统，并且它的时序是无法保证的，跟自动驾驶集中式的处理是相悖的，所以用我们自己打造的系统会有更高的效率，从我们实测来看，在同样的场景，我们系统的性能要高一个数量级。保证了车辆会有更大的空间去做算法的开发，并且能够保证安全性。如果实时性比较差的话，车辆在高速行驶中的安全性就得不到保证，因为 0.1 秒的差距就有产生很大距离的误差。

在数据上，我们也面临非常大的挑战，首先它跟互联网数据一样都很大，单辆车就会产生 PB 级的数据，远远多于一个互联网用户产生的数据，并且来源和互联网数据也不同，它的数据来自于客户端、大量的传感器数据以及模块运行日志。所以我们会自己打造一套数据系统，当然我们会利用之前已经有的技术，以及互联网的经验积累。

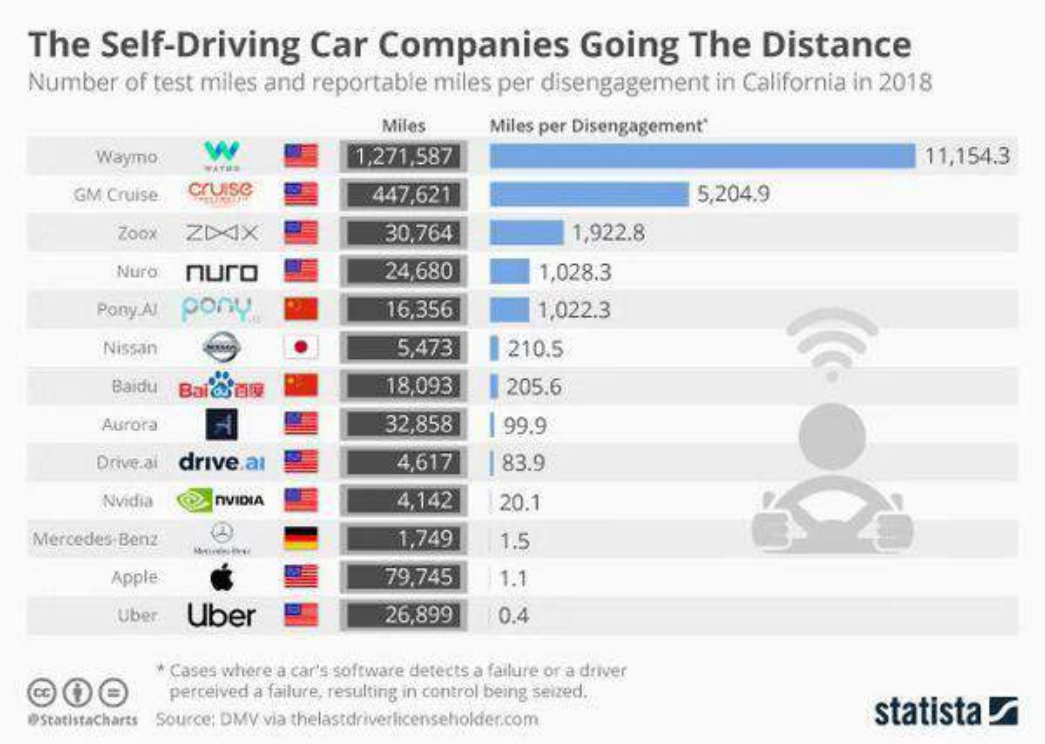


仿真系统，这个对我们也是非常重要，如果要保证算法的快速迭代，没有仿真的验证就会很慢，原因是很多的验证不可能去做实测，左边显示的是我们路测时候的场景，所有我们在路测过中认为是很重要的场景或者 case，都会收集到仿真系统中，所有代码的改动，都会触发重新测试这些 case，它可以保证我们的系统是进化的，而不是解决了一个问题，又冒出另外一个问题。仿真系统还有另外一个功能就是可以创造一些场景，因为某些场景在现实中是很难收集得到的。

高精地图也是一个绕不开的话题，与传统导航地图相比，我们的数据量非常大，并且要求厘米级的精度，而导航地图到米就足够了。最后就是更新的频率，导航地图可能一个月更新一次，对于自动驾驶，我们最早是天级别的，后来发现不够，又采用增量和全量更新的方式做到了小时级别，后来发现小时级还是不够，因为经常会有人把路封住，比如修路，这时单纯靠感知的能力不是特别理想，如果我们的第一辆车感知到这个场景，我们就会下发到其他所有车辆，告诉他们这个地方是不通的，所以我们可以达到分钟级别的响应。



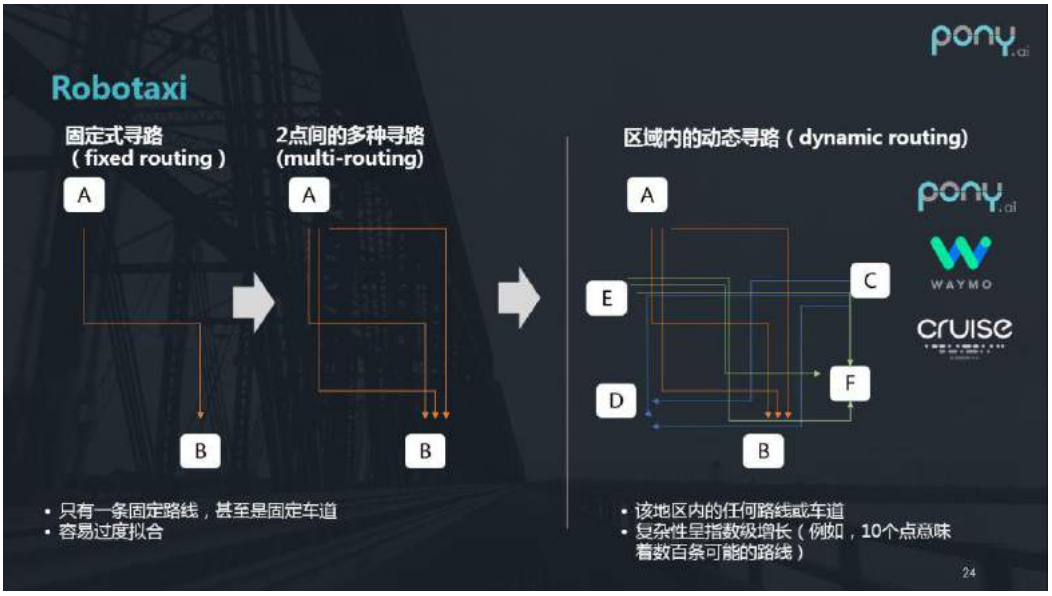
传感器的融合，最核心的传感器有毫米波雷达，摄像头，激光雷达等，这几个传感器都有各自的优点，目前还无法互相取代。那么如何去使用这些传感器，达到安全驾驶，我们的思路是把传感器的数据融合起来，这里展示的是，把激光雷达的数据和摄像头的数据进行了融合。激光雷达的一个特性是有非常准确的距离数据，摄像头的优点是分辨率非常高，并且有色彩和纹理，如果能把更多类的传感器数据融合在一起，那么后面无论是传统方法还是深度学习方法，都能取得一个更好的效果，从源头上解决感知效果的问题。当然这里有很多很难的技术，比如传感器的标定，我们需要把所有的传感器结果投影到一起，它们之间的 3D 关系，是一个很难的问题。还有就是时间的同步，每种传感器触发的时间都是不一致的 ,如果误差在零点零几秒 ,我们就无法做到图中的情况，做不到时间上的同步。



这是我们通过一系列创新和迭代达到的效果，这是加州车辆管理局（DMV）公布的，各家在加州获得牌照进行测试的一些自动驾驶公司的数据（不包括中国路测的数据），最重要的两个数据是里程和 MPI（平均多少公里以后需要人工接管，这个数据越高越好，表示车辆智能化程度越高）。

这是 Pony.ai 部分路测片段，包含了晚高峰、动态寻路和雨夜等场景：

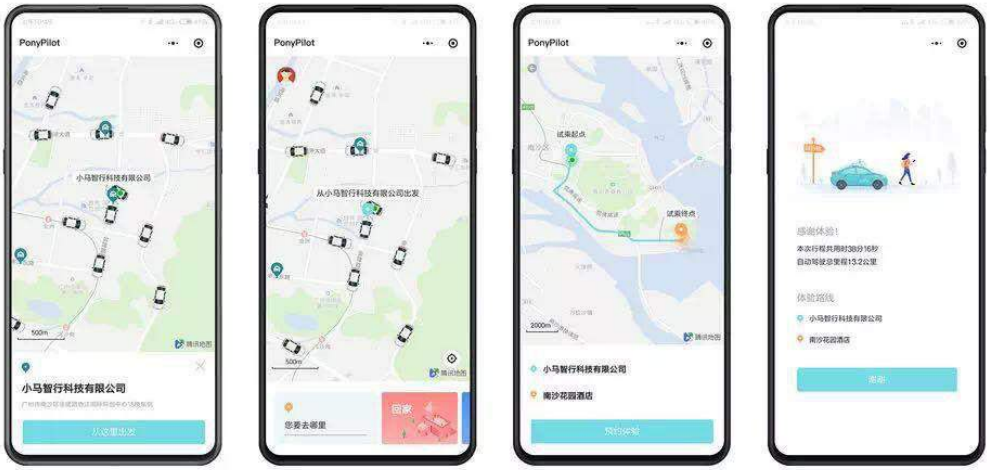
Pony.ai 部分路测片段



Pony.ai 在做的一个未来自动驾驶的可能落地模式：自动驾驶出租车（Robotaxi）。这里的技术挑战是什么呢？在很多公司的展示中，我们看到的是从 A 点到 B 点的展示，它只有一条固定路线，甚至是固定车道，并且容易过度拟合。而区域内出租车是多点动态寻路，面临的场景是该地区内的任何路线或车道，复杂性呈指数级增长。

2018 年 12 月，我们发布了 PonyPilot 项目的同名内测版小程序。通过小程序，试乘者在覆盖区域内可以自由选择想去的目的地。目前除了内部员工外，该小程序的权限为邀请制，未来考虑开放为可自由下载。

PonyPilot 体验



PonyPilot 内测版小程序

离 1 还有多远

我们自己认为自动驾驶会有这样的几个阶段：最早的就是原型车或者普通的展示，然后有些公司会往前走，做一个优秀的展示，第三个阶段就是我们现在在努力实现的，打造的无人驾驶车队的运营，只有通过这个

阶段来验证系统的可靠性，去不断的收集数据，不断的算法迭代，才有可能达到商业化，它无法从一个优秀的展示，直接跨越到商业化产品，所以中间的阶段是目前头部公司一直在努力达到的一个阶段，这个阶段做好之后就会尝试在某个区域进行商业化的运营，这时自动驾驶就会真正进入到人们的生活，到最后才是自动驾驶进入城市或者千家万户。最终的目标还非常遥远（10 年计），但是自动驾驶的商业化没有那么远。

最终自动驾驶会成为一个现实，我们的智慧交通、智慧城市也会真正的实现，也许到那时在道路上开一辆非自动驾驶的汽车是非法的，比如现在在马路上骑马就是非法的。我们期待着这一天的实现，谢谢大家。

自动驾驶中轨迹规划的探索和挑战

嘉宾：梁亚雄　编辑：孙昊

大家好，今天我们主要介绍一下轨迹规划的探索和挑战，我主要从四个方面介绍:

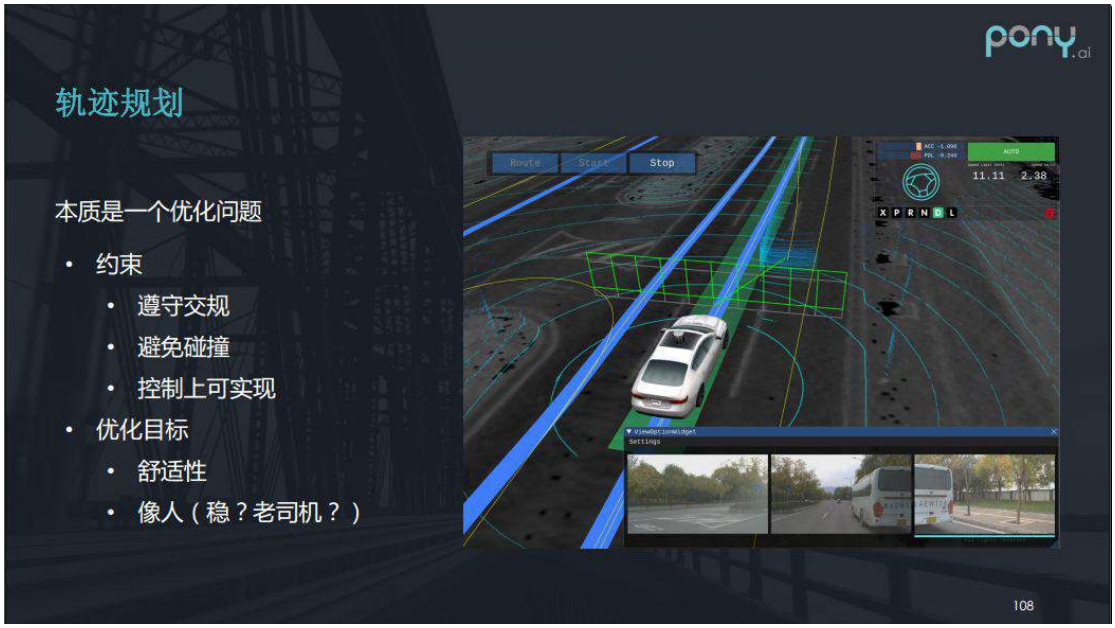
- 轨迹规划的概念
- 决策
- 横向规划
- 纵向规划

轨迹规划的概念：

轨迹规划的核心就是要解决车辆该怎么走的问题。比如我们知道了附近有行人、骑自行车的人以及前方的卡车，如果我们现在将要左转，该怎么办？这就是轨迹规划该解决的问题。

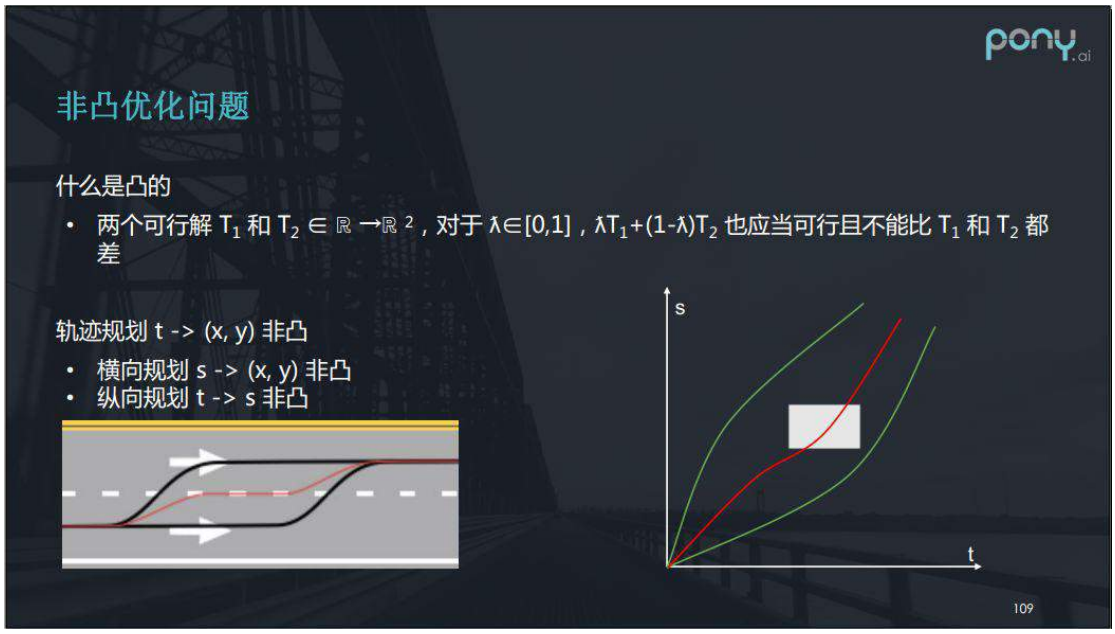
轨迹规划的输入包括拓扑地图，障碍物及障碍物的预测轨迹，交通信号灯的状态，还有定位导航（因为要知道目的地是哪才能规划路径）、车辆状态等其他信息。而轨迹规划的输出就是一个轨迹，轨迹是一个时间到位置的函数，就是在特定的时刻车辆在特定的位置上。在 PPT 上是 t ->（x,y,z），把 z 标灰是因为目前为止我们的车不会飞。

轨迹规划本质上来说是一个优化问题。谈到优化问题，我们需要知道优化的约束和优化的目标。首先看约束，第一个约束就是车辆要遵守交规，这是强制性的约束；第二个约束是要避免碰撞；第三个约束是要使规划的轨迹在控制上可实现，不能出现规划了一个急弯的轨迹，车根本拐不过来的状况。而优化的目标总的来说就是要跟人开的像，具体表现就是让人乘坐起来感到舒适。但像人也有不同的优化目标，比如我们可以开的稳一点也可以开的急一点，像老司机一点。



这样一个优化问题在数学上的性质是什么呢？我们关心一个优化问题首先看这个优化问题是不是凸的，因为凸的问题比较好解一些。什么是凸的？我们可以简单描述对于一个问题它的两个可行解，其线性组合也应该是可行的，而且不比这两个解都要差。那么轨迹规划是不是一个凸优化问题呢，并不是,它是非凸的。为什么呢？我们在对 t ->(x,y) 问题进行求解时，复杂度比较高计算量比较大。所以通常的做法是把他分成横向规划和纵向规划。横向规划，是 s ->（x,y）,就是决定轨迹的形状是什么。纵向规划是 t ->s, 也就是指在此形状上运动的速度状态，也就是时间与位移的关系。横向规划和纵向规划联合起来就是 t ->(x,y)。那么横向规划是凸的吗？如下图，图上没有时间信息，这就是横向规划。两条黑线代表不同的变道轨迹都是可行解，然后我们看其线性组合也就是中间的红线，但明显红线的轨迹是不能用的，是很差的，他变道太慢，骑线行驶的距离太长了，所以横向规划是非凸的。

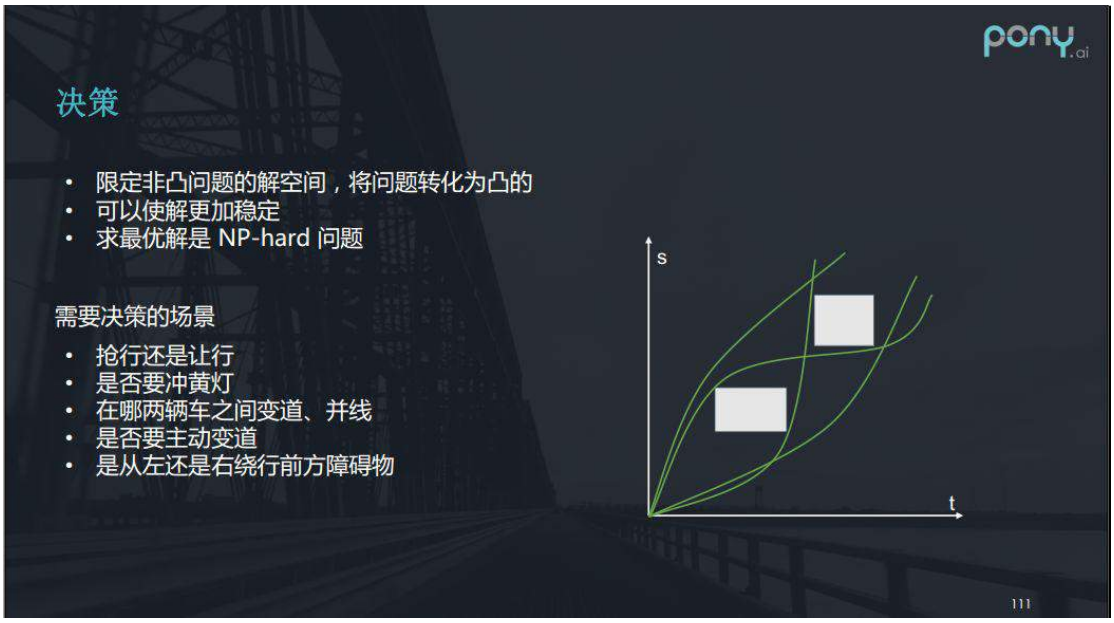
我们再来看纵向规划，我们经常用 t - s 图表示。t 表示时间，s 表示我们走过的路程。当一个人横穿马路时我们可以用白色矩形来在 t - s 图上表示这个过程，左边界表示人进入我们规划路径的时刻，右边界表示离开规划路径的时刻。横向上，上下边界差可以理解为人在占用我们规划路径的宽度。对于这样一个纵向规划，有两种选择，一种是我们要让人，既下边的绿色路线，一种是车辆加速超过，既上边的绿色路线。但它的线性组合可能是这条红线，明显这已经撞上了，所以这并不是一个凸的问题。

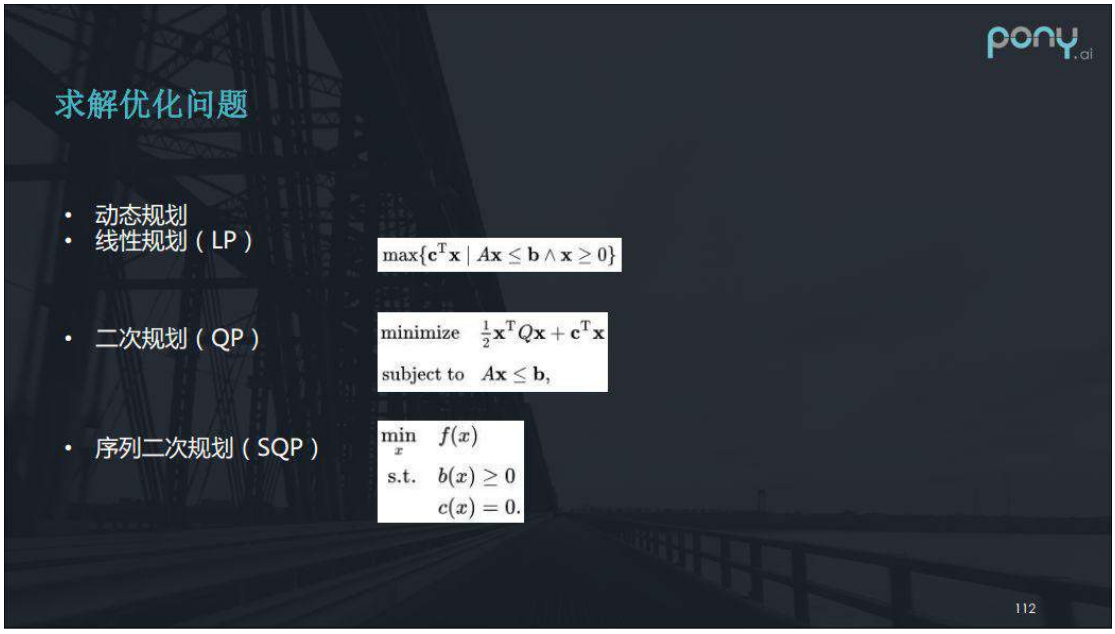


决策：

正是因为轨迹规划是非凸的，所以我们需要用决策的模块来解决。什么是决策呢？从数学上来讲，决策就是限定非凸问题的解空间，将问题转化为凸的。我们可以看下图的例子，如果两个人在穿马路的话，我们大体上可以有四种不同的决策，既让两个人，超两个人，让一超一和超一让一。而一旦决策确定，那么其实就可转换为凸的问题，求解就

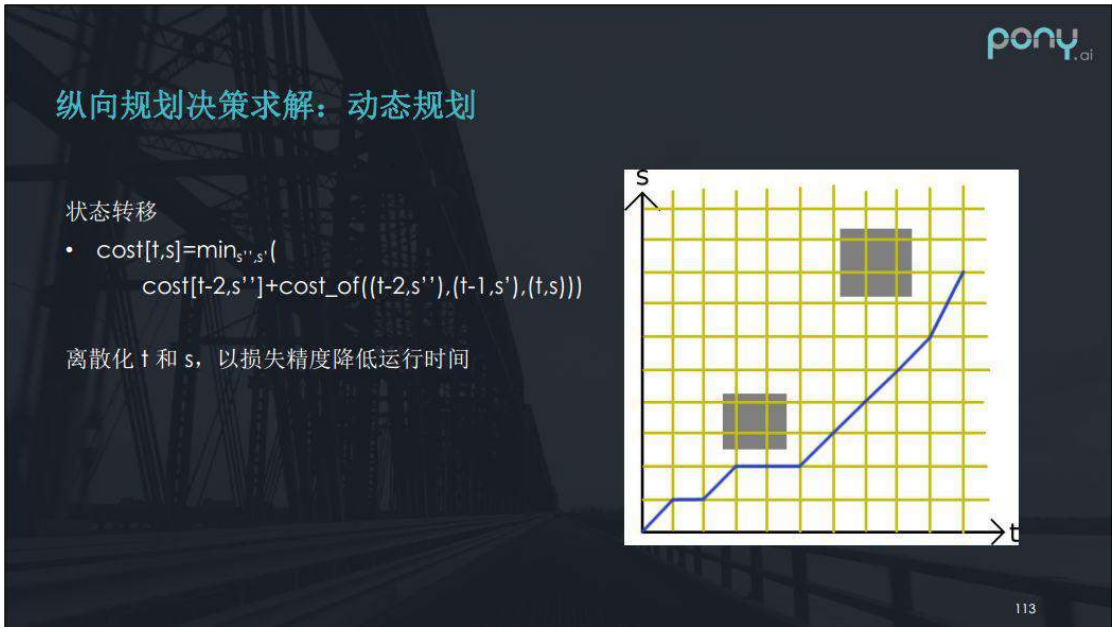
会相对来说比较容易。决策也会使解更加稳定，因为一个单纯的数值优化问题的求解，会很难保证每一帧的解是相对稳定的，不会出现第一帧要让，而第二帧却要冲的问题。但需要注意决策的问题是一个 NP-hard 问题。有一些需要决策的场景，比如抢行还是让行，是否要冲黄灯，在哪两辆车之间变道、并线，是否要主动变道，是从左还是右绕行前方障碍物，只有做了决策才能使问题变为凸优化问题，便于求解。对于凸问题，有很多比较快速的算法进行计算了。线性规划，二次规划，序列二次规划，都是数值求解优化问题的方式。我们可以看一下二次规划中，如果 Q 正定，二次规划就是凸的问题，有很快的方式求解它。





下边就是看如何把自动驾驶中的问题转化为优化问题。

首先是纵向规划的决策，决策的方法我们已经谈到了一些，比如动态规划。这是一个 NP-hard 问题所以没有一个简单的算法进行精确求解。看下图，如果有两个人穿马路，我们来研究一下如何走才是最优的，我们可以将 T-S 进行离散化，离散化之后，我们就可作出一个状态转移方程，而这个状态转移方程我们可以看作每一步的决策只与前边的两步有关系，所以有转移方程。虽然这个方程在实际过程中比较难写但确实是一种做法。虽然离散化 t 和 s 降低了精度，但降低精度也能降低了运行时间。注意，这种方法并不能保证最后的速度比较舒适，它只是大概的一个决策方法，既到底让或者不让。

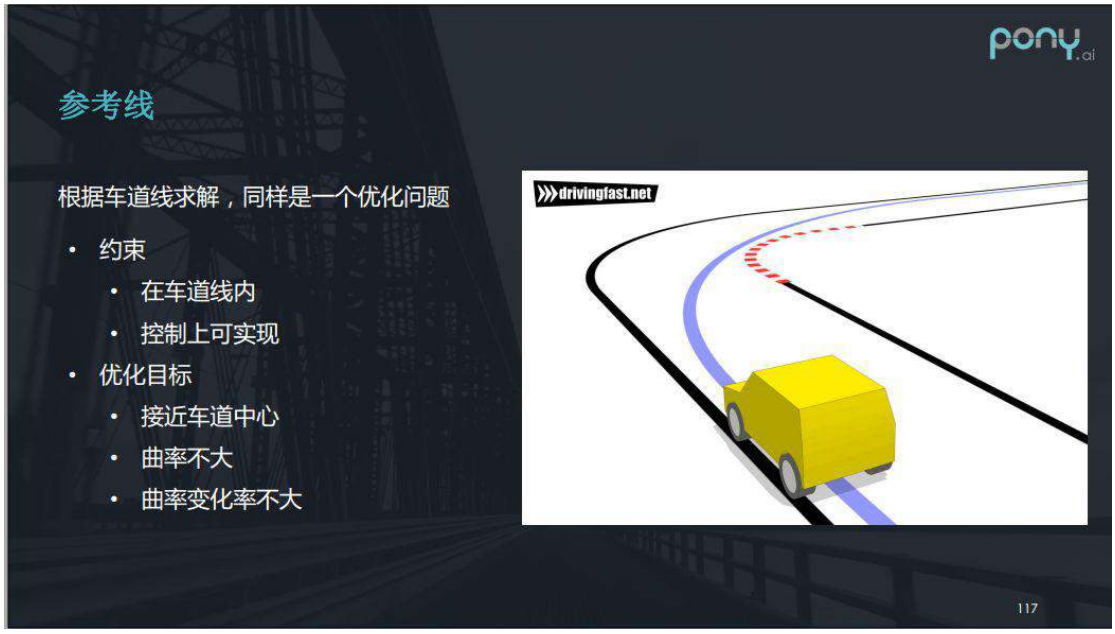


我们再来看一下决策方面的挑战。第一个挑战上面已经讲过了，决策问题是一个 NP-hard 问题，不易直接求解，有多种多样的近似算法。第二个挑战是很难用规则去拟合人的经验，包括上述的状态转移方程中的 cost 也很难去表示。而一些解决的办法是我们可以根据各种不同的情况去建立一个数学模型。以及用机器学习的方法去做一些决策。比如说下图左边的例子，我们已经打算绕行，但前车突然起步了，我们该怎么办？我们是该变道还是跟在后边走还是继续绕行？这种情况确实很靠人的经验。再看右边的图，我们要左转，但是前边这辆车停了很久也没有走我们要不要变道呢？作为人类司机也很难判断。场景的多变而复杂使得决策面临着很多挑战。

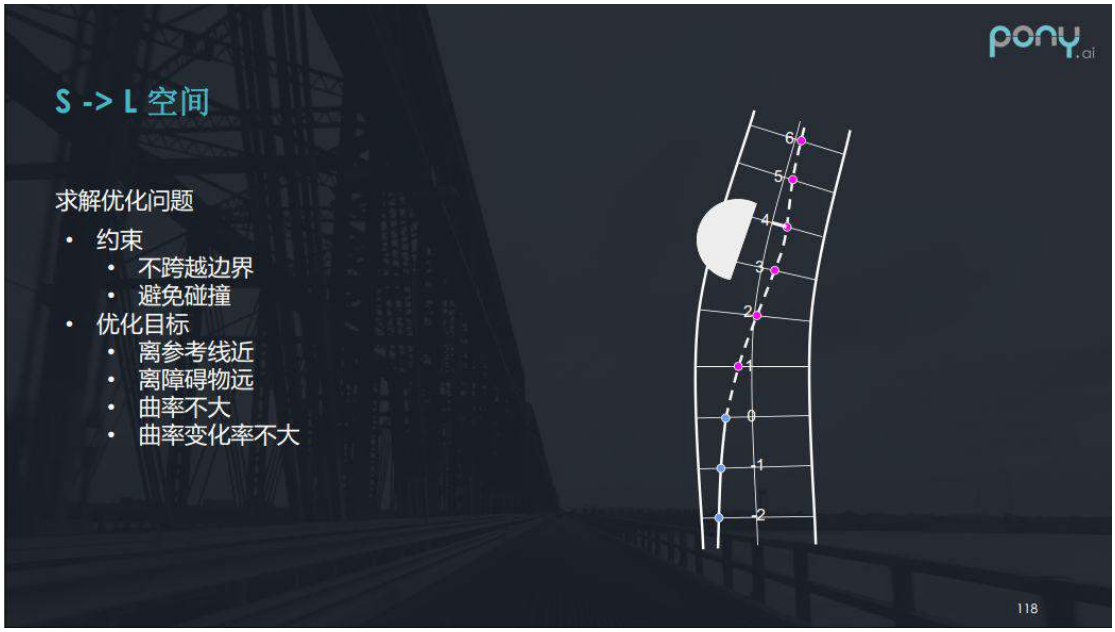
横向规划：

刚刚我们已经说过了，轨迹规划可以拆成横向和纵向的规划。横向规划是一个方向的规划，它可以看成一个如何打方向盘的规划，它只决定了轨迹的形状。这个问题通常的解法分两种，一种是无车道的，比如说 freespace 中的规划，或者说是停车之类的问题，它是没有车道线信息或者说先验信息的，大多都用 OpenSpace 路径生成的方式去做，一般也是低速的。对于有车道的情况，虽然有车道线但是输出 s ->(x,y)这个函数难度也是比较大的，常见的做法是是离线生成参考线，之后我们就可以将 s -> (x , y) 这个问题变为一个求解 s ->L 的问题，L 是指车辆在这个参考线上的横向偏移量。可以看到下图右边这张图，本来是沿车道往前走，但由于有车辆的遮挡，我们就必须绕行它。即横向偏移就可以了。

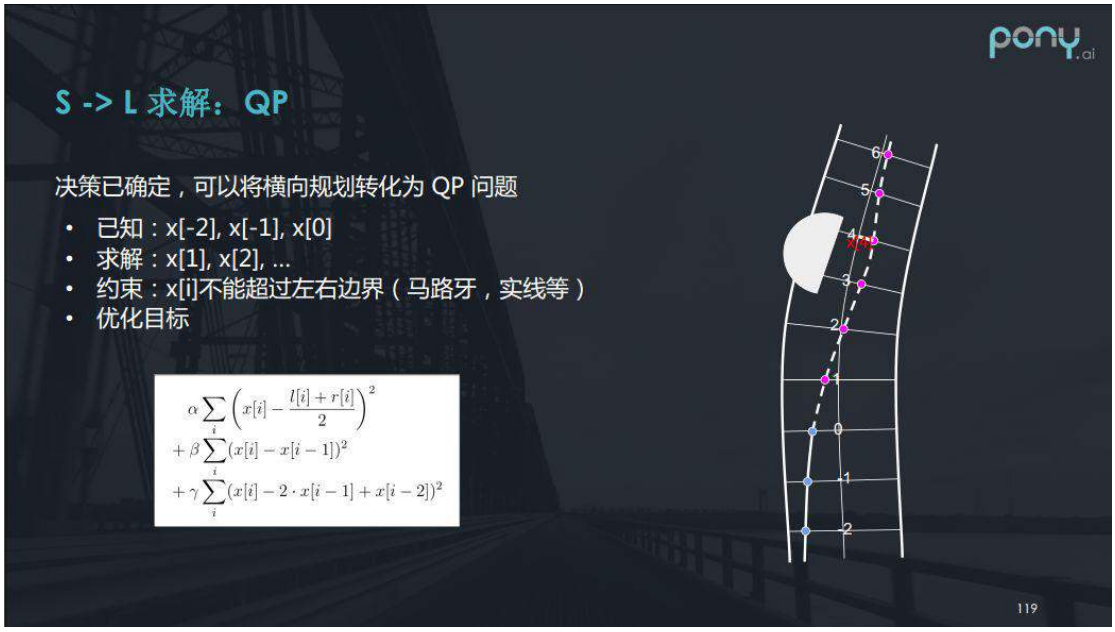
对于参考线的生成，很类似于开卡丁车时别人教你的过弯的最优路线，它也是一个优化问题，当然也要保证安全性和舒适性，但是方便的是有了地图以后我们是可以离线去做的，所以可以用一些开销比较大的算法做到非常优。它的约束是在车道线内，且控制上要可实现。优化目标就要接近车道中心，曲率不能太大且曲率变化率不大。



有了参考线之后我们就可以把参考线离散化，采一些点出来，那么就变成了解一个离参考线偏移距离的一个问题。就变成了 s ->L 的问题，约束是不跨越边界，避免碰撞，而优化的目标是要离参考线近，要离障碍物远，曲率不大，曲率变化率不大等等。

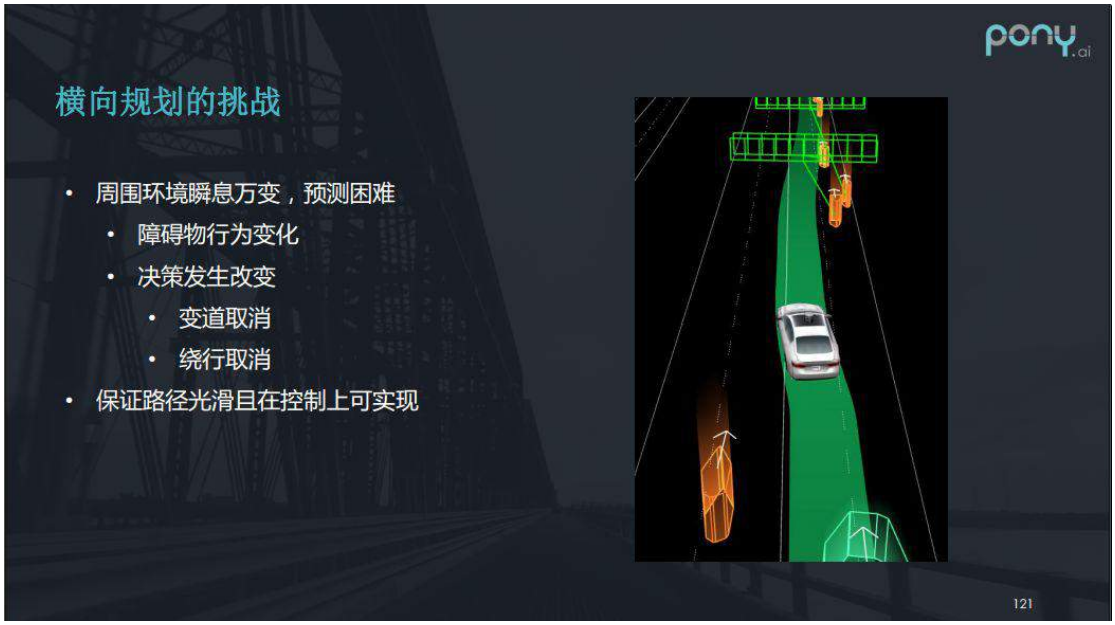


我们看下这个例子，那么这个问题就可以看成一个二次规划 QP 的问题。0，-1，-2 是我们车过去的路径，0 号点是我们车当前的位置，我们需要解的就是 1，2，3，4，5，6 这种点的相对于参考线的横向偏移 x，已知就是 x[-2]，x[-1]，x[0]，求解 x[1],x[2] 等，而约束就是他不能超过左右边界包括马路牙，实线，障碍物等。优化目标就是我们要离中心参考线近，方向盘不能打太多太快。下图中的公式的形式就是一个关于 x 的二次的形式，所以它们可以去用 QP 的方法来解。



之后我们再来看看横向规划的挑战。虽然大部分时候车都在一个有车道线的地方开，但比如像下图左下部分的路口，这边是三条直行的车道，而对面是四条条直行的车道，右边的白车就不会按车道线开，会横跨车道线挤你。右上的图，是有辅路的情况，他是没有车道线的，这种进辅路的问题更像是一种没有参考线的 freespace 的规划，这种都是比较大的挑战，如果说没有车道线或者说没有人按车道线开，那么这样的规划就会比较难做一点。

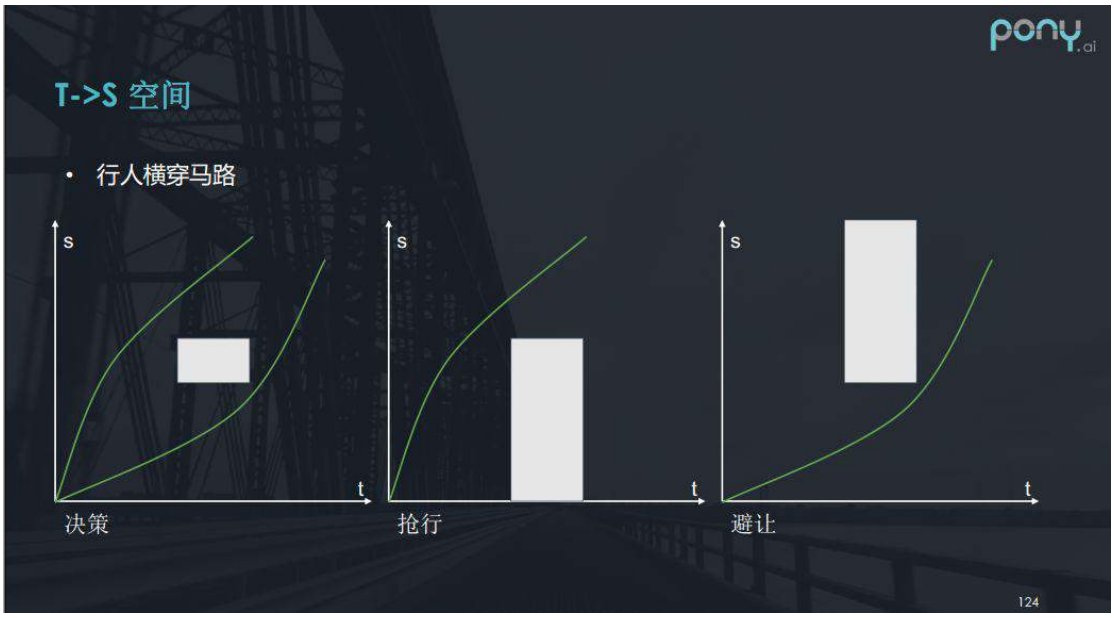
另外一些挑战就是环境的问题，因为周围的环境是瞬息万变的周围的环境预测也是困难的，我们看下图，本来车辆是要往左变道，而左下角黄色的块表示摩托车，这个摩托车正在飞速的过来，所以车辆又进行了取消变道的决策，也就是生成平滑的曲线回到原来的道路上去，这样的情况我如何能保证路径光滑且在控制上可实现，也是一件很难的事情。

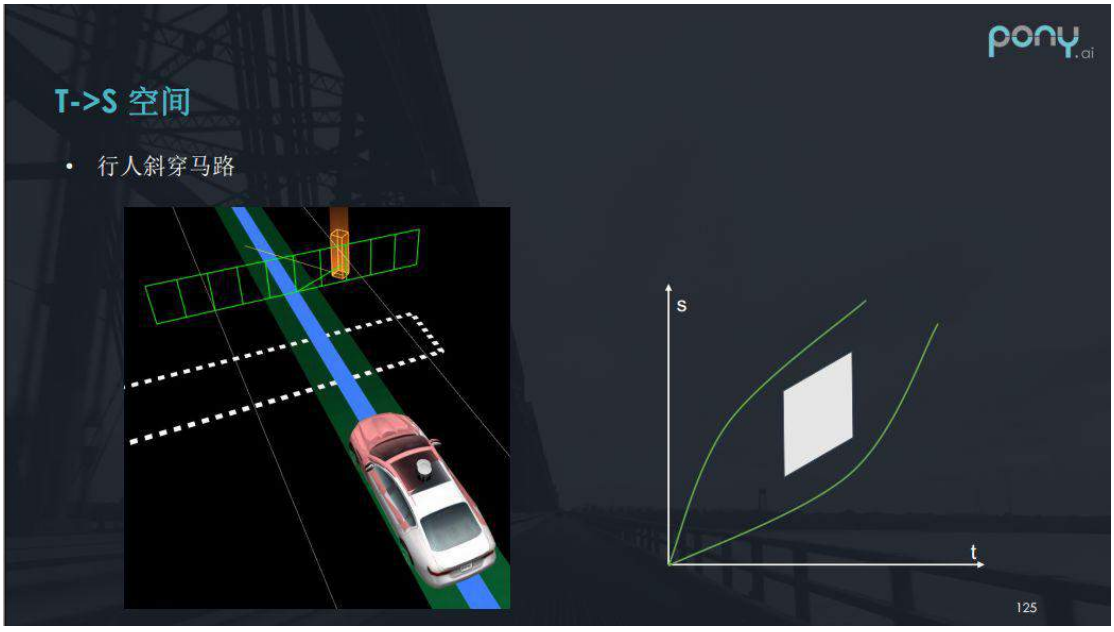


纵向规划：

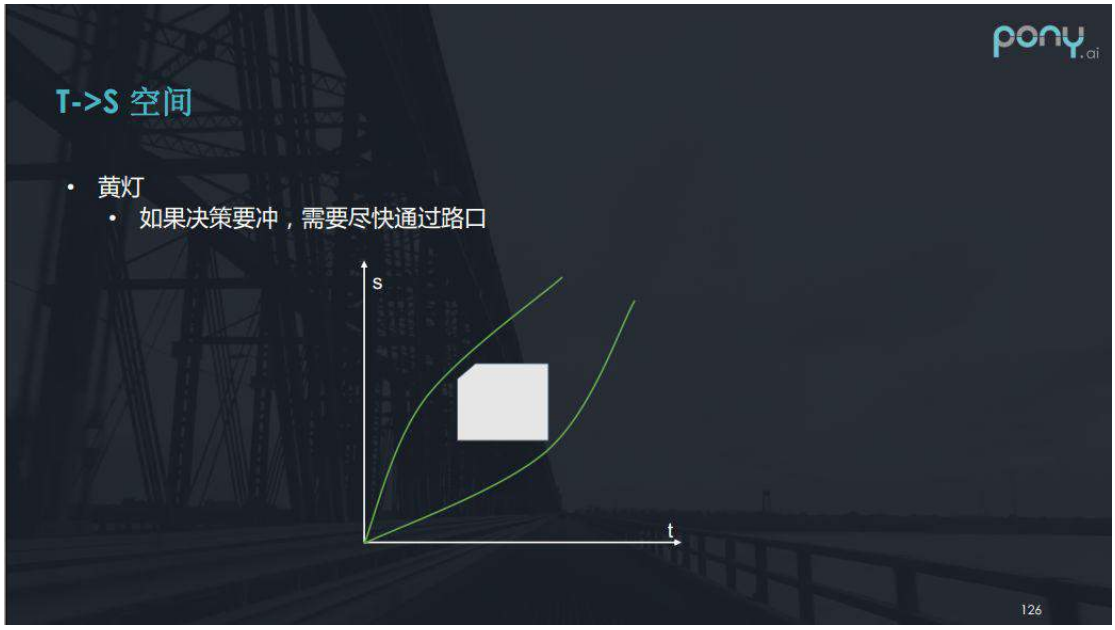
纵向规划其实是对于我们设定好的路径上的速度上的规划，决定了在这整个轨迹上的运动过程。求解这种优化问题，有的约束第一就是遵守交规（信号灯、限速、停车让行等），第二就是避免碰撞。优化的目标，想都可以想的到就是舒适，舒适就是速度变化率不大，加速度变化率不大，行驶也要尽量快一点，等等。

我们举一些例子，前边提到了行人横穿马路，行人就是这样一个矩形，我们对应了两种决策，决策完成后我们应该怎么办呢？如果决定要抢行，我们就可以将矩形的约束条件扩展到最下部，这样就是凸问题求最优解。如果是避让我们就可从下边过，但是如果是行人斜着穿过马路呢？那么请大家想一下按 s ->t 图是如何表示的呢？答案就是它是斜向上的平行四边形。





再看一些类似的例子,如下图。比如黄灯，黄灯是一个特殊的例子，如果决策要冲的话，需要尽快通过路口，否则的话很容易被逼停在路中间出不去，这种情况我们可以用一个这样的 s - t 图表示，左边界是表示黄灯亮的时刻，大家看这样的白色矩形有一个缺角，当黄灯亮起的时候，我们如果要尽快通过路口，t 在增大的过程中 s 也要尽快的增大，增大的速率要超过缺角的斜率。



我们再看一些更高级的想法，我们在跟车的时候，假如前车在匀减速，在 s - t，如下图所示，如果能够精确描述前车的运动的状态，那么白色部分会有各种各样的形状，这样就可以在解优化问题是解出一个好的速度曲线。

之后我们再看一下纵向规划的挑战。如下图，比如我们绿灯左转，而有一个电动车，在左侧很快速的横穿马路，正常来说那么人应该怎么开呢？人会和电动车司机对一下眼神，通过眼神比较气势谁猛，然后谁就会主动的让对方，开个玩笑。但在决策上是这个事情并不好处理，这是一个博弈的过程，并不能一开始就决定要无视他还是让他，而要在一开始对他有一点反应，所以在很多时候要在激进和保守之间掌握一个平衡点，很多时候要有不同的参数和不同的模式去处理这个事情。

另一方面，感知和预测带来的困难也会带来纵向规划的挑战，可以看下边这连续两张图，突然就有人从车前冲出来了，这个我们俗称叫做鬼探头，你要检测到有盲区，需要在在检测到有盲区的时候进行减速等等。还有问题就是预测，我们可以看到左下角的图，右车道有一个面包车，而前边有速度慢的一辆自行车，人一看就知道，这个面包车极有可能变道，但这对预测也提出了很大的挑战，如果没有预测那么你的反应就会比较慢。右下角的图，你也无法判断这辆左拐的车要拐到哪条车道上去。

还有一些极端的情况，需要考虑到横纵向协调配合，比如下图一个极限的车辆插入，你在高速行驶，而旁边的车突然就要插进来，如果是人类司机他会打方向避让，如果左边没有车，甚至会变道，左边有车，他也会扭一点方向去避让。这种就需要横纵向的配合去一起解决这种极端的情况，从纵向规划来说，这种情况，你已经不可能保持安全车距了，你更多的只能说保证不撞，然后尽快拉开车距，而不是一脚刹到底。

pony.ai
纵向规划的挑战
• 与横向规划之间的协调
• 横向规划需要考虑纵向规划的能力
• 统一为一个优化问题
• 远小于安全车距的 cut-in
• 保证不撞，并尽快拉开车距
高速、近距离突然 cut-in
26

L4 自动驾驶中感知系统遇到的挑战及解决方案

嘉宾：李阳光　编辑：安小宏

本次分享的大纲：

- Perception Introduction
- Sensor Setup & Sensor Fusion
- Perception Onboard System
- Perception Technical Challenges

1. Perception Introduction

Perception 系统是以多种 sensor 的数据，以及高精度地图的信息作为输入，经过一系列的计算及处理，对自动驾驶车的周围的环境精确感知。能够为下游模块提供丰富的信息，包括障碍物的位置、形状、类别及速度信息，也包括对一些特殊场景的语义理解（包括施工区域，交通信号灯及交通路牌等）。

Perception 系统包括多个方面及子系统：

1. 传感器：传感器的安装，视场角，探测距离，数据吞吐、标定精度，时间同步。因为用到的传感器比较多，时间同步的解决方案这里会起到非常重要的作用。

2. 目标检测及分类：为了保证车辆百分百的安全，能够达到近似百分之百的召回率及非常高的准确率；这里会涉及到深度学习方面的工作，包括 3D 点云及 2D Image 上的物体检测及多传感器融合方面等。

3. 多目标追踪：跟进多帧的信息计算并预测障碍物的运动轨迹。

4. 场景理解，包括交通信号灯，路牌，施工区域，以及特殊类别，比如校车，警车。

5. 机器学习分布式训练基础架构及相关的 Evaluation 评价系统。

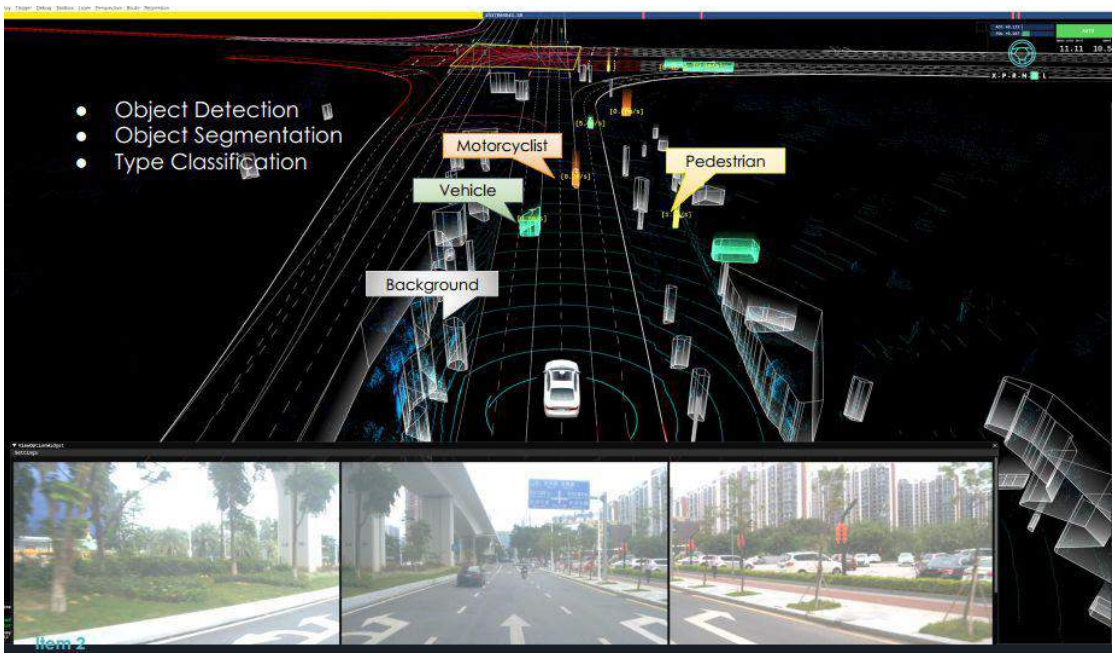
6. 数据：大量的标注数据，这里包括 3D 点云数据及 2D 的图片数据等。

目前传感器主要分为 3 个类别：

1. Lidar 激光雷达；

2. Camera 相机；

3. Radar 毫米波雷达。



这张图相当于 Perception 物体检测的输出，能够检测车辆周围的障碍物，包括车辆、行人、自行车等，同时结合高精度地图，会对周边的 Background 信息进行输出。

图中绿颜色的是一辆车，橙色的是一辆摩托车，黄颜色的是一个行人，灰颜色是一些背景如植被信息。

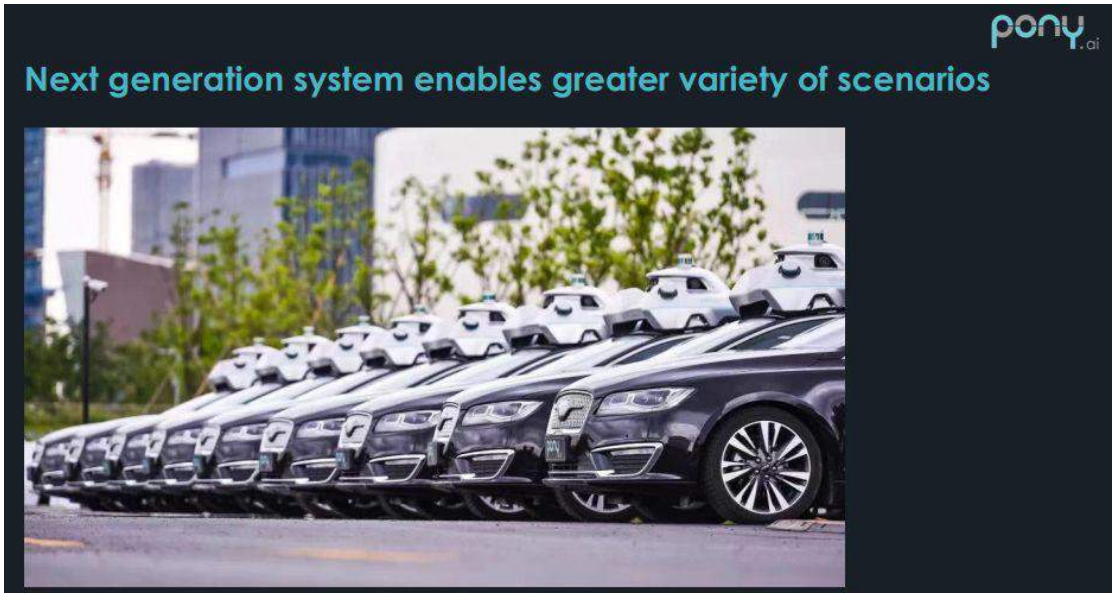
结合多帧的信息，对运动的行人和车辆的速度，方向，进行精确的输出。

2. Sensor Setup & Sensor Fusion

以上是 Perception 系统从输入到输出的一个大概介绍。接下来总体介绍 Pony.ai 第三代车辆的传感器安装方案以及传感器融合的解决方案。

目前我们的整个传感器安装的解决方案，是能够覆盖 360 度，范围 200 米的感知距离。从不同传感器的安装，不同角度来看，首先用到了 3 个激光雷达，顶部和两侧。激光雷达有 100 米的感知距离。同时通过 4 个广角的摄像头来覆盖相机 360 度的视野。远处的视野通过前向的毫米波雷达以及长焦的相机，将感知距离扩到 200 米的范围。这套

传感器配置能保证我们的自动驾驶车辆在居民区、商业区、工业区这样的场景进行自动驾驶。



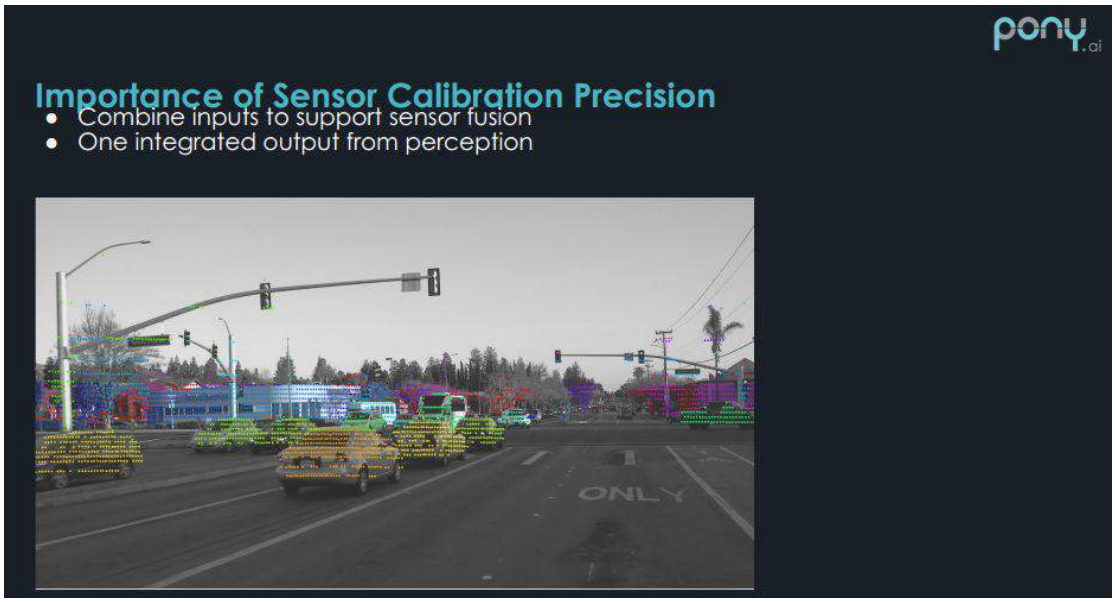
18 年 9 月份世界人工智能大会上推出的第 3 代的传感器配置方案。

前边的相机两个广角，一个长焦。使得可以看到更远距离的信号灯的信息，两百米内红绿灯的状态。

上面介绍了整个传感器的安装方案。下面主要介绍下多传感器融合的解决方案。

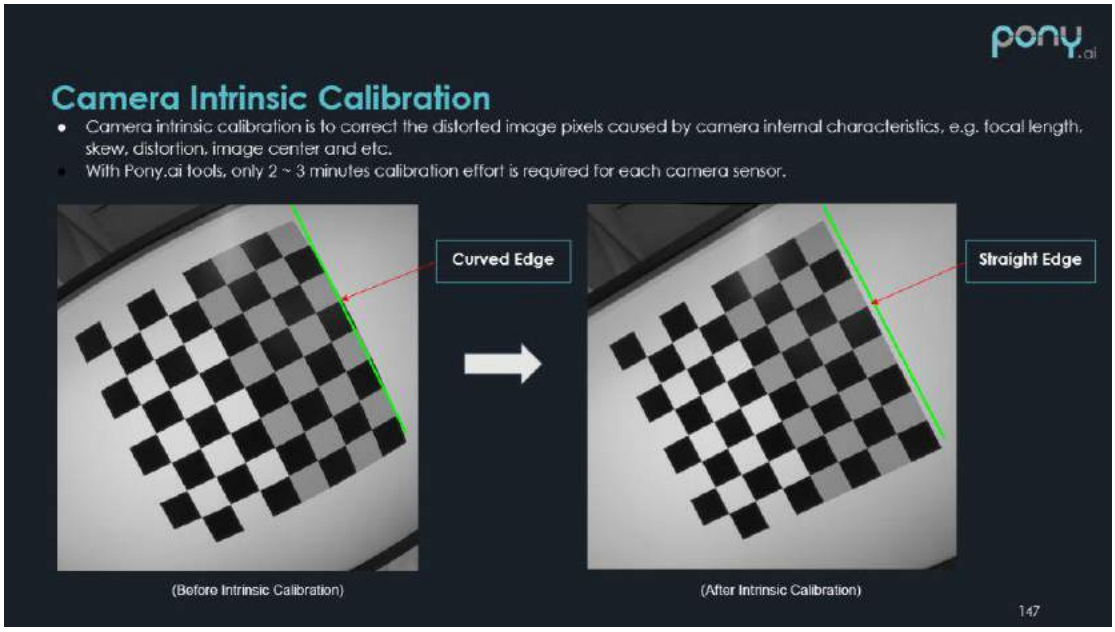
第一个要解决的问题是把不同的传感器的数据标定到同一个坐标系里。包括相机的内参标定，Lidar 到相机的外参标定，雷达到 GPS 的外参标定。

传感器融合重要的前提是要能把标定精度提高到非常高的一个精度，不管是做结果层面的传感器融合，还是元数据层面的传感器融合，是一个必要的基础。

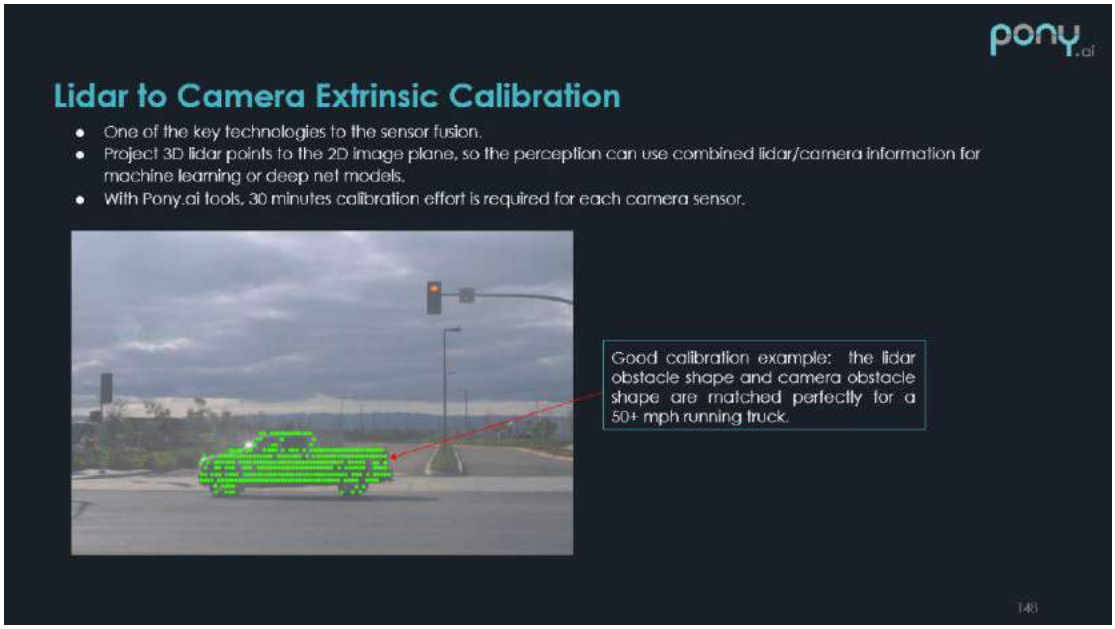


这张图可以看到，我们将 3D 的激光点云投射到影像上，可以看到传感器标定的精度还是挺高的。

整个标定的工作基本上已经做到完全自动化的方式。不同传感器标定的方案：

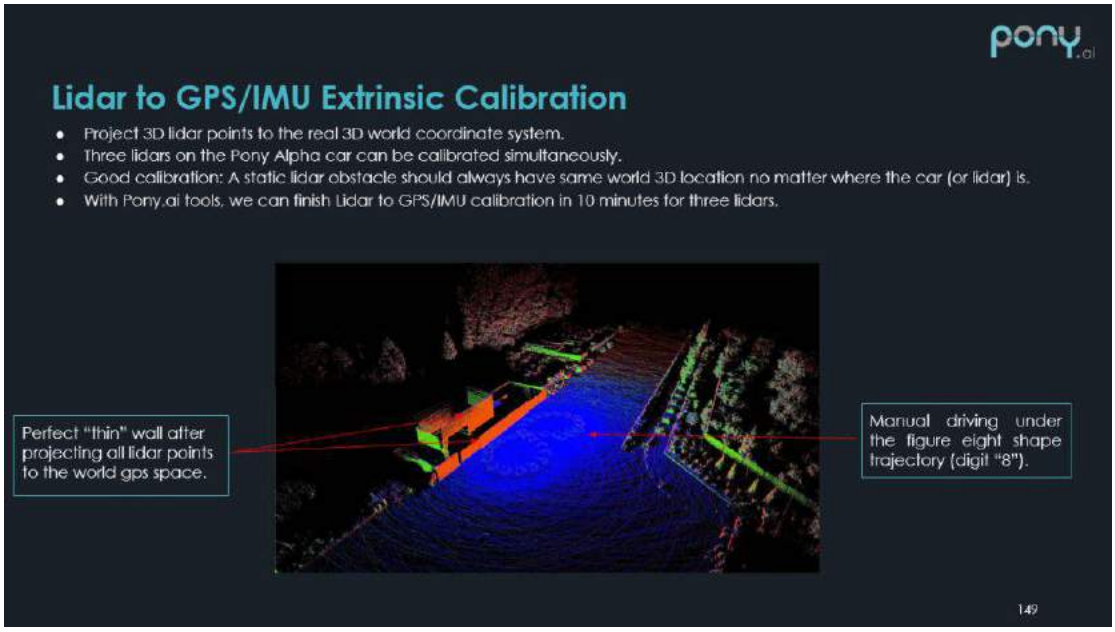


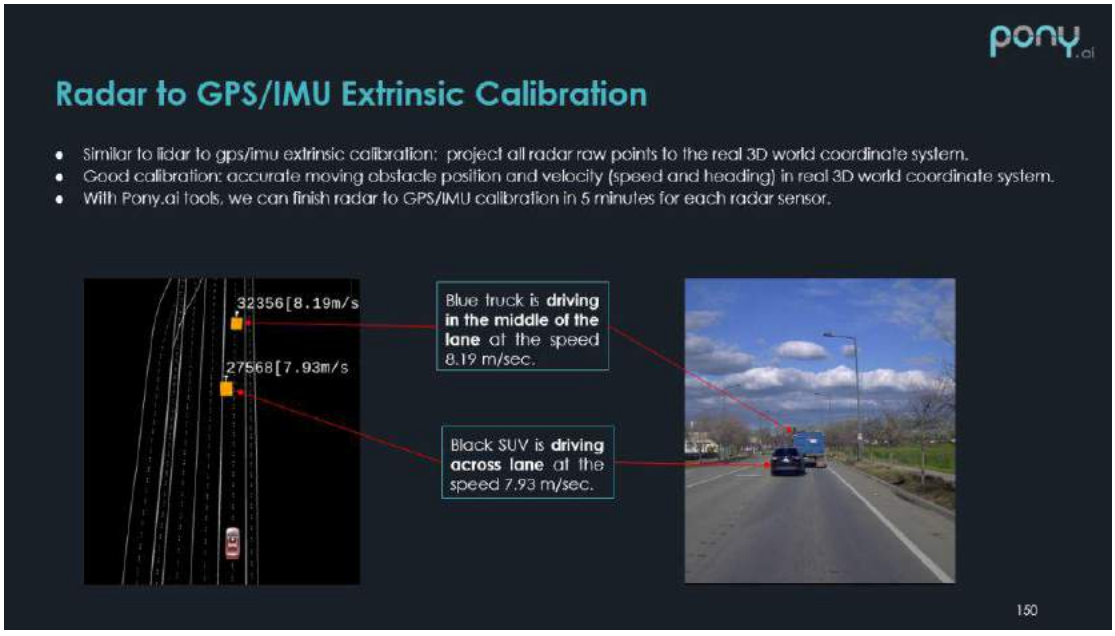
首先是相机内参的标定，内参的标定平台对每一个相机能够在两到三分钟之内做到传感器的标定。



这个图是相机到激光雷达的外参标定，激光雷达是 360 度旋转的方式，每旋转一周是 100 毫秒。相机是某一瞬时曝光的问题，所以涉及到时间同步的一个方式，通过 Lidar 来触发相机曝光。比如说我们有 4 个相机，能够通过激光雷达去保证时间同步。

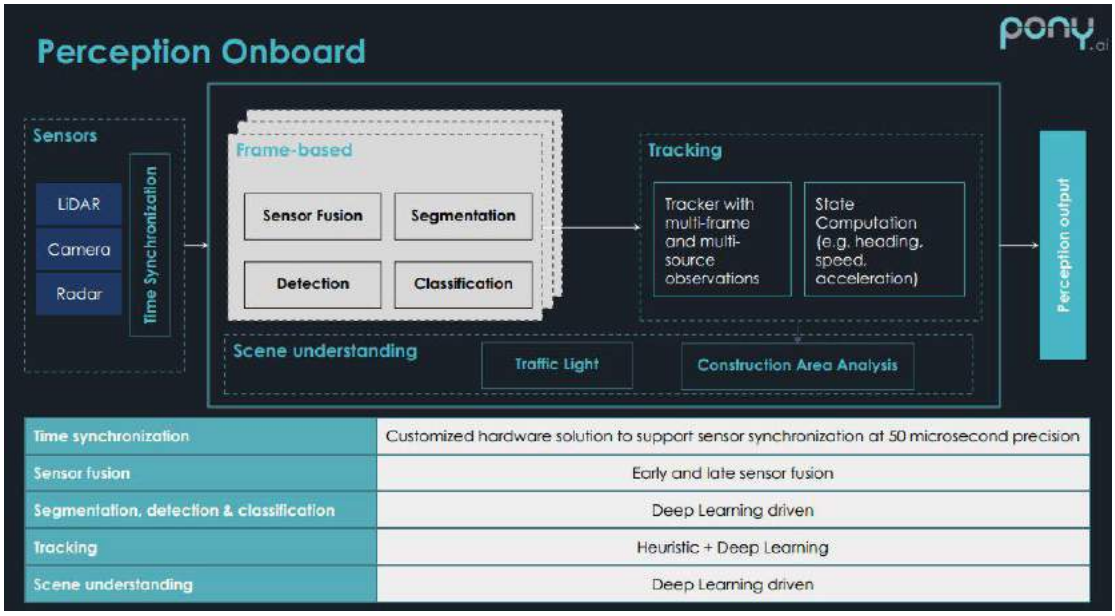
3D 和 2D 彼此互补，二者更好的融合能够对感知会有更精确的输出。





3. Perception Onboard

上面大概介绍了整个 perception 的 sensor 的 setup，以及 sensor fusion 做法。接下来介绍车载的 Perception Onboard 的架构是什么样的，以及解决方案是什么。



这是整个 Perception Onboard 的架构。首先用 LiDAR，Camera，Radar 三种 sensor 数据通过时间同步，所有的时间误差控制在 50 毫秒以内。结合 sensor 的数据，进行 frame wise 的 detection 及 classification 等计算，最后利用多 frame 信息进行多目标跟踪，最后将相关结果输出。这里涉及到 Sensor Fusion, Deep Learning 相关的技术细节，这里不做过多的讨论。

整个 Perception 系统的解决方案要保证这 5 点：

1. 首先是安全，保证近乎百分之百的 detection 的 recall

2. Precision 要求非常高，如果低于某个阈值，造成 False Positive，车辆在自动驾驶状态下会非常不舒适

3. 尽量输出所有对行车有帮助的信息，包括路牌，交通信号灯及其它场景理解的信息

4. 保证高效的运行，能够近实时的处理大量的 sensor 的数据

5. scalability 可扩展性也很重要，deep learning 依赖大量数据，整个模型的泛化能力如何都是非常重要的。能够让我们的 model 和一些新的算法能够去适配更多的城市和更多的国家。

4. Perception Technical Challenges

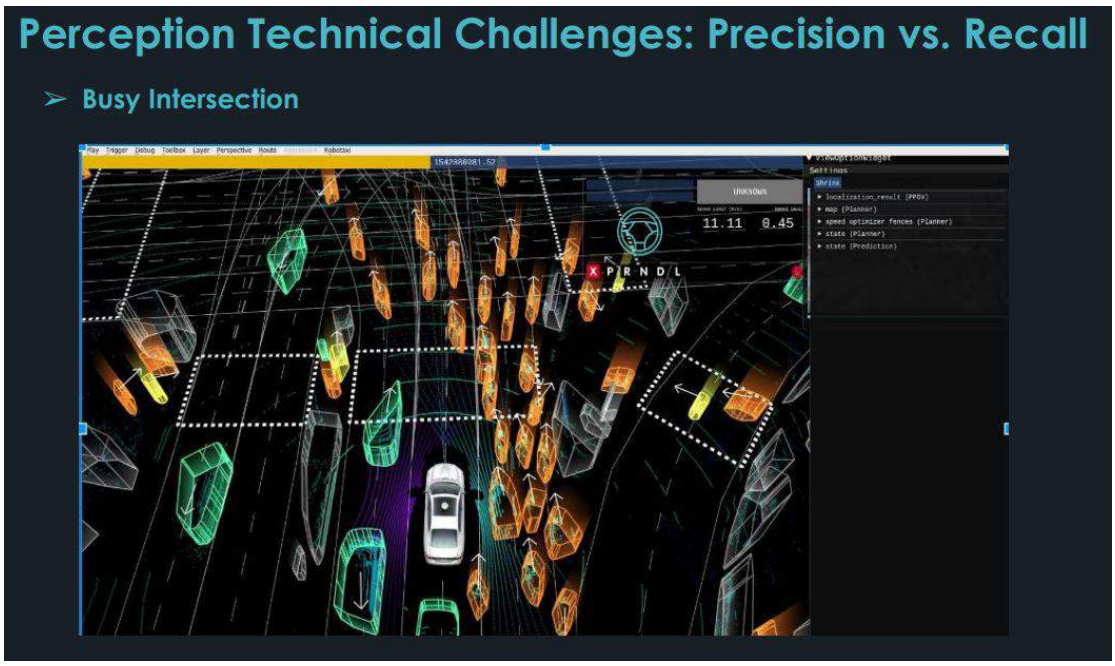
下面分享有挑战性的一些场景：

第一部分：precision 和 recall 的 balance；

第二部分：长尾场景。

这是晚高峰的十字路口的繁忙场景，有大量行人摩托车穿过十字路口。

通过 3D 点云数据能够看到的对应的原始数据。

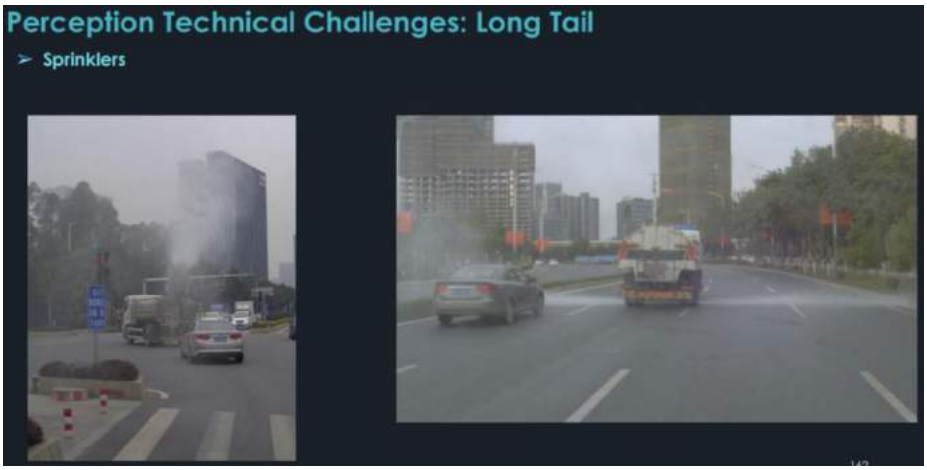


这是目前我们 Perception 的输出，正确的 segmentation 结果和类别。

雨天的问题。自动驾驶系统在一些特殊的或者恶劣的天气条件下，处理是非常难的。

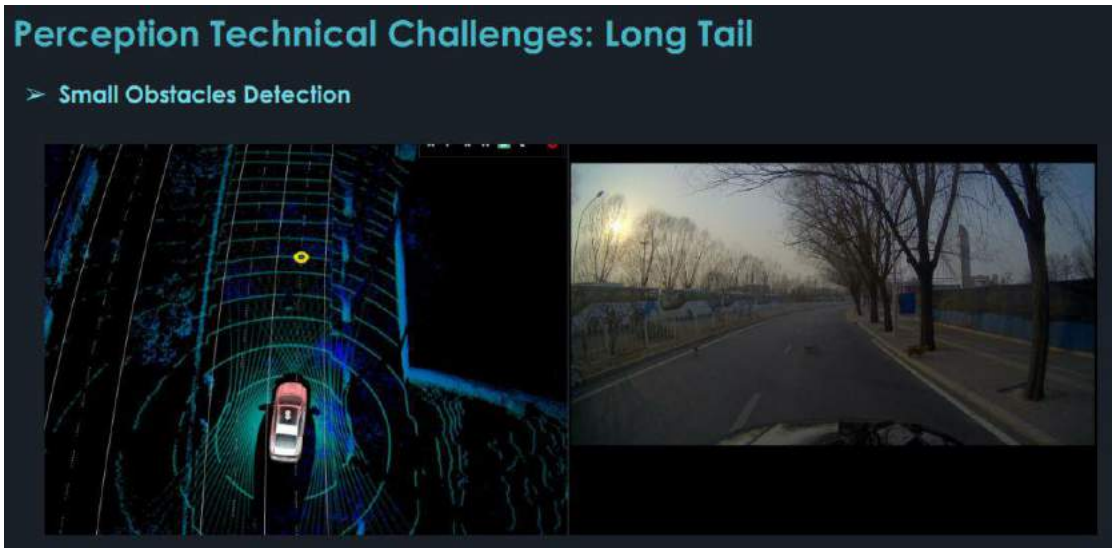
比如，激光雷达是能打到水花的。图中白色的是对水花的 filter。如果不能对水花进行准确的识别和过滤，会对自动驾驶车辆造成麻烦。这里看到我们目前系统的处理结果，结合 Lidar & Camera 的数据，对水花有很高的识别率。

长尾的问题

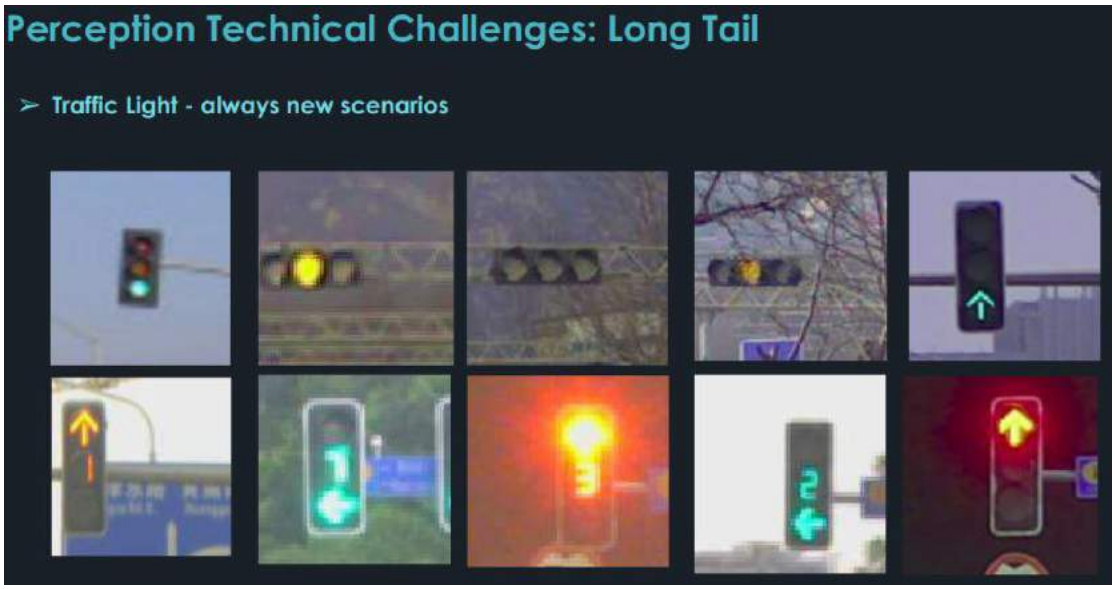


这个是我们在路测时，遇到的两种洒水车。左边是向上喷的雾炮，右边是向两侧喷的洒水车。人类司机可以很容易超过洒水车，但是对于感知系统来说，需要花很多的时间去处理和识别这类场景和车辆，最后让自动驾驶车辆在遇到类似场景获得更好的体感。

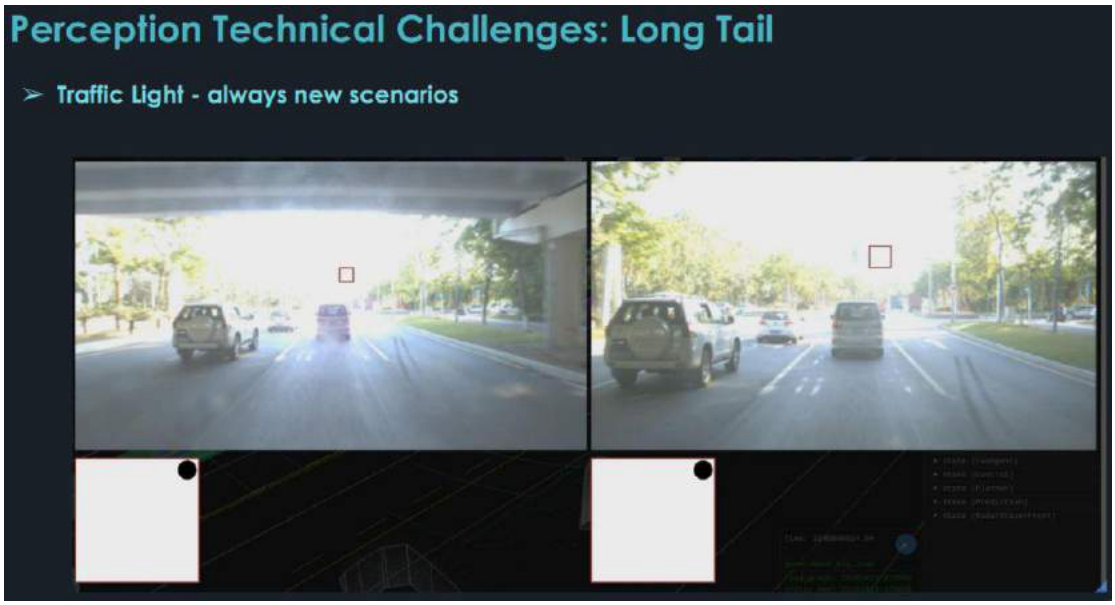
小物体的检测



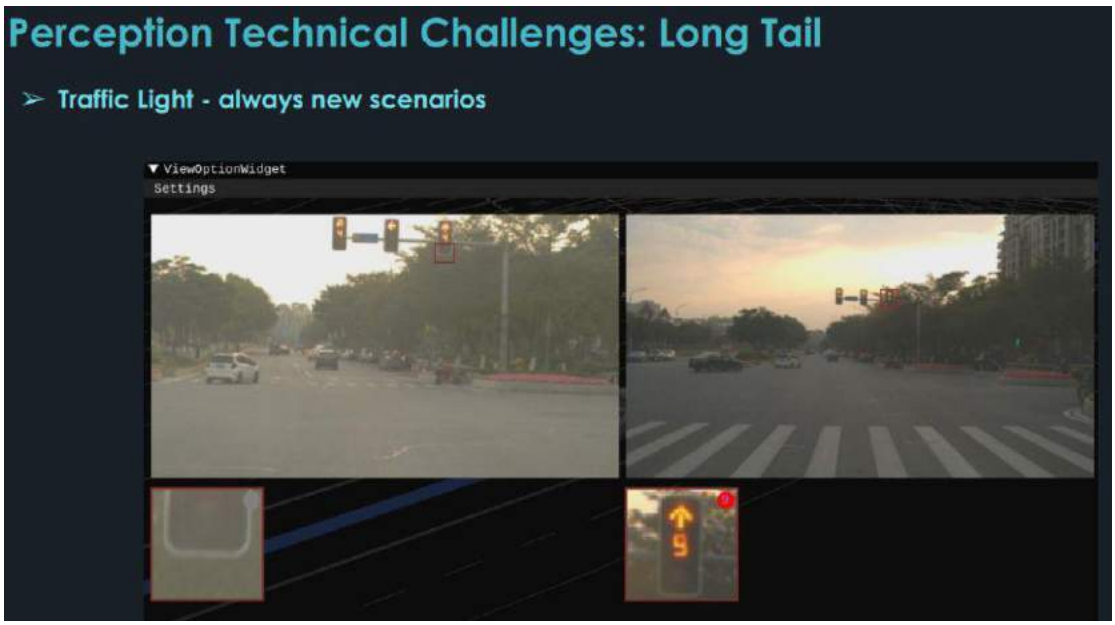
意想不到的事件，路测时比如流浪的小猫小狗的突然出现，我们期望感知系统能够对小物体能够有准确的召回。



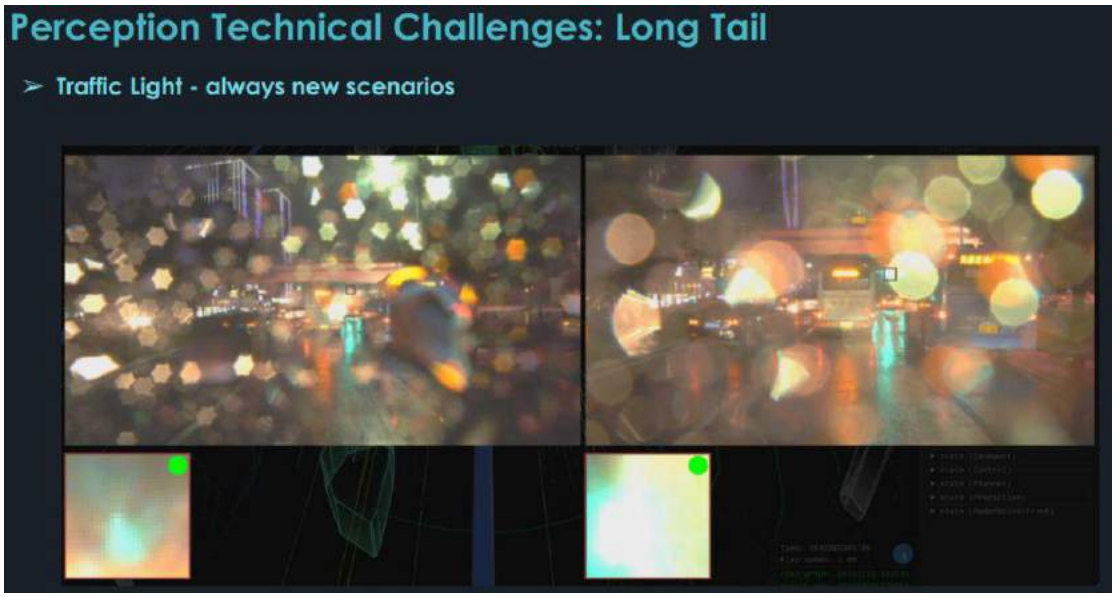
对红绿灯来说会更有挑战。会一直遇到新的场景。因为在不同的城市或国家会遇到各种各样的红绿灯。



逆光的问题，或者突然从桥洞下边出来相机曝光的问题。通过动态的去调整相机曝光等方式去解决。



这个也是红绿灯的场景，红绿灯上有倒计时，我们需要识别出倒计时，能够让自动驾驶车辆遇到黄灯时，乘车体验更好。



雨天摄像头的防水问题，也是处理极端气候条件下所必须具备的。

红绿灯进进度条的识别，绿等快变黄要减速。

图中部分图片来自互联网，只用做交流学习，若涉及侵权，请联系 DataFun 删除。

高精地图和定位在自动驾驶的应用

嘉宾：梁基宏　编辑：HohXil

本次分享聚焦于高精地图在自动驾驶中的应用，主要分为以下两部分：

1. 高精地图 High Definition Map

   ✧ 拓扑地图 Topological Map / Road Graph

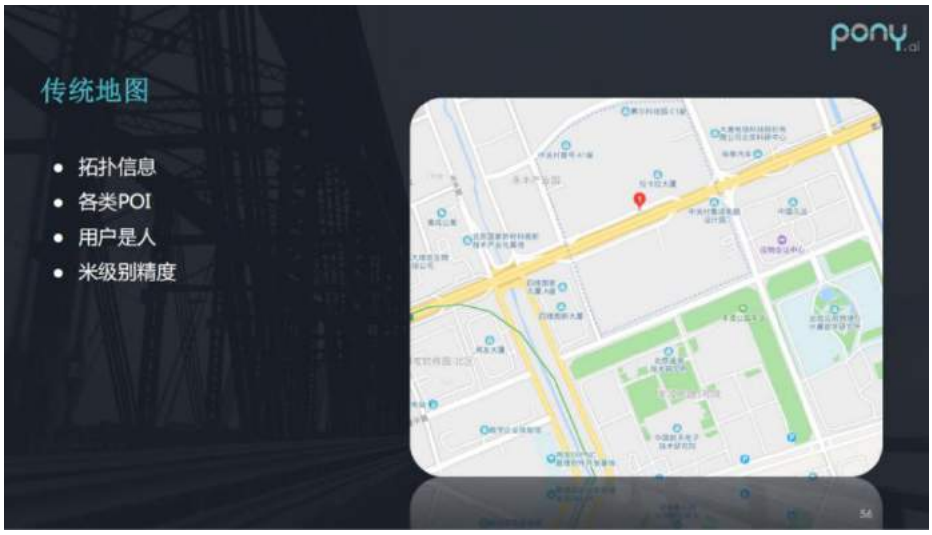
   ✧ 3D 栅格地图 3D Grid Map

2. 定位 Localization

一．高精地图 High Definition Map

★ 拓扑地图 Topological Map / Road Graph

1. 传统地图



我们先看下传统地图，这是从百度地图里截出来的图，从这张图我们可以看到很多信息：

- 拓扑信息：我们可以看到北清路的主干道，可以看到北清路可以延伸到哪些道路上去，这些拓扑信息从人的角度来看是能够识别出来的；
- 各类 POI :我们同时还能看到许多 POI 信息，包括中关村壹号-A1座，拉卡拉大厦等；
- 用户是人
- 米级精度：精度是米级别的，这对人来说足够了，因为人会自动判别周围的环境，做出自己的变化。

但是这些信息对于自动驾驶的车辆来说，到底是不是必要的、准确的或者足够的，那我们继续往下看。

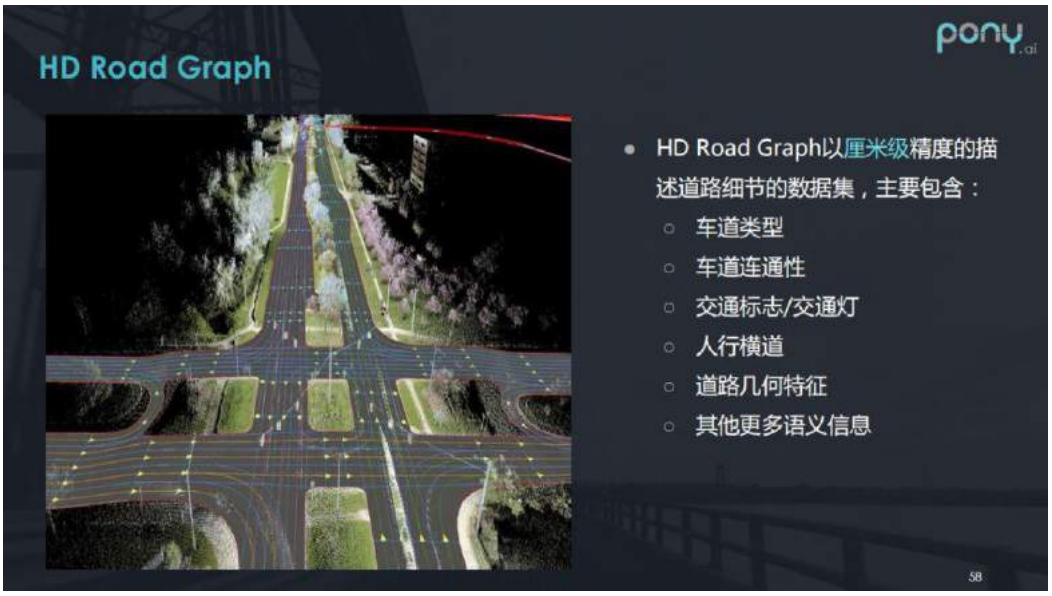
2. 典型驾驶场景

这是一个典型的驾驶场景，假设我们正在驾驶的车辆是后面的白车，可以看到他的前面有辆自行车，那么对于白车来说，他可以有俩个决策：跟车或者变道。那么对于自动驾驶车辆来说，我们作出决策的前提第一个是要知道全局的路线规划信息，变道后是否还能够到达终点，此外还需要知道很多具体的信息：

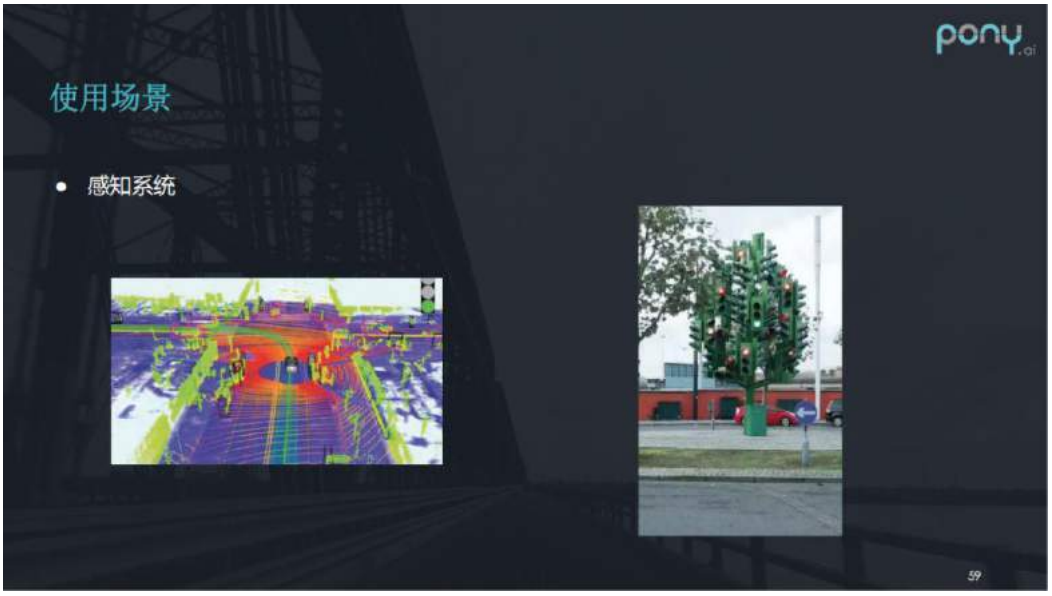
- 在哪条车道上以及当前车道的具体位置
- 是否有隔壁车道，如果没有车道你是变不过去的
- 是否允许变道：车道的类型，虚线实线，变道距离是否足够长

自动驾驶车辆还不能够像人一样做出各种复杂的动作，这时需要有一些先验的约束条件，这些信息的话都应该从地图中来。

3. HD Road Map

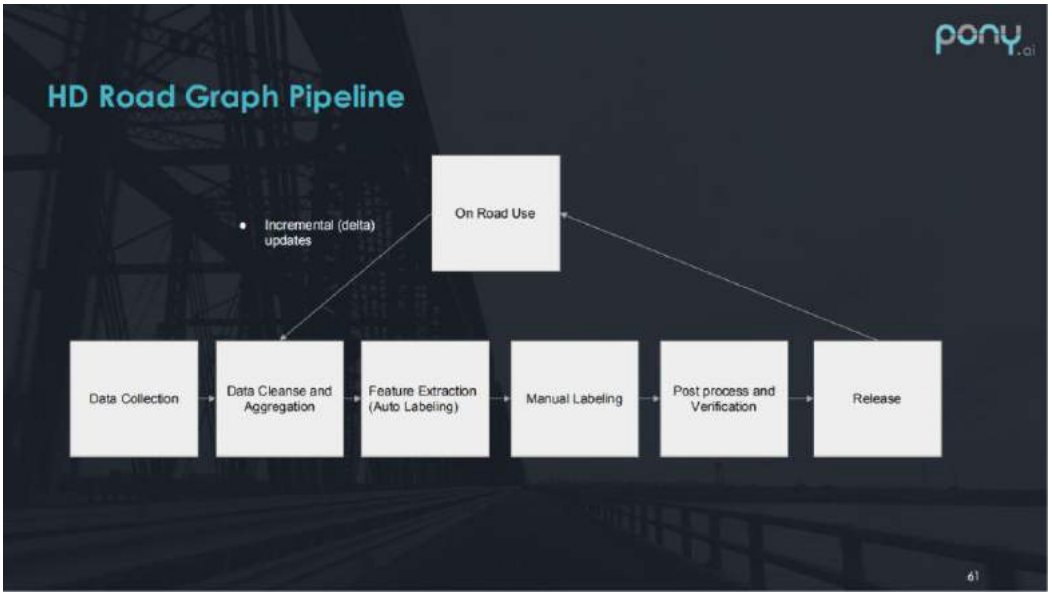


HD Road Graph 也就是高精地图其中的一部分,又叫做拓扑地图。HD Road Graph 以**厘米级**精度的描述道路细节，主要包含：车道类型、车道连通性、交通标注/交通灯、人行横道、道路几何特征、其他更多语义信息。大家可以看下左边这张图，对里面的每一条路，也就是路级别，我们已经精细到每条车道，包括直行的车道，左转的车道，也包括周边的一些几何信息，总体上你会发现高精地图上的信息要比传统地图大很多。



高精地图在自动驾驶里有很多应用，我们先看一个应用场景：在自动驾驶里有感知系统，那一个典型的应用是我们会在地图里面描述红绿灯的位置，自动驾驶车辆到达路口后，可以根据当前的位置，去正确的识别当前的红绿灯到底是红色还是绿色，左边我们可以看到是一个普通的场景，但是右边这种特殊的场景有几十个红绿灯(现实情况基本不存在，只是用来举个例子)，对人来讲也很难去知道在哪个路口应该去看哪个红绿灯，那如果我们在离线地图上能够把上面的信息验证准确的话，对自动驾驶的安全性、可靠性来说是起到很大帮助的。

我们再来看下决策规划，刚才提到的一个典型场景，你需要知道一个全局的路径信息，包括变道所需要的必要信息，那对右边来说，这里主要是强调绿化带，我们这里提供的信息，前面有提到过叫做语义信息，我们为什么要在地图里包括这些语义信息呢？如果自动驾驶车辆可以提前知道旁边是绿化带，那么他可以帮助自动驾驶车辆做一些先验的决策，他可以知道在那个时间段不会有人或者车直接从对向车道穿过来。所以从准确性来讲他可以得到很大的提升，对性能来说也会有很大的提升，因为你不需要过多关心对面车道的物体，很明显你的计算就会少，你会发现整个先验信息对决策来说必不可少，并且是非常关键的。



我们再来看下我们当前整体用到的 Pipeline，分为以下几个环节：

- 数据收集，地图信息的采集
- 数据清洗/聚合
- 自动化识别，把地图里所需要的必要信息处理出来，包括车道线、红绿灯、十字路口等等各种信息
- 人工检查/标注，现有的算法我们还做不到 100%的自动抽取，所以还会加入人工检查和标注的环节
- 后处理和验证，整个数据出来之后，我们需要有后处理和验证，因为数据出来之后怎么样去保证你提供的信息是准确的，这是我们需要解决的问题，因为不能单靠人或者高成本的方式来验证道路信息是否准确，假

设一个红绿灯位置标错了，或者自动识别错了，那可能会导致你在路测过程中会出现一些安全隐患，所以整个验证环节占有很重要的部分

- Release，整个验证完之后，我们会进行 Data Release 数据的发布，数据会纳入统一的管理中，每一版 release 的地图的质量已经得到比较好的验证

这是一个常规的发布流程。现实中整个路网结构是不断往外扩张的，且周围环境会有变化，所以我们也会有增量的更新。

前面简单介绍了整个 HD Road Graph 的 Pipeline，我们来看下 HD Road Graph 里面遇到的问题：

成本

- 厘米级别的精度：需要各类传感器（LiDAR、cameras、GNSS/IMU），右边是苹果的一个地图车，大家可以看到上面装了很多传感器，很明显制造一辆高精地图车辆的话成本非常昂贵

- 数据量：海量存储资源和计算资源

- 覆盖范围：从局部区域逐步扩张，从中关村壹号附近地区，扩张到海淀区，再到北京市，甚至是半个中国，这里面的硬件成本和人力成本等，都会极剧增加

那么如何去解决成本问题？这需要我们在工作中想一些好的方式。

挑战-复杂性

第二个是复杂性带来的挑战，对于我们整个 Data Model 来说会包含三部分信息：

1. 语义信息，从右边的图标志牌来看，你会发现有很多的交通标记，如何提高准确性，并且尽可能提高覆盖度

2. 空间信息，右图中的立交桥是西直门立交桥，我们会发现通过导航软件或者老司机的经验都未必能够在某一条车道上找到准确位置，对我们来说就是如何能够把整个空间信息描述好，然后集成到地图中去

3. 时间信息，在北京有一些潮汐街道，早高峰和晚高峰走的方向是不一样的；对一些公交车道在早高峰的时候只能公交车去跑，自动驾驶车辆是不能跑的，很明显需要把这些信息包含在地图里面

这样的话，决策模块和感知模块才能去准确的使用。

挑战-扩展性

1. 自动驾驶车辆行驶 1KM 会产生 GB 级别的原始数据，处理这些数据在功能架构上会面临很大的挑战

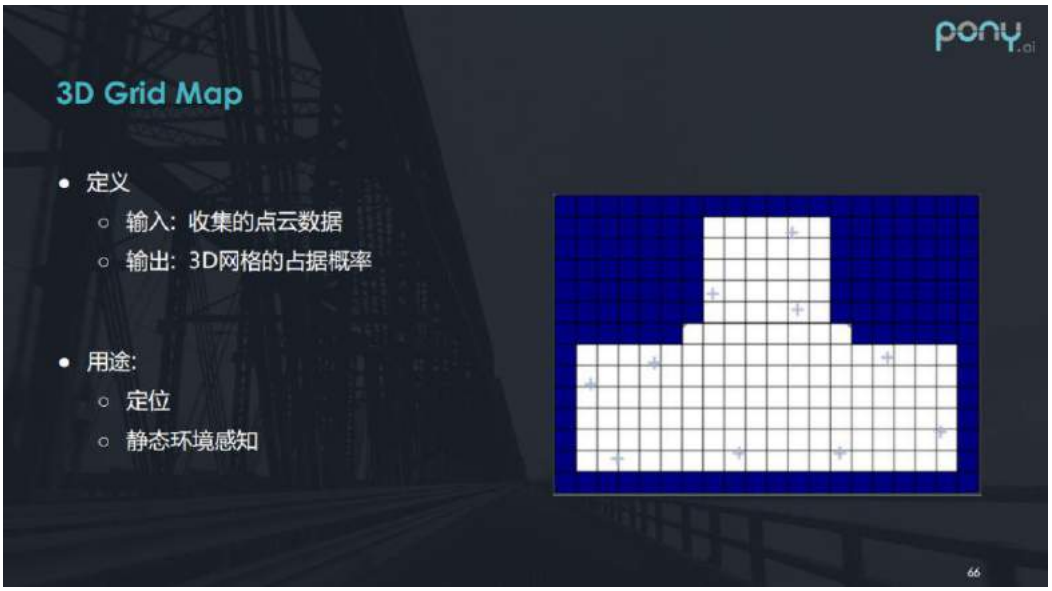
2. 更新，从天级别到小时级别再到分钟级别的更新，整个功能架构是不一样的，比如我需要对北京市地图做小时级别的更新，到底能不能做到，这需要很多的优化功能在里面

挑战-实时性

交通管制、环境变化等怎么反馈到 HD Road Graph 上，如果今天某条路修路了，对于人来讲我就可以直接绕过这条路，而这种信息怎么去反馈到自动驾驶里呢？是靠人去反馈，还是系统自动去识别，即使反馈之后，怎么实时更新到地图数据里，我们需要打通 offline 到 online 的环节，及时更新到我们正在运行的自动驾驶车辆里去。

★ 3D 栅格地图 3D Grid Map

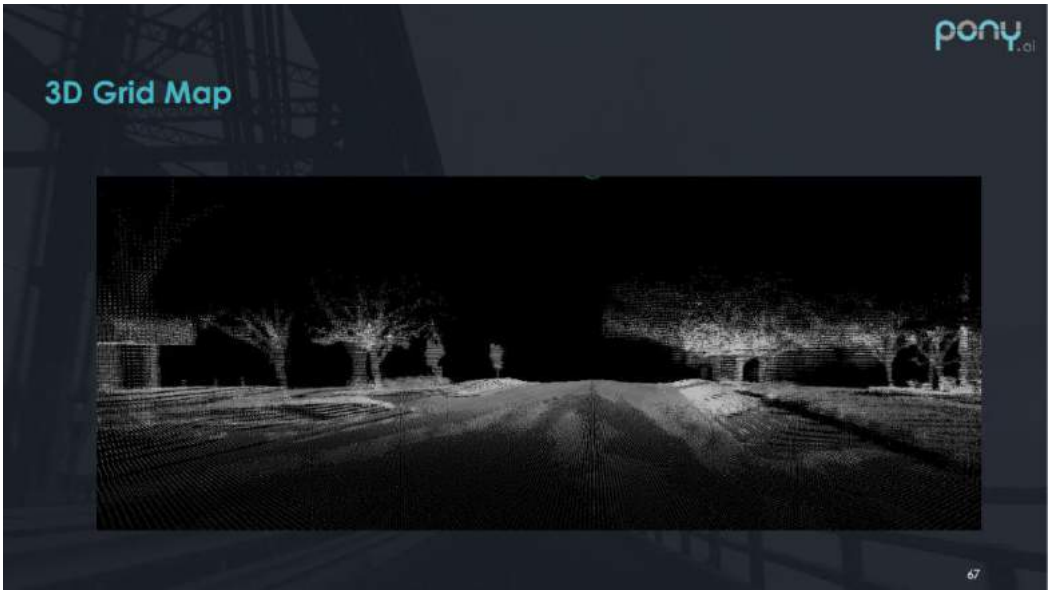
1. 3D Grid Map



前面介绍了拓扑地图的部分，下面介绍我们的 **3D 栅格地图（3D Grid Map）。**

定义：我们认为一个真实的三维世界可以用一个概率模型来表示。比如像今天这个会场，我们会把他离散化成一个个立体方格，对每个方格里面我们会存储一个概率，这个概率是说这个空间不为空的概率。

- 输入：收集的激光点云数据
- 输出：3D 网格的占据概率
- 用途：
  - ✓ 定位，这个稍后会讲
  - ✓ 静态环境感知，减少工作量，专心识别动态环境



这是我们做的一个 3D 栅格地图，他并不是一个连续的描述，从数学的角度讲，已经把他离散化，我们会保留他的一些概率值和一些属性。展现出来的

话，他是一个厘米级精度的栅格,从这个图我们可以很清楚看到周围的环境，包括一些树、建筑之类的。

2. SLAM

那么我们一般会怎么去做高精地图呢？在传统上有一个方式叫 SLAM，如果是做这个方向的同学会比较清楚，但是对于做工程的同学来讲这个概念可能是第一次接触。他是指运动物体根据传感器的信息，一边计算自身位置，一边构建环境地图的过程。目前，SLAM 的应用领域主要有机器人、虚拟现实和增强现实。其用途包括传感器自身的定位，以及后续的路径规划、场景理解。

我们来看下传统和自动驾驶 SLAM 的区别：

传统 SLAM：

- 2D
- 视觉传感器，单靠摄像头，在某个空间转来转去，然后通过一些算法来建模
- 没有 GPS 信息，很多场景在室内的话是收集不到 GPS 信息的
- 生成地图和定位同时进行

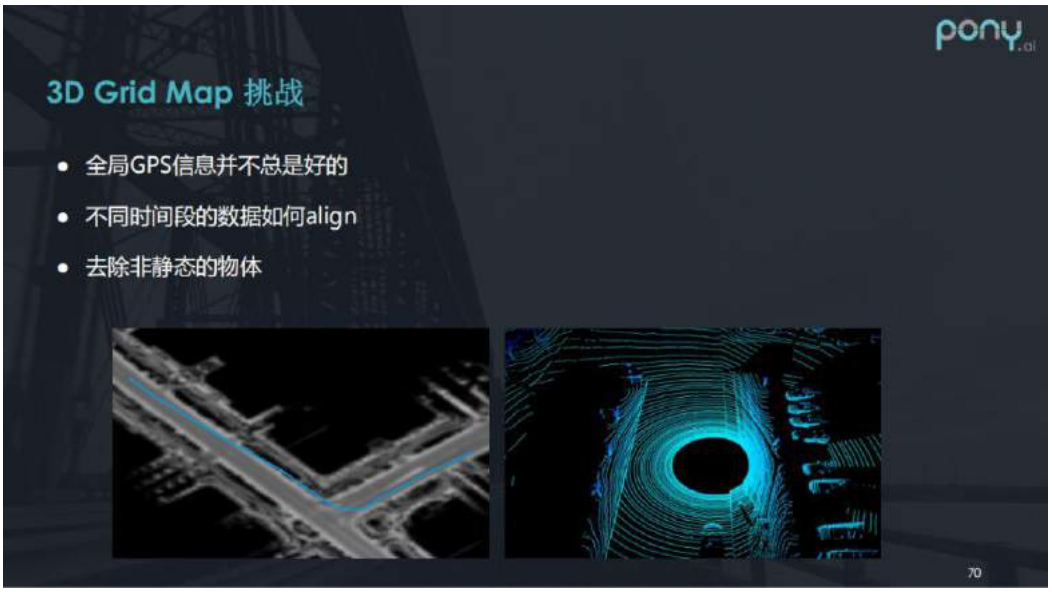
自动驾驶：

- 3D，他描绘的是一个三维环境

- 激光传感器，我们需要达到比较高的精度，所以我们一般采用激光传感器

- GPS 信息，室外的话我们有个优势就是很多时候我们是有 GPS 信息的

离线处理，最后我们的数据不需要在线处理，前面介绍过，在线的话我们有一些场景算法是不能很好的处理，所以我们需要抽取一些信息来做离线的验证。



相比于传统 Slam，我们来看下有哪些问题是需要解决的：

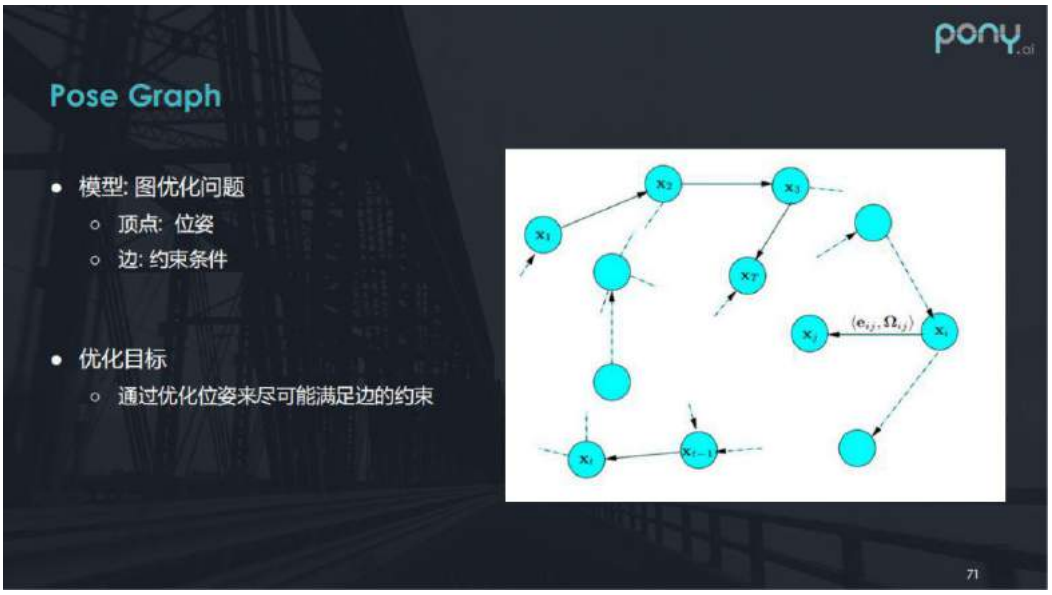
- 全局 GPS 信息并不总是好的，左下角的图，大家可以清楚的看到，这个是我们建出的底图，里面有当时车辆的一个行驶轨迹，你会发现如果根据 GPS 的行驶轨迹的话，车已经开到马路牙子（路肩）上了，跟实

际的情况是不一样的，当你拿到这样一份数据，你在算法层需要做哪些优化，把细节做好，这是需要我们考虑的一个问题

- 不同时间段的数据如何 align（校准），因为整个地图的采集不可能是一次就完成的，你可能需要今天采集一次，明天再去采集一次，这就需要我们在算法层面上考虑如何把不同时间段的数据 align 到一起

- 去除非静态的物体，我们建的是一个概率的模型，对于很多非静态的物体我们是不需要保留的，我们只需要保留一些静态的物体给感知系统去使用，所以我们在处理完数据之后，需要把一些非静态的物体去掉

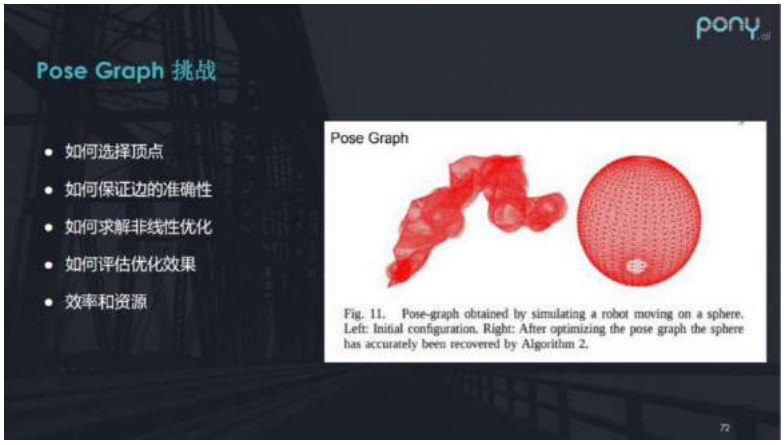
3. Pose Graph



整体上我们可以认为这是一个优化问题。如果我们把每一次采集的路线，按照时间去划分，比如每隔 4 分钟采一个点，我们可以得到不同的位置，举个

例子：我们在大厅里走来走去，我现在站在这里，一分钟后我可能在那里，再绕一圈的话可能站在门口，我们把整个 GPS 得到的位置信息离散化之后，就可以得到一个个点，很明显我们把当时的点所看到的周围的环境直接拼起来，就可能得到一个三维的地图，因为每个点看到的角度都是不一样的，那我们把不同位置的点都拼起来就可以了。但是直接拼起来会有很多问题，因为原始位置可能跟真实位置有偏差，所以我们把这些点离散化之后，希望能够把所有点都调整到准确的位置，那我们怎么去把最终的结果求正确？因为对整个图来说，除了顶点之外，还有边，我们希望通过边的约束把点调好，边的约束你可以认为是个相对约束，假设我知道下个点的准确位置，那么我就可以倒推我这个点的准确位置。就是在有准确的相对位置的基础上，加上一些比较少的绝对位置，我们直接可以得到一个全局的准确位置。

优化目标：我们把整个图离散化到一个点之后，我们会尝试建立一些边的约束，最终我们是希望通过优化一些点的位置来满足边的约束，最终会转化为一个优化问题。

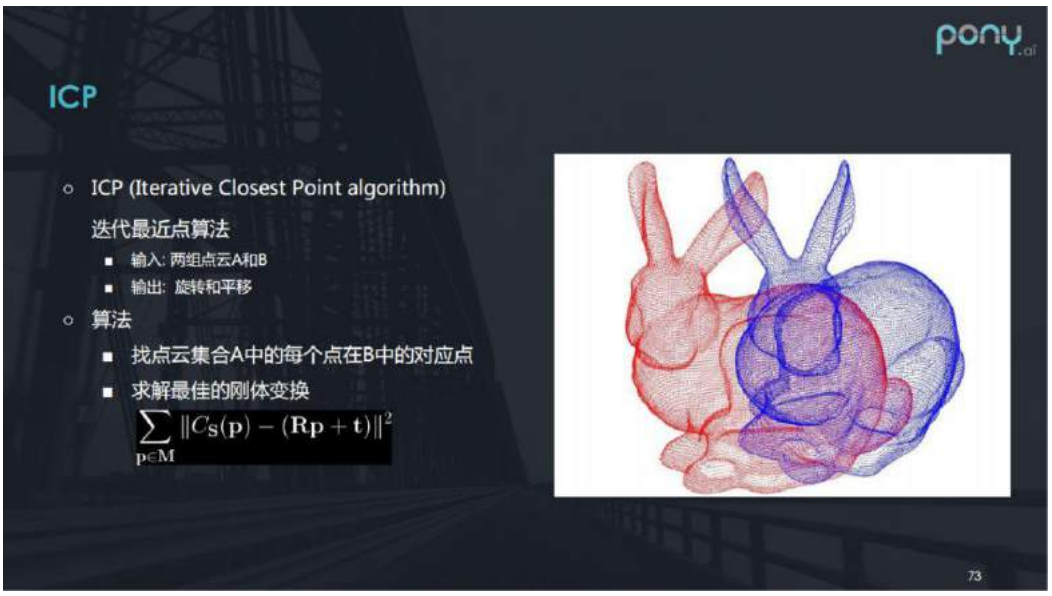


整体的数学模型看起来是比较简单的，但是在整个细节里面是存在很多问题需要去做优化的：

- 如何选择顶点，比如是不是毫秒的点都要加进去，GPS 信号不好时的点要不要加进去
- 如何保证边的准确性，一个顶点到另一个顶点的相对位置，我要通过什么方式去建立
- 如何求解非线性优化，图建好之后如何去做非线性的优化
- 如何评估优化效果，是否和真实的环境是一致的呢？如何评估我们的结果满足要求
- 效率和资源，对于工程来讲，数据量这么大，怎么去设计工程算法，能够保证我们的效率和资源，比如说，是不是需要在算法基础之上去做一些定制优化，用 GPU 或者分布式的计算方式，这都是做工程时需要考虑的问题

上图右边其实是一个例子，左边是原始的图，有一个物体来回在球面上运动旋转，边与边之间我们已经建立相对约束，最终优化完之后，你会发现整个环境的重建就变成了一个比较理想、完整的球体，概括来说我们要做的事情就是需要把左边的图来变成右边的图。

4. ICP

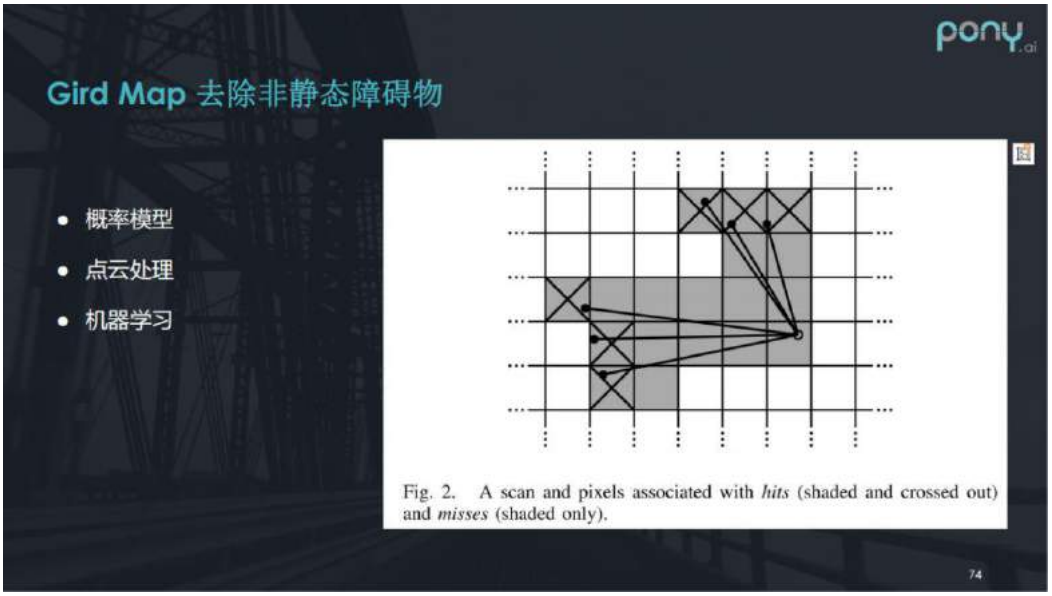


我们前面提到我们要保证边的约束，那我们怎么知道求出这个约束，业界用的比较多的传统的经典算法叫做 ICP（Iterative Closest Point algorithm）迭代最近点算法。

举个例子：左边是一个红色的兔子，右边是一个蓝色的兔子，现实中这俩个兔子的位置是一样的，但现在俩个兔子是不一样的，说明位置存在偏差。他的输入是俩组点云 A 和 B，输出是俩组点云之间在空间上的旋转和平移。

具体算法的思路大致是：找点云集合 A 中的每个点在 B 中的对应点，通过求解最佳的刚体变换，不断的迭代优化，最终得到一个收敛解。

5. 去除非静态障碍物体



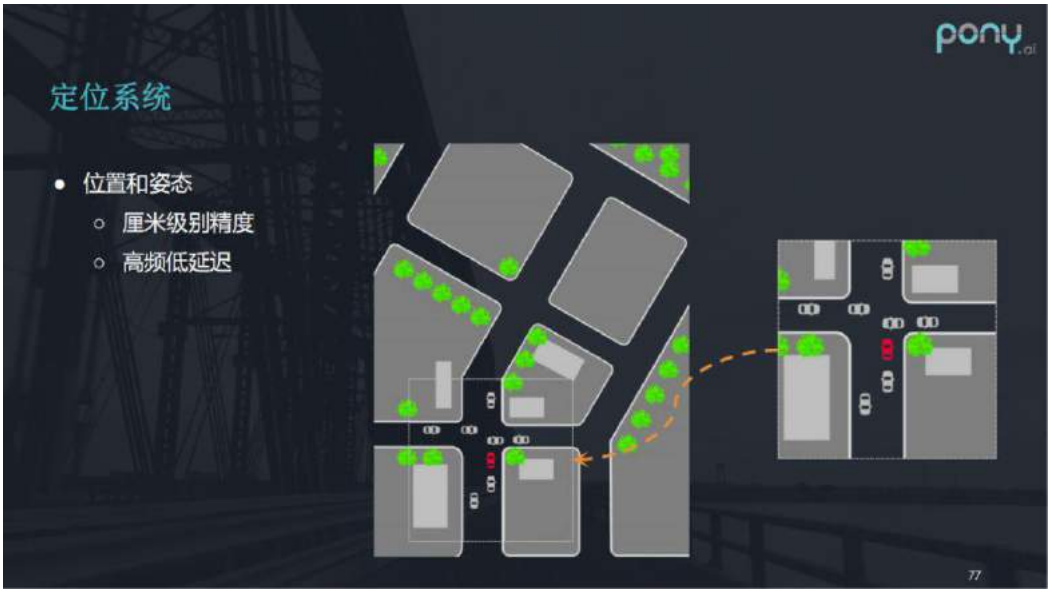
概率模型

- 点云处理，单纯的概率模型是不够的，比如有时候小区门口经常是有车的，但是在某时刻他是开走的，只是恰好在我们采集数据的时候这些车都在，所以我们需要依赖离线点云处理

- 机器学习，同时也可以利用机器学习算法去把非静态的物体从地图里面抠出来

这是一个简单的对比图，没有做处理的话，你会发现右边是有一些车的轮廓，当从空间上去除之后，会变得比较清晰。

二．定位 Localization



接下来简单介绍下定位，定位大家都比较熟悉，对于定位系统和传统定位来说，他的位置和姿态有如下特点：

- 厘米级别的精度，需要精确到车在车道线的具体位置离左右边线几公分
- 高频低延迟，需要毫秒级别的时间延迟，告诉我的车在什么地方，否则会出现安全事故

定位系统的几种定位方式：

全球导航卫星系统 Global Navigation Satellite System

特征：

- 米级别精度
- 低频
- 干扰或者遮挡

解决方法：RTK（Real - time kinematic，实时动态）载波相位差分技术，可以达到厘米级别。

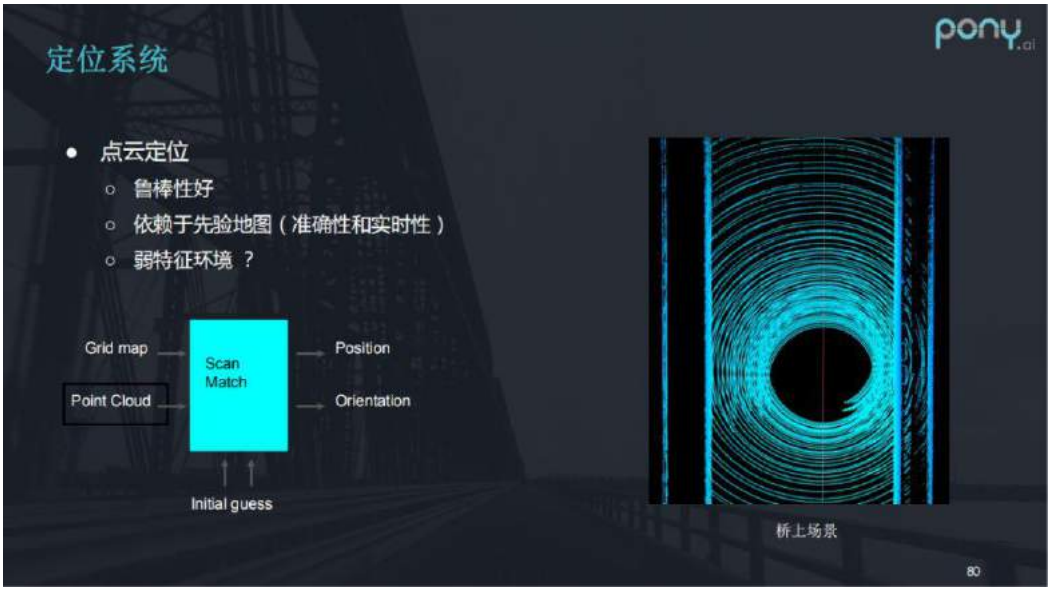
相对定位：惯性测量单元 IMU（Inertial Measurement Unit）

特点：

- ❖ 加速度以及角速度
- ❖ 三个方向的加速度计和三轴的陀螺仪
- ❖ 高频（通常 100HZ）
- ❖ 累积误差

解决方法：惯性导航系统 Inertial Navigation System

点云定位



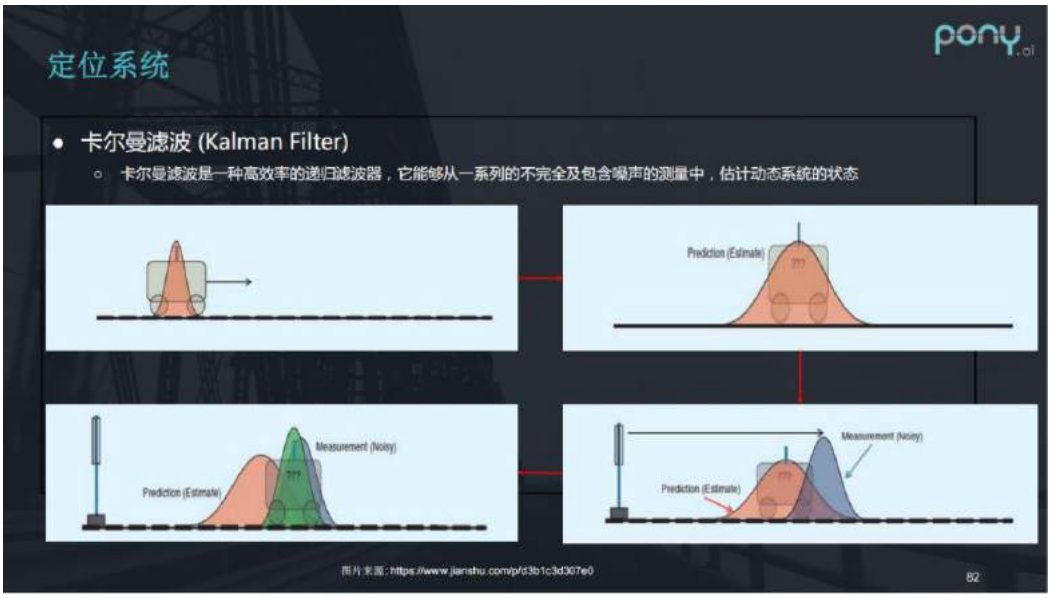
特点：

1. 鲁棒性好，不依赖于 GPS 信息

2. 依赖于先验地图，那么这个地图的准确性和实时性会导致一个定位误差

3. 弱特征环境？有些弱特征环境是需要解决的，比如说在桥上，往前开 10 米和往后开 10 米看到的场景都是类似的，对于这类环境来说，我们如何去找到准确位置，这是需要考虑的一个问题

综合来说，我们单靠单一传感器是不够的，所以采取的是多传感器融合的方案，整体效果是会有更高的精度和更好的鲁棒性，然后我们整体上是基于卡尔曼滤波模型。



简单介绍下卡尔曼滤波（Kalman Filter）模型：

卡尔曼滤波是一种高效率的递归滤波器，它能够从一系列的不完全及包含噪声的测量中，估计动态系统的状态。

我们把整个当时的位置认为是高斯分布，对这个小车来说他大概在某个范围之内，在中间的可能性是最大的，然后我们会有个预测（预测是根据自己的运动方程），比如我往前走了 10 米，他大概在这个位置，你会发现他的误差值会变大，因为可能是 IMU 告诉你走了 10 米，但是传感器本身也是有误差的，那同时假设我有个 GPS，告诉我可能是在蓝色的部分，他也是包含一定

误差的，这样我其实有了俩个位置，那么我就可以把俩者合在一起得到一个比较准确的可信的结果，整体可以认为卡尔曼滤波是不断的在更新迭代的，来最终得到一个比较准确的位置。

最终我们定位系统的目标是万里无忧，对我们的挑战是更低成本和更苛刻的外部环境。比如右边这个图，大堵车，这时候可能 GPS 不行，堵车堵了几个小时，走走停停，会导致累积误差，实时讯号又不好，你可能搜不到，这样的环境你会发现，他只能看到周围的很多车，也没办法跟周围的环境去做对比，知道自己在哪里，这种情况下，我们到底怎么去得到一个比较准确的定位效果，这需要我们把这些问题都解决好。

文中部分图片来自互联网，只用做交流学习，若涉及侵权，请联系 DataFun 删除。

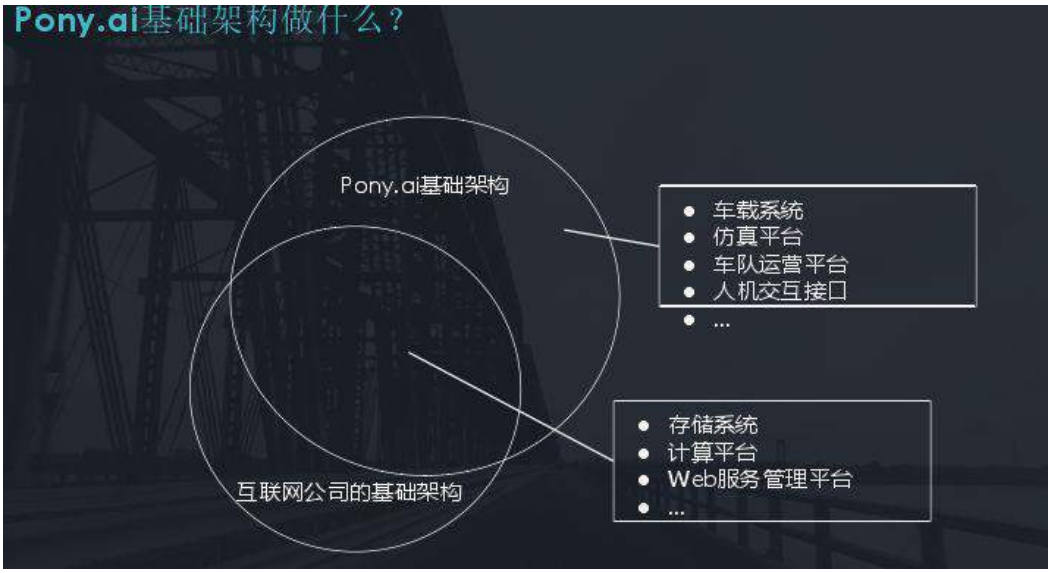
Pony.ai 的基础架构挑战与实践

嘉宾：王轩　编辑：HohXil

本次分享将从以下几个方面介绍：

- Pony.ai 基础架构做什么
- 车载系统
- 仿真平台
- 数据基础架构
- 其他基础架构

1. Pony.ai 基础架构

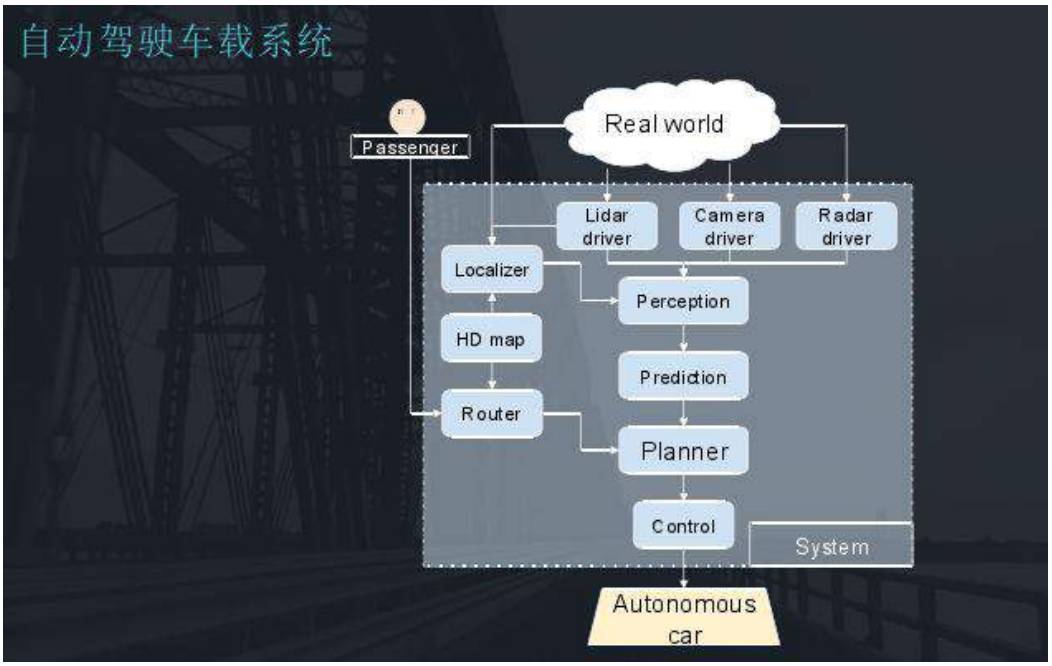


首先给大家介绍一下 Pony.ai 的基础架构团队做什么。互联网公司在系统基础架构在业务扩展时通常会遇到一些通用的技术挑战，比如存储系统、计算平台还有 Web 服务治理。对于 Pony.ai 而言我们是一家做自动驾驶的公司，除了上述提到的互联网公司会遇到的一些技术挑战之外，会有很多自动驾驶技术本身相关的技术挑战，比如大家都可以想到的车载系统、仿真平台。此外，对于 Pony.ai 来说，如果运营一个大规模的自动驾驶车队，需要有一套完善的车队运营基础。自动驾驶需要和人进行交互，因此需要一个人机交互的接口。这些都是 Pony.ai 的基础架构团队正在做的事情。

Pony.ai 的目标是建立一个比较大规模的自动驾驶车队，车队规模在未来会成百上千计，而目前公司正处于自动驾驶车队数量快速扩张的阶段，虽然对其他技术挑战也存在，但是对基础架构的挑战更是巨大的，它要求整个自动驾驶的基础架构要有足够的可扩展性。随着车辆数量的增加，运营区域会增

加、数据量会增加、工程师数量增加，系统各个模块数量会增加、内部代码量也会增加。这些都会给基础架构带来非常多的技术问题。接下来会详细介绍在我们在基础架构演进过程中遇到的挑战。

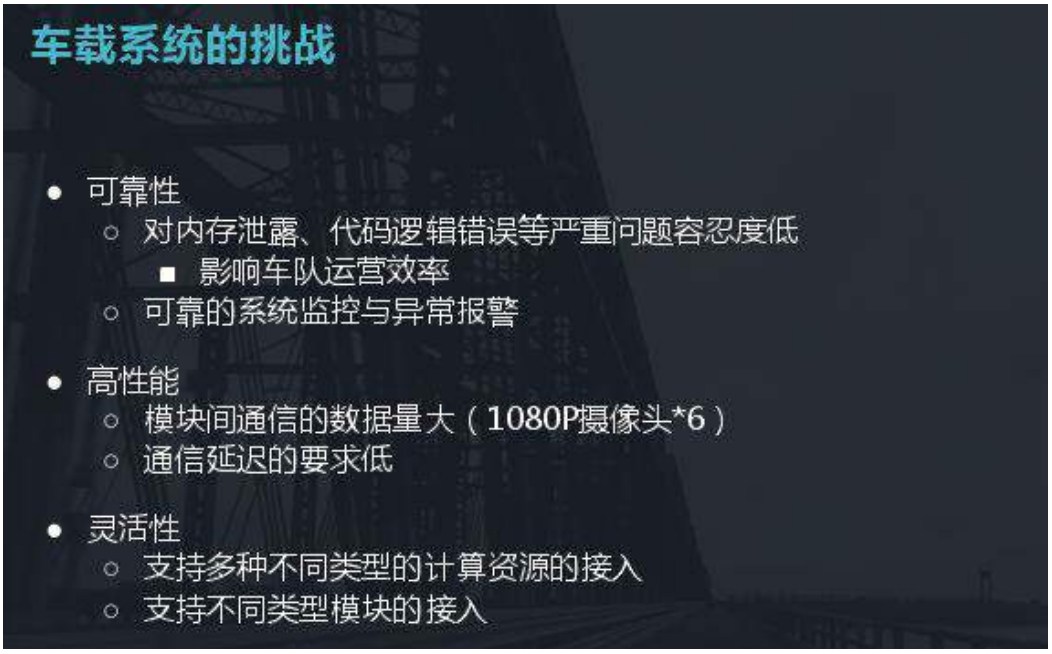
2. 车载系统



首先介绍的是自动驾驶车载系统。那么什么是车载系统呢？一般来说自动驾驶系统上有很多传感器代替人类收集信息，车载系统需要处理这些数据信息，最终转化为对车辆的控制决策，如踩刹车、转多少方向盘，做出决策后传递给车辆。上图是一个自动驾驶系统示意图，每一个方块对应系统中的一个模块。首先，车载系统通过一些传感器去感知真实世界，有激光雷达、摄像头等驱动收集真实世界中的信息，这些信息会传递给 Perception 模块，这个模块会实时感知车辆行驶中周围的环境，比如车辆周围有哪些车辆以什么样

的状态在行驶；车辆周围有哪些行人等。当 Perception 模块收集到这些信息后将其传递给 Prediction 模块，我们不能只依据当前车辆周围环境信息做决策，需要对车辆周围的任何物体或者人、障碍物接下来的行为预计，如前面路口的车是转弯还是直行对接下来车辆的决策很重要。当 Prediction 模块生成这些信息后会传递给 Planner 模块，这个模块会规划接下来车辆一段时间的行驶轨迹、做出什么样的决策。比如前面有行人需要踩刹车等待行人先过去，前面车行驶比较慢可能需要变道超车。当 Planner 模块规划好轨迹后传递给 control 模块，计算出车辆需要进行操作的指标参数，如打多少方向盘，踩多少刹车等，最后传递给自动驾驶车辆。在模块运行中，可能还需要其他一些模块，需要高精度地图、实时定位，还需要 router 模块，当指定起点终点时，需要计算出一条合理的路线。

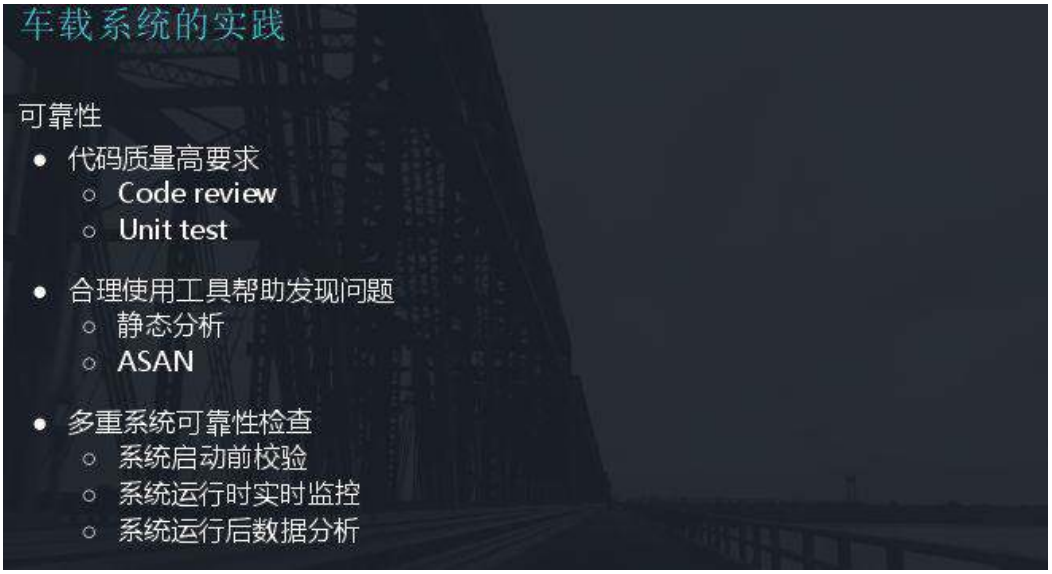
车载系统需要去调度各个模块的运行，模块间的消息通信（一个模块的输入可能是另一个模块的输出），还有车辆上有不同的计算资源，不同的模块可能也需要不同的计算资源，一些感知的算法可能需要很大的 GPU，有些模块可能是 CPU 计算多一些，因此需要系统合理分配不同模块需要的计算资源。日志记录也是车载系统需要提供的非常重要的功能功能，在自动驾驶的过程中，系统需要记录车辆运行过程。这样如果出现一些非预计的情况可以通过日志信息进行 debug，来帮助系统持续演进。此外，自动驾驶安全要求很高，因此系统需要有非常完善的监控和报警系统，当系统出现一点点问题都需要及时报警，使得安全员可以进行正确的反应来避免潜在的危险。



对于车载系统有很多要求，在某些方面可能实现比较困难。首先是可靠性，代码逻辑设计错误会出现各种问题在车载系统可能引起严重后果，因此对这方面的可靠性要求很高。特别是 Pony.ai 正在扩展车队的规模，不稳定的系

统可能严重影响车队运营效率。可靠性也体现在系统监控和异常报警的可靠性要求也很高。第二个系统设计的要求就是高性能，各模块间消息通信的数据量很大。在我们最新一代车载系统中有 6 个 1080P 摄像头 ,当以每秒 10-20 帧的帧率获取数据，每秒会产生的数据量是非常大的。要使这么大的数据量高效在各个模块之间传输，对于设计上的挑战还是很大的。吞吐量之外，模块通信延迟也要求很低，想象一下如果识别到车辆前方的行人之后需要 1 秒才能作出反应，就可能会造成严重的后果了。最后一点是系统需要保持高效的灵活性，需要系统灵活的支持各种不同的模块计算资源的接入，还有不同类型模块的接入。

针对这些系统设计上的需求公司早期决定放弃有名的开源 ROS，自主研发 PonyBrain。做出决策的原因有：首先 ROS 作为开源系统，代码质量可能不受公司控制，不能很好地保证代码质量。另外一点是 ROS 作为一个通用的机器人操作系统，可能不太贴合一些特殊的自动驾驶需求。基于上面这些因素，我们选择自研 PonyBrain 系统。



对于之前提到的系统设计上的挑战，我们也都在工程实践上有一套针对性的措施。首先可靠性挑战，只有高质量的代码才有可能让系统足够可靠。为了提高代码质量，我们在公司执行非常严的 Code review 和 Unit test 的机制来保证代码质量，确保不会有意外状况出现。同时，我们还会使用一些工具来及时发现问题，如静态分析和 ASAN，还有监测内存泄露的工具。在系统运行时也会有多重系统可靠性检查，在系统启动前会校验系统启动环境，系统运行时也会有实时监控检查，监控系统运行是否符合预期；系统运行后也会将收集到的所有数据都会执行数据分析，监测系统运行是否存在潜在问题。

我们公司还会有持续的集成与发布平台来保证每一册代码改动后的系统可靠性，会提供一个稳定的 Release 版本发布给车队测试。

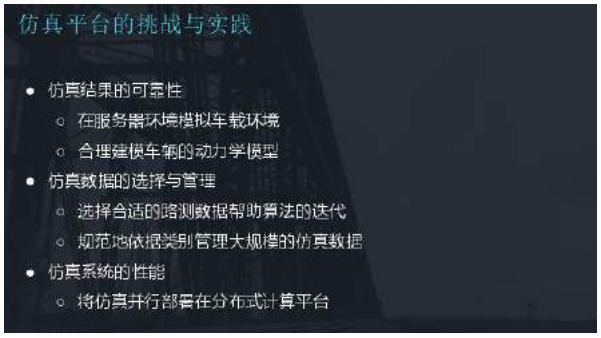
对于高性能方面，我们不可能设计一个通用的高性能系统，只能依据车载系统的实际需求做一些合理的架构设计上的取舍。我们在设计时更可能多的考虑些消息通信中需要支持的一些场景，避免大数据的拷贝时影响性能的逻辑。我们也会预计不同模块资源需求合理的分配计算资源，保证不同模块高效运行。还有是对系统定期进行 Profile 分析，通过数据矩阵的方式分析系统瓶颈，去优化性能。

对于系统灵活性方面，一开始就需要定义好通用的模块接口的通用的消息通信接口去支持不同类型的模块和不同的消息通信。

3. 仿真平台

仿真平台在日常开发起到非常重要的作用。其重要性不言而喻。首先仿真平台相对于路测低开销，不需要路测和人力的介入来评价系统性能变化；第二个就是低风险，不会对实际中的任何车辆产生影响；第三点提供了基于数据驱动的快速迭代算法的可能性，新算法直接可以在仿真平台上评估。

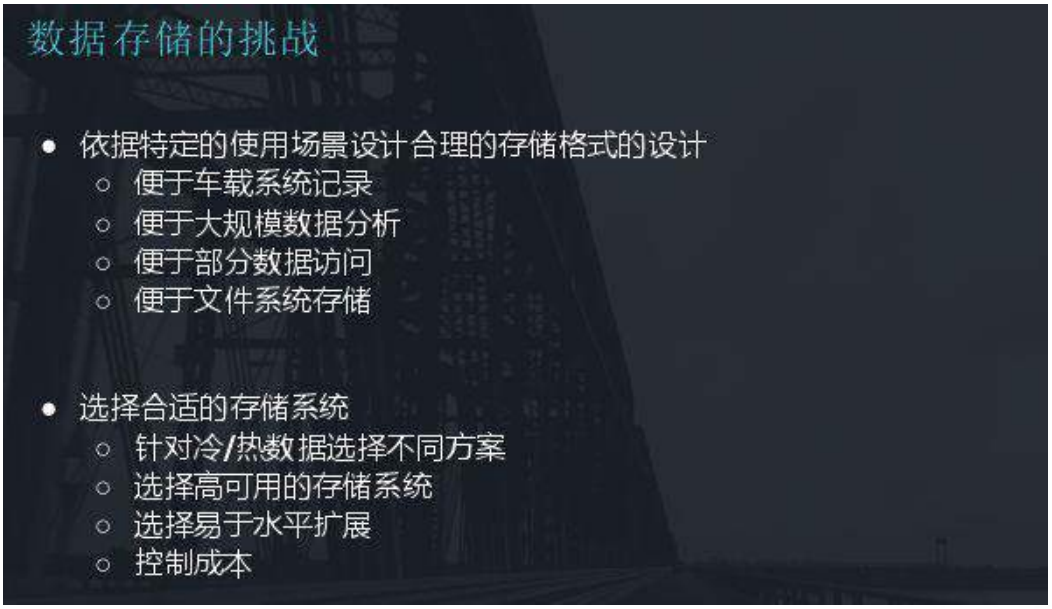
仿真平台有两个主要的使用场景。第一，它需要支持真实路测收集的场景，在路测中可能出现某一个场景处理不好，需要能够对异常进行追溯和修复，需要在仿真平台进行复现来检验算法。第二，仿真平台需要支持人工或者随机生成的场景，来帮助我们去测试一些不太成熟的算法或者是很难路测到的一些场景，如前面几辆车行驶较慢，车辆在变道的时候左边有一辆运行比较快的大车，这时需要车辆做出正确的决策，这样可以保证所有的算法和 feature 都可以在仿真平台上做足够多的测试。



下面来谈谈仿真平台的挑战与实践。首先是如何保证仿真结果的足够可靠。服务器的环境不可能和车辆的环境足够一致，机器配置和服务器配置可能存在区别，如何在仿真平台上模拟车载计算平台的环境并不容易。在仿真平台里不像真是场景有控制反馈的机制，因此需要对真实车辆的动力学模型进行仿真建模来支持仿真平台。第二个就是仿真数据的选择和管理。我们需要对路测数据进行管理，提取一些类型收集这些路测数据，仿真流水线可依据不同需求运行不同的 feature 和 case。第三点就是仿真的系统性能很重要，快速的得到仿真的结果可以帮助工程师更快的验证新的算法。因此需要一个分布式高性能的仿真平台，能够快速的将仿真结果反馈出来供工程师去决策。

4. 数据基础架构

数据是自动驾驶技术的核心驱动力，在做仿真、模型验证，解决路测不好的 case 时，都需要靠数据支持，因此我们需要解决数据的存储和访问、数据处理机制、数据同步机制等问题。对于数据基础架构而言，面临的核心挑战是数据量很大，数据量是 PB 级。还有就是数据属性不同于互联网数据，有大量客户端产生的数据（互联网是用户累计数据量大，单个用户数据量不大），有大量传感器数据，有大量模块运行日志数据。



数据存储的挑战。首先会依据特定的数据使用场景设计一些合理的存储格式，例如需要便于车载系统记录，需要进行大规模数据分析。有些数据有部分访问和随机访问的需求，一个 8 个小时运行数据，我们可能关心的只是运行过程异常的时间段，这个时候就需要系统能够方便获取我们感兴趣的数据。最重要的一点是要便于文件系统存储。当确定了存储格式还需要选择合适的存储系统，如冷热数据不同选择方案，存储系统高可用，容易水平扩展，控制成本。

数据处理可以帮助收集性能指标，有 MPI、模块运行效率、乘客舒适度体验等，还有就是路测有趣场景的挖掘，如接管、急刹、感知算法识别、不合理的变道策略等用于模型训练和仿真。在数据处理我们要着重减少从数据采集到数据处理完成的延迟，这直接影响了路测的问题得到解决的效率；另外就是依据不同类型数据处理的任务选择合适的处理系统，如有些数据处理任务是 CPU 密集型，有些是 IO 密集型，要区分不同需求用不同的机制去完成；与车载系统一样，数据处理系统同样需要有通用的任务定义以支持灵活的添加新任务。

特别说一说数据同步，Pony.ai 有三个办公室和多个测车地点，但这并不意味着需要多个系统，整个公司在用同一整套的解决方案去部署在多个不同的测车地点，这样我们的系统是有足够的泛化能力的。那么数据需要在测车地点和研发办公室之间按需同步服务，这个服务又受到带宽等网络条件的限制，设计起来还是非常有挑战的。

5. 其他基础架构服务

公司需要建立一个自动驾驶车队，车队运营会有很多业务需求，车辆增长快，运营人员增长也快。如何高效运营这个车队对配套服务的要求很高，需要进行车辆状态管理、路测任务管理，还有车队运营实时监控，帮助车辆运营状态实时观测车辆状态。

对于车队运营基础平台的挑战在 Web service 的产品开发和产品的迭代速度很快、业务逻辑复杂。相关的基础架构需要去面对快速变化的复杂需求以

及大量 Web service 的部署与管理。为了支持各种各样的计算和服务需要大规模的服务调度平台充分利用服务器集群支持多种应用，因此会在 Kubernetes 平台上做一些开发。

随着规模扩大可能会有一些可视化需求，需要维护一个通用的可视化工具，便于工程师方便查询算法运行和场景运行，因此需要高性能，需要注重 3D 实时渲染性能，系统是跨平台，支持桌面/移动/Web 等多平台，还有人机交互接口，方便乘客使用的用户界面。

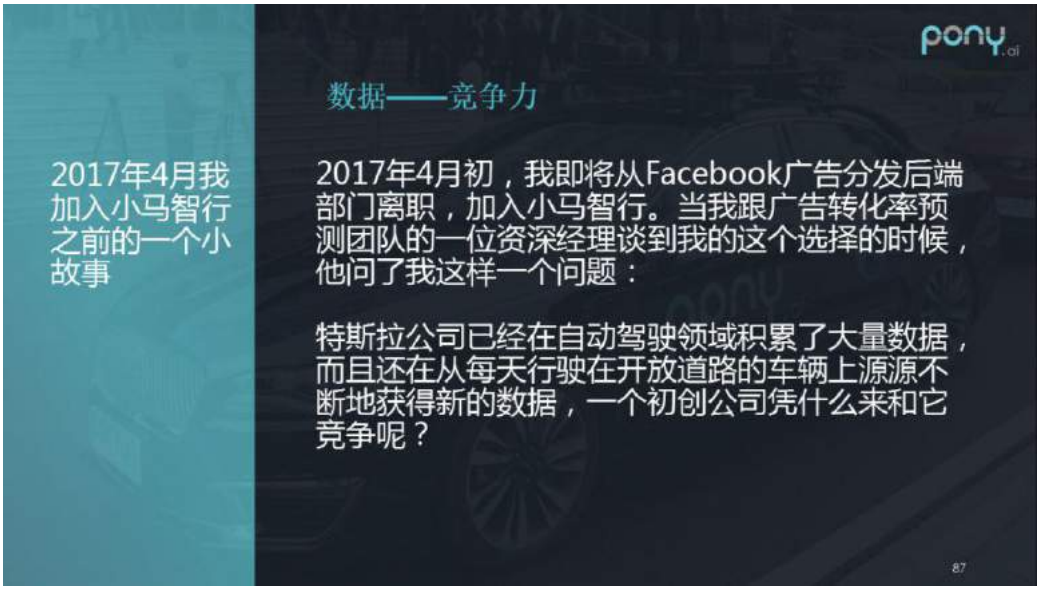
在自动驾驶研发中充分发挥数据的潜能

嘉宾：宋浩　编辑：HohXil

本次分享内容提纲

- 数据标注
- 数据驱动开发
- 数据驱动决策

前言



上图这是我加入小马智行之前的一个小故事。这不断的提醒我，人工智能需要有足够的数据量，并且充分发挥这些数据的潜能，是我们作为人工智能公司的一个非常重要的核心竞争力。

数据的作用

1. 数据驱动开发：提到数据的作用我们首先会想到，数据驱动开发，包括感知领域、行为预测领域、决策领域，需要有数据（标注好的数据）来作为我们模型训练的粮食和作为系统准确度评测的依据。

2. 数据驱动决策：尤其是优先级的决策，作为创业公司我们现在想做的事情，远远多于我们的人力，如何把人力用在我们需要优先解决的问题上，不管在行车安全性、乘坐舒适性、车辆的运营等等，每个方面都有更重要的问题，或者次要一点的问题，需要做优先级的决策，通过对于数据的分析和处理，拿出一些有效的决策。

3. 展现公司实力：通过对数据的分析，展示出来的结果是可以展示公司实力的。

4. 满足监管要求：数据的保存、处理、分析也是为了满足监管的要求。

数据标注

在讲数据驱动开发和决策前，先分享下关于数据标注的一些体会。

1. 评价指标

对于数据标注来说，它的评价指标有：

① 团队规模：只有你有足够大的团队，你才能在单位时间内标注出你所需要的数量的数据。

② 效率和成本：二者间的关系像是硬币的正反两面，效率是指单个人单位时间的产出量，成本是指单位产出的人力、设备、场地开销。

③ 质量：标注的准确度，比如标注一个激光点云里障碍物的尺寸、位置、朝向的准确度，又比如预测下一步行为的准确度。

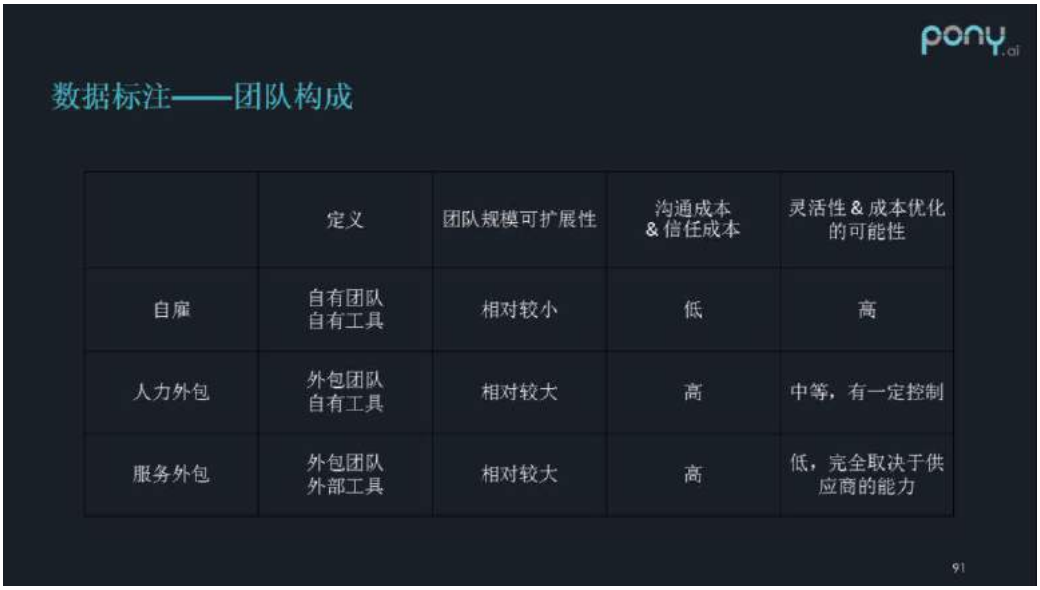
这是大家通常所关注的三个方面，下面再讲下另外两个很重要的点：

① 能力多样性：能够处理各种不同种类的标注任务的能力，对地图来说我们需要标注车道线，对于感知来说我们需要标交通灯、障碍物，还有行为预测所需要的标注等。

② 标准灵活性：各种长尾场景（如雨点，汽车尾气等）的处理方式在不断的探索和迭代，随之标注的方式也在不断的探索和迭代，如何保证在这些快速探索和迭代过程中标注团队直接的高效沟通，不至于出现混乱，这需要花很多精力去做。

2. 团队构成

如下图所示：

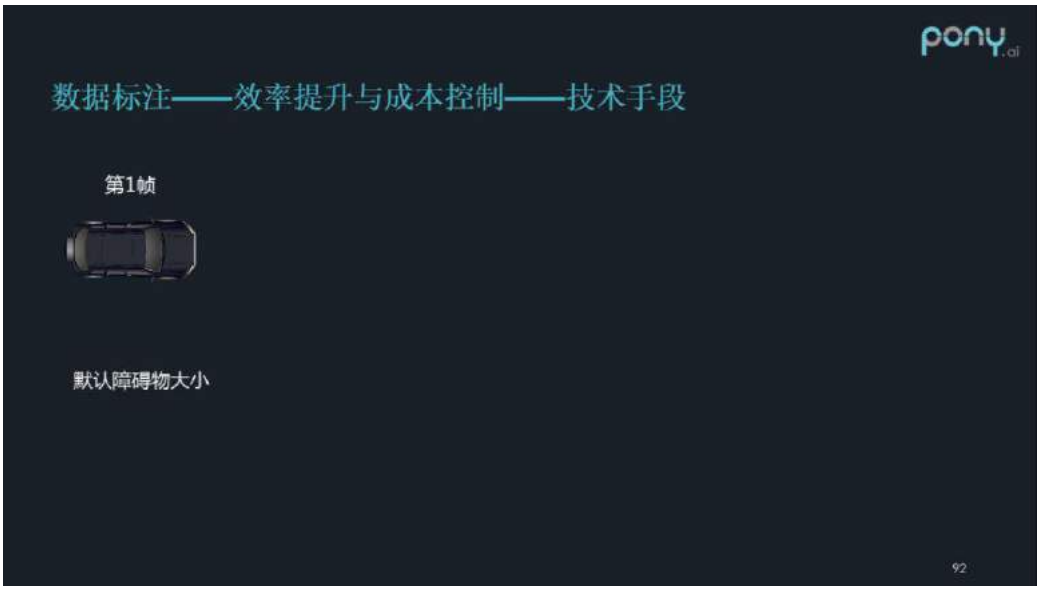


	定义	团队规模可扩展性	沟通成本 & 信任成本	灵活性 & 成本优化的可能性
自雇	自有团队 自有工具	相对较小	低	高
人力外包	外包团队 自有工具	相对较大	高	中等，有一定控制
服务外包	外包团队 外部工具	相对较大	高	低，完全取决于供应商的能力

3. 效率提升与成本控制

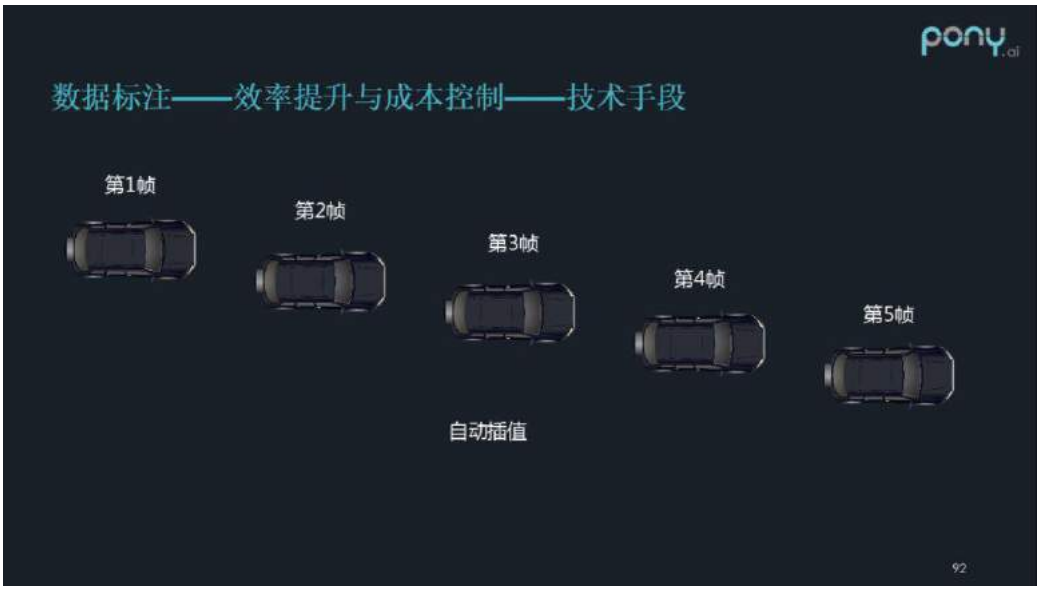
技术手段：

① 默认障碍物大小：第一帧可以通过一些人机交互的方式默认障碍物大小。



② 自动追踪外推：接下来，比如说标注员跳了一帧到第 3 帧，然后把车新的位置手工找到了，当再跳到第 5 帧的时候，系统就可以通过智能算法做到自动的追踪外推来找到车在第 5 帧中的新位置。

③ 自动插值：当车在第 1 帧、第 2 帧、第 5 帧的位置都标出来之后，系统可以做自动插值，自动找到车在第 2 帧，第 4 帧中的位置。



这里大概看下车的整个标注过程，以及人机交互方面的技巧，通过人工智能的方法提高标注员的效率。

相比于车载系统，智能标注系统所受的限制更少：

① 更多的可用信息：比如在尝试智能标注某一帧数据的时候，可以参考其后的数据帧里的信息；

② 更宽松的计算资源和时间限制。

需要注意的问题：

- 对自动化结果的依赖，可能导致标注结果中产生系统性偏差
- 如何发现和识别这些系统性偏差
- 不同的自动化功能，产生系统性偏差的几率和程度各不相同

非技术手段：

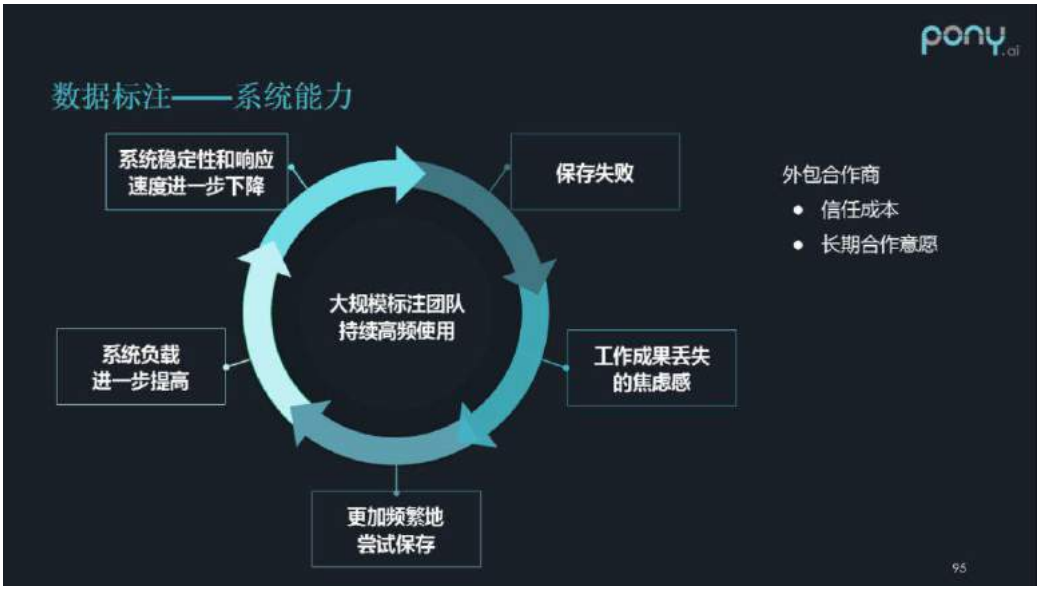
① 薪酬激励

② 组织结构设计：这里最主要的是信息流动的结构，比如：具体的某些场景的某个细节如何通过标注平台的某些技巧去标注，如何促进这些技巧在标注团队内部被高效地总结、传播。

③ 各工段之间成本平衡：系统化的思考，我们标准的流程分为标注、质检、复检等多个工段，通过各工段的配合达到整体的优化，而不是单纯的只优化某个工段。

当然所有的非技术手段依赖于标注平台对于标注任务生命周期与标注团队架构、绩效的管理。

4. 系统能力



这里的一个核心目标是**何如保证系统在大规模标注团队持续高频使用的条件下能保持稳定运行。**这里举一个因为平台不稳定性产生的焦虑感，进而导致恶性循环的例子：

- 保存失败：比如标注了 5 分钟，尝试保存的时候失败了
- 工作成果丢失的焦虑感
- 更加频繁地尝试保存
- 系统负载进一步提高
- 系统稳定性和响应速度进一步下降

而且有些时候牵扯到外部合作商时，会进步一加剧一种情况：信任成本升高，降低长期合作意愿。

这对我们提出的要求是：

① 不断优化、提升效率

② 保持稳定、保障效率：最细微的稳定性问题都可能导致效率下降

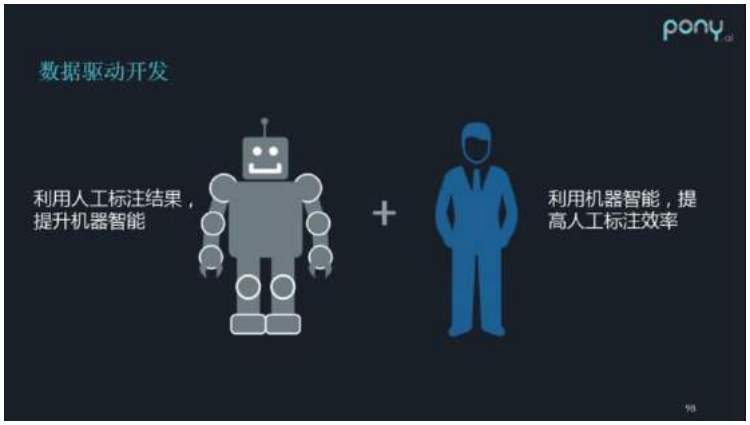
③ 良好的工程实践：

- 与线上系统隔离的完整测试环境 ,要求能较为准确地重现线上数据规模和数据分布
- 分级发布流程
- 线上系统监控及应急处理预案

关于提升效率的 Tips：

- 开发：实现效率优化方案
- 测量：在实现方案后，对标注员的操作流程和节奏进行记录和准确复现
- 提升：在测量和观察中，发现可能的效率提升点，然后再循环到第一点。

数据驱动开发



上图是我们的一个愿景：先是利用机器智能，提高人工标注效率，然后利用人工标注结果，提升机器智能，最后再反过来利用机器智能，提高人工标注效率，达到一个交互促进的过程。

1. 充分利用海量标注数据

① 分布式训练和评测系统

② 人工标注的质量是有极限的，这需要我们：

- 对标注数据的进一步处理与修正
- 在设计评测指标时，要考虑到标注数据常见的质量问题。避免设计出的评测指标对于这些常见质量问题过于敏感。

2. 数据索引平台

数据在各个维度上的分布，例如：

- 时段和天气
- 道路等级
- 障碍物种类
- 住宅、商业区、工业区

当我有了索引平台，可以做的事情有：

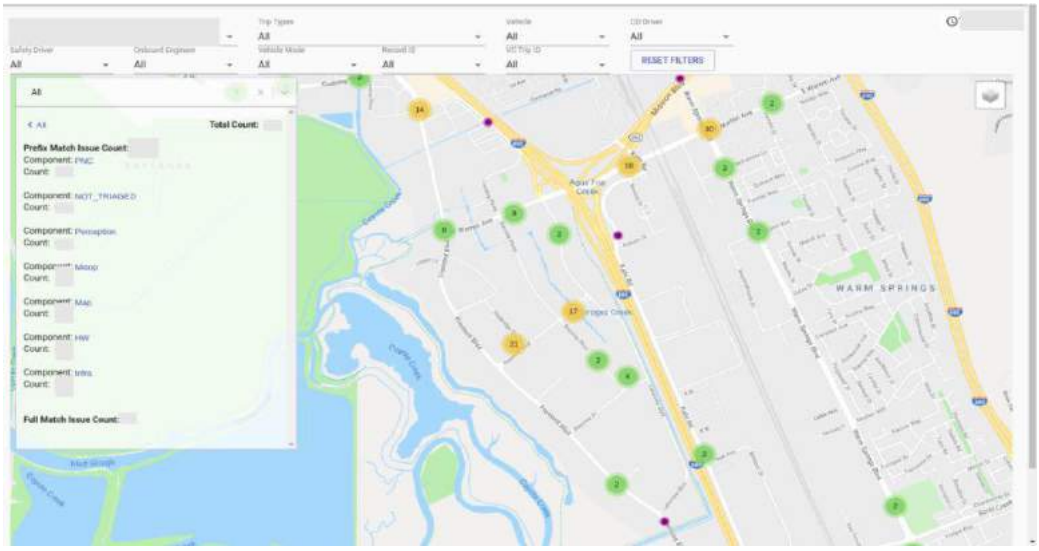
① 标注任务筛选：基于分布上不平衡的维度，对路测原始数据的自动化初筛

② 训练数据选取：按照指定的维度检索访问标注数据

③ 评测数据集维护：难度和规模分级

数据驱动决策

1. 路测事件分析



- 问题路段
- 问题模块
- 问题车辆

- 问题时间段
- 深入分析的工具

2. 数据展现方式

这里我们主要面临的挑战：

- 准确性：给的数据要靠谱，有说服力
- 实时性：每次采集的数据都可以实时更新
- 易用性：从界面上可以直观的看到关键数据

这需要我们：

① 以用户为中心，依据关键决策流程，不断迭代与优化

② 根据不同受众和使用场景，提供差异化的视图

- 运营团队周会
- 公司高管 C-level
- 团队 tech lead

③ 在每个视图中，提供最简洁实用的图表

- 在默认视图中，提供刚刚好用户想要的信息，不多不少

- 对于每一个数据点，提供进一步深入分析的工具

传统方法和深度学习结合的感知策略探索

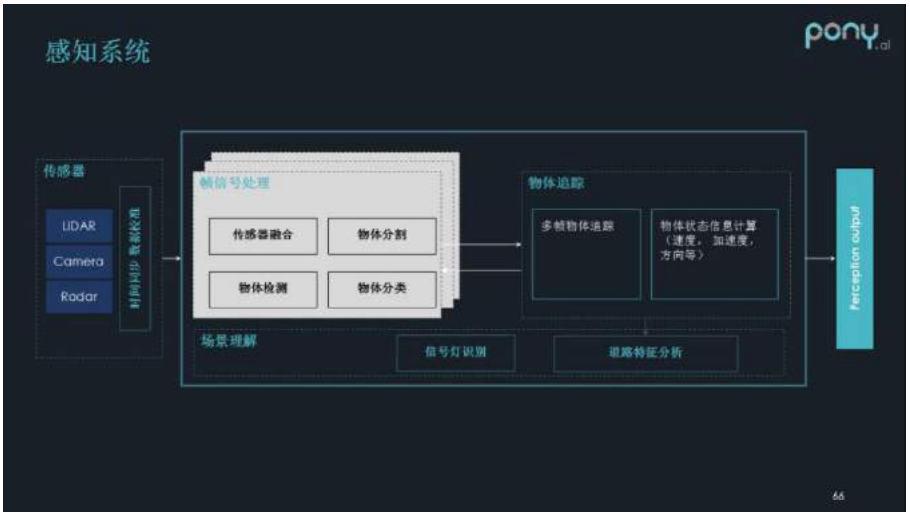
嘉宾：黄凯宁　编辑：HohXil

今天分享下 Pony.ai 在感知探索的过程中，使用的传统方法和深度学习方法。传统方法不代表多传统，深度学习也不代表多深度。它们都有各自的优点，也都能解决各自的问题。我们希望发挥它们的优点，并且结合起来。

本次分享的大纲：

- 感知 in Pony
- 2D 物体检测
- 3D 物体检测

一、感知 in Pony



首先介绍下 Pony 感知系统。感知可以认为是对周围世界建模的过程，比如车辆在行驶过程中，需要知道物体的地理位置、速度、运动方向、加速度等各种各样的信息，接收这些信息之后，再通过后续的规划和控制模块，来对车的运动做真正的调节。

感知可以类比人类眼睛的功能，就是观察的过程：

-> 传感器：激光雷达、照相机、毫米波雷达等；

-> 帧信号处理：传感器融合、物体分割、物体检测、物体分类；

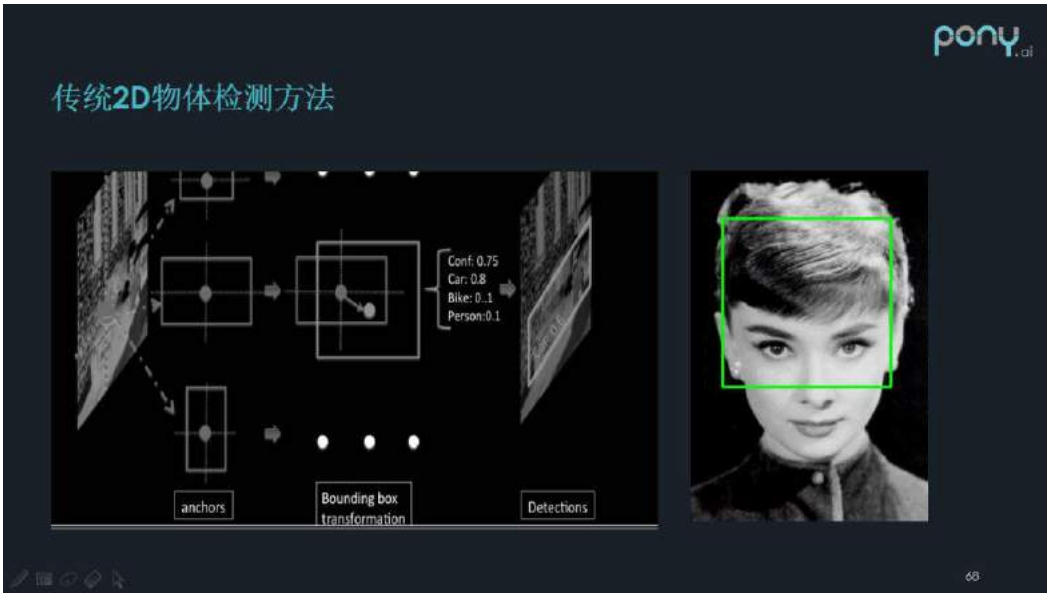
-> 物体追踪：当有多帧信息之后，可以推算速度、加速度、方向等更有意义的信息，甚至可以用多帧的信息调整分割的结果；

-> 道路特征分析：除了上述步骤，还需要对道路特征进行理解，比如交通信号灯、交通指示牌等。

感知可以认为是自动驾驶系统的基础部分，假如感知不到这个世界，就谈不上对这个世界作出反应，更谈不上后续的规划过程。

二、2D 物体检测

由于时间关系，本次分享主要介绍物体检测部分。因为必须有准确的检测和分割结果之后，才能对物体做准确的分类、追踪等等。首先介绍下 2D 物体检测：



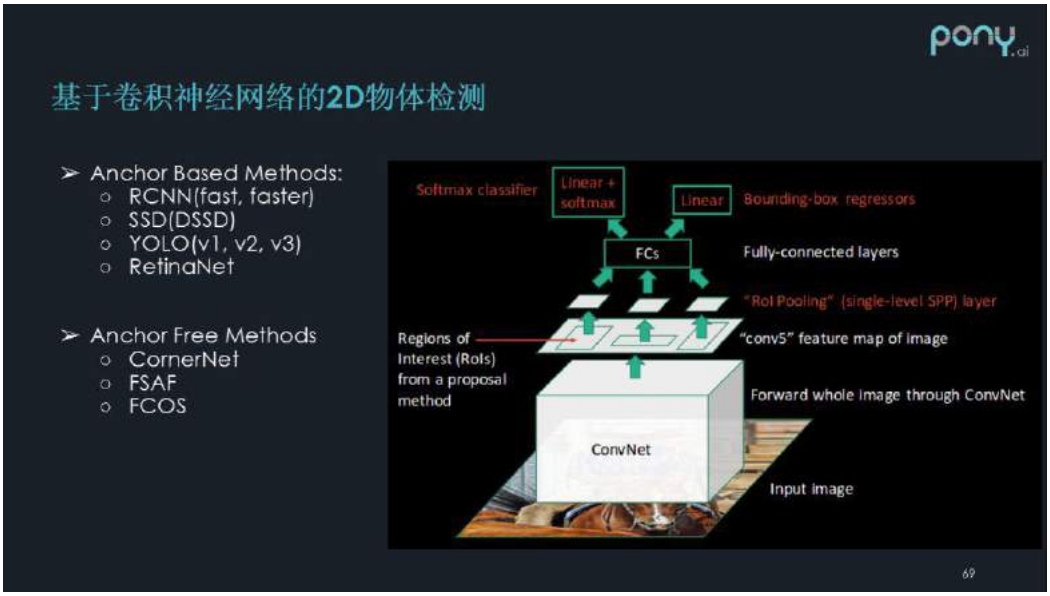
2D 物体检测是指以 2D 信息作为 input 的物体检测过程，典型的 2D 信息是照相机。传统的 2D 信息检测方法，是使用检测框遍历图片，

把对应的图片位置抠出来之后，做特征提取，用 Harris 算子检测角点信息，Canny 算子检测边缘信息等。把特征聚集在一起之后，做分类器，比如 SVM ，来判断提取的图中有没有物体，物体的类别是什么。

传统 2D 物体检测方法的缺点：

1. 检测物体时，需要预置检测框，对不同物体需要设置不同的检测框；

2. 我们需要的是高级的组合特征，而传统方法提取的特征维度比较低，对后续的分类会造成比较大的影响。



现在有了卷积神经网络，可以把传统方法上的一些问题解决掉。

卷积神经网络首先是多层感知机加卷积操作的结合，它的特征提取能力非常不错，卷积神经网络经常会有几十上百的卷积，这样就具备高维特征提取能力。其次，现在可以通过 roi pooling 和 rpn ，使整张图共享同样的特征，不用遍历整张图片，这样可以在单次操作中对图片中所有物体进行检测，这样真正具备了把物体检测模型应用到实际场景中的性能。

目前基于卷积神经网络的 2D 物体检测有俩个分支：

1. Anchor Based Methods：跟传统方法比较类似，先预置检测框，检测过程是对预设框的拟合过程。

- RCNN（fast，faster）
- SSD（DSSD）
- YOLO（v1，v2，v3）
- RetinaNET

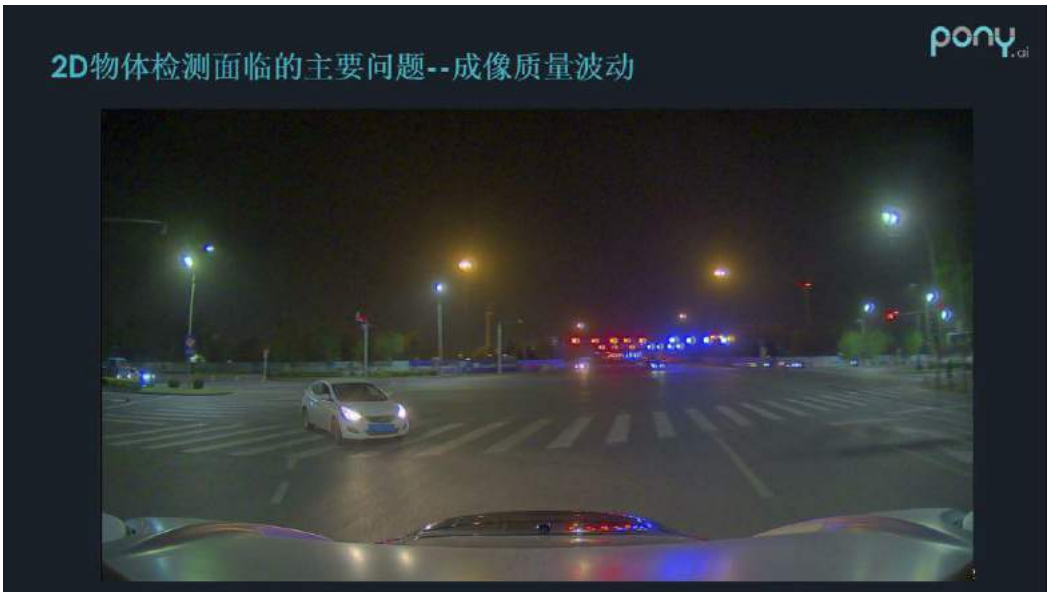
2. Anchor Free Methods：直接对特征金字塔的每个位置，直接回归对应位置上，存不存在物体，它的大小是多少，这个方法是去年年底开始大量出现的，也是未来的一个发展方向。

- CornerNet
- FSAF
- FCOS

这是 Pony 路测场景中的一个真实检测案例 ,2D 物体检测已经应用到检测一些小物体。

同时远距离物体检测也是我们在 2D 物体检测中比较关注的点，远距离物体受限于激光雷达和毫米波雷达的物理特征，缺乏良好的检测效果，而照相机在这方面比较有优势，可以和其他的检测方法进行互补。

但是用照相机做 2D 物体检测也面临一些问题。因为只有俩个维度，当俩个物体堆叠起来的时候，对一个网络来讲它的特征就比较聚集。所以，一般做物体检测的过程，会用一些非极大值抑制的方法，对检测结果进行后处理，当结果非常密集的时候，这种方法往往会受到影响。

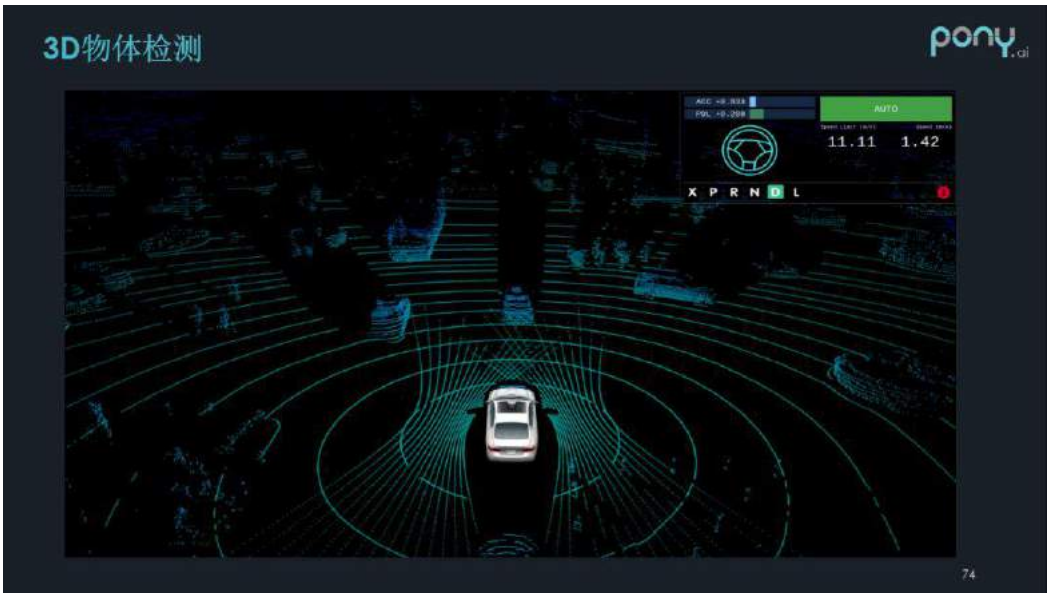


因为照相机是可见光设备，会受到光照强度的影响，并且希望特征不管是在图中的那个位置，都有足够的表达。再一个，远处的车灯和路灯也很难区分开，可能会都检测成车或者路灯。在这种情况下，特征难以区分。

再一个很大的问题就是测距问题，因为照相机是被动光源的设备，不具备主动测距的能力，这样的话需要做很多的假设，或者求解一些变态的数学问题，去估算它的距离，这个结果通常是不如主动测距设备的，比如激光雷达和毫米波雷达。

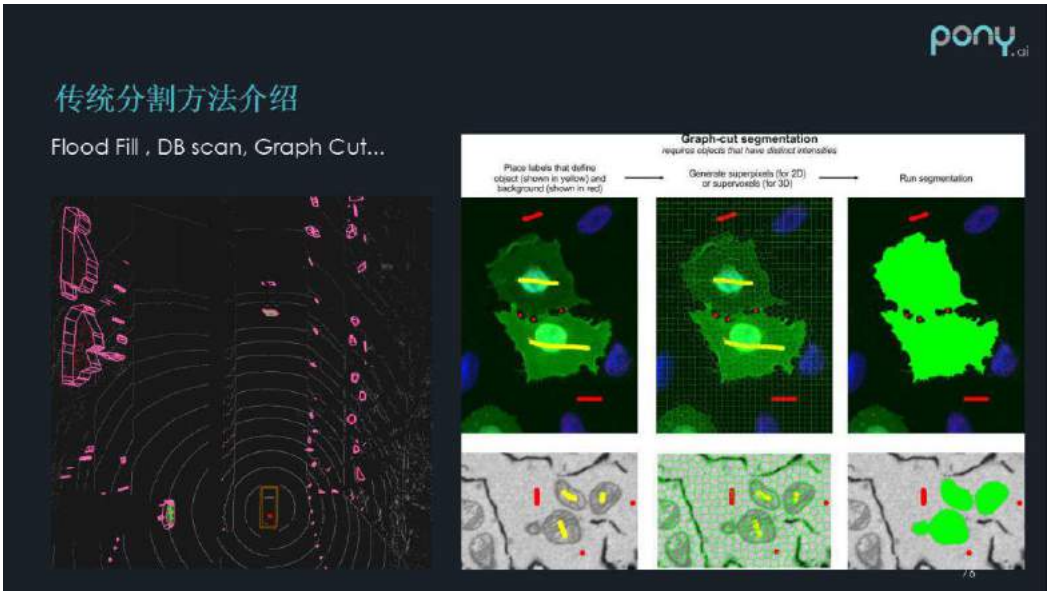
三、3D 物体检测

正是因为照相机的这些问题，所以我们也采用了其他的传感器，来做物体检测，然后把它们的结果结合起来，达到更可靠的检测效果。



3D 物体检测，顾名思义就是把 3D 的一些数据坐标，聚集起来做物体检测，比如激光雷达，类似于我们拿一个激光笔不断的扫描周围，它是一个比较明显的信息。当把数据聚集起来之后，用来推测物体的位置，大小，朝向等等。

3D 物体检测一个很大的好处就是我们在 2D 物体检测中很难区分的物体，在有了距离信息之后，就可以在聚集的维度上分开，这样在做分割的时候就有更多的信息使用，达到一个更好的工作效率。



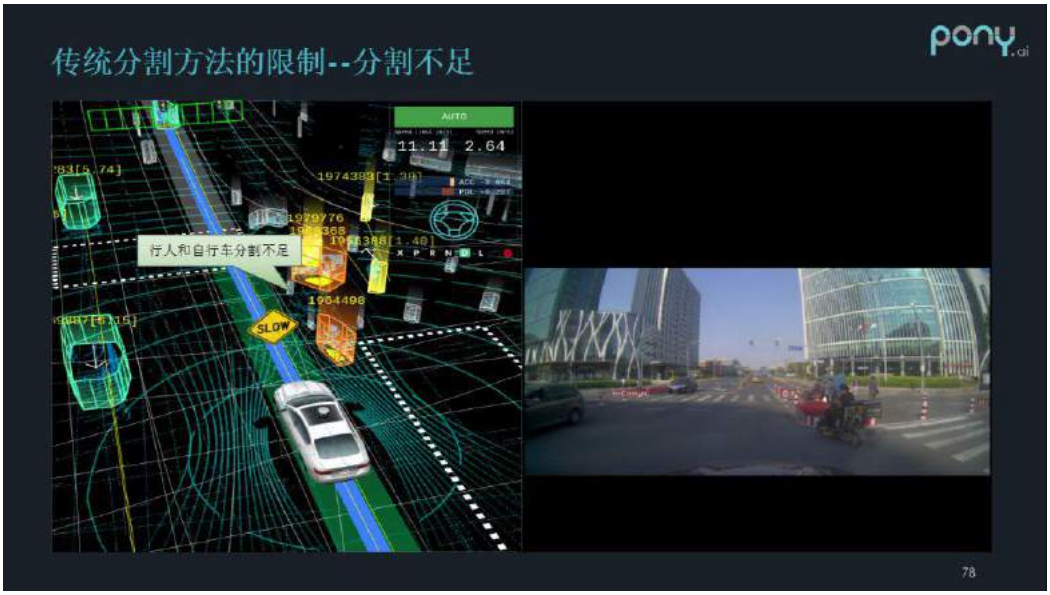
传统的 3D 分割方法：

- Flood Fill
- DB scan
- Graph Cut

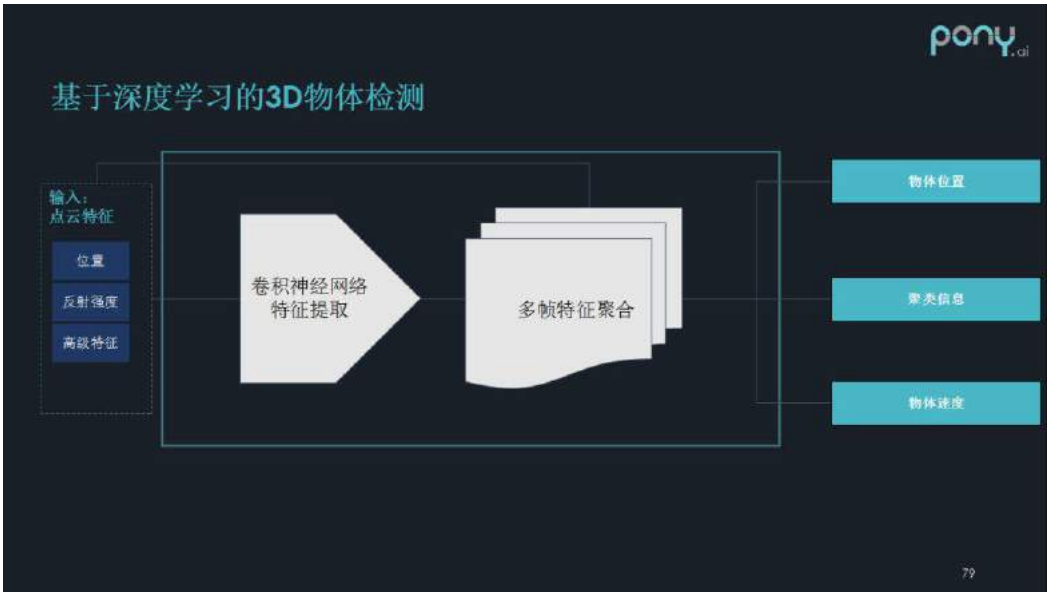
主要是利用一些点的距离信息、密度信息或者点的一些天然属性，比如它的强度，把物体聚类分割。

传统分割方法的限制：

1. 过度分割：比如图中的车，车尾和车头之间有缝隙，就很容易在 3D 检测中被分割成多个物体，因为点和点之间有间隙，在激光雷达检测时呈现的是离散信息，就会出现过度分割。



2. 分割不足：这个是 Pony 感知在早期出现的一个问题，我们称之为“三人成车”，就是当三个人离的很近的时候，很容易被系统识别成一辆车。



为了解决这些问题，就把一些深度学习跟卷积神经网络引入到 3D 物体检测中来。把点云信息进行特种工程，比如把点的位置、反射强度、高级特征聚合在一起，组织成类似图片或者图的关系，然后进行卷积神经网络特征提取，再进行多帧特征的聚合（它的意义是对运动的物体有一个更好的反映），最后得到物体的位置、聚类信息、物体速度。

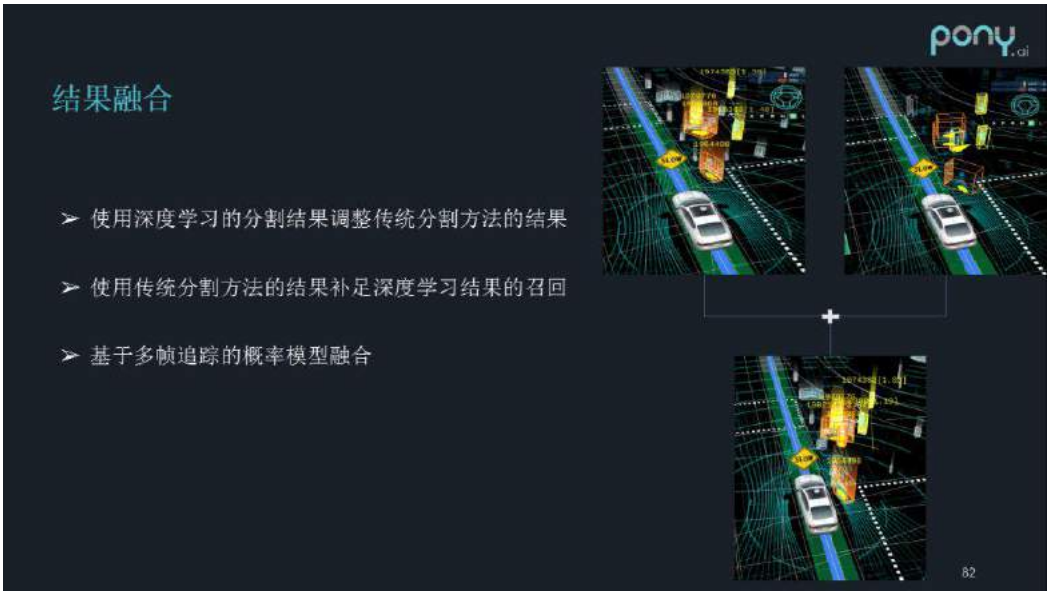
通过上述方法，“三人成车”的情况一定程度上就会避免，用户不仅可以提取人的距离关系，还可以提取到更多的高级信息，比如在点云变化中，人类似长的柱体，而自行车类似于小山一样的点云分布，这样我们可以了解他们不应该属于同一物体，而割离开。

深度学习分割方法的限制：

1. 结果的不完全可控：首先卷积神经网络经常有几百层的卷积层，参数总量可能有百万级，并且是自动学习的，导致对网络的输出没有很好的把控，就是我们无法预期 input 数据输入之后会得到怎样的数据，这在自动驾驶领域是比较致命的，因为自动驾驶是对召回率和精度有非常高要求的场景，比如车辆在行驶中，前面的一个人被 miss 掉，可能就是件严重的事件。

2. 无法保证 100%的 recall：如图所示，垃圾桶和人的特征其实非常相似，可能就把人学成了垃圾桶，导致行人在系统中出现 miss 情况。

3. 易导致过拟合：由于卷积神经网络有非常好的特征提取能力，可能在固定的数据集上导致网络过拟合，比如同样的数据集训练后，在北京的表现很好，但是当你到一个新的城市测试的时候，因为路面特征和北京有所区别，就可能导致在新的城市的效果，不如北京，这样的话对系统就非常不友好。



为了解决这些问题，我们的一个想法就把传统方法和深度学习方法的结果进行一定的结合。

1. 使用深度学习的分割结果调整传统分割方法的结果；

2. 使用传统分割方法的结果补足深度学习结果的召回；

3. 基于多帧追踪的概率模型融合：比如利用马尔可夫分布的特点、贝叶斯的方法对多帧数据进行一定的平滑，得到更好的效果。

最后，和大家分享下 Pony 在真实场景中分割的一些表现，以及可能会出现的一些问题，视频中是 Pony 非常普通的一次路测。大家可以看到在 2D 上出现了大量的物体堆叠的情况，其实 3D 上也会存在一定的遮挡，但是好在有距离信息，可以有效的分割，但是对于远距离物

体还是会存在一些问题。大家可以看到，我们的感知系统还是有很多改进空间的。

做自动驾驶真的是一个很崎岖的旅程，不断的解决问题之后又出现新的问题，不过正是因为过程的艰难，才带来更多的快乐。

感谢大家，今天的分享就到这里。

自动驾驶复杂环境下高精度定位技术

嘉宾：崔留争　整理：Hoh Xil

今天为大家分享下，自动驾驶在复杂环境下的高精度定位技术。

定位/导航负责实时提供载体的运动信息，包括载体的：位置、速度、姿态、加速度、角速度等信息。

自动驾驶对定位系统的基本要求：

1. 高精度：达到厘米级。

2. 高可用性：保持它的稳定性，自动驾驶测试已经从封闭的场景转移到更开放的场景，这要求我们定位系统能处理更多更复杂的情况。

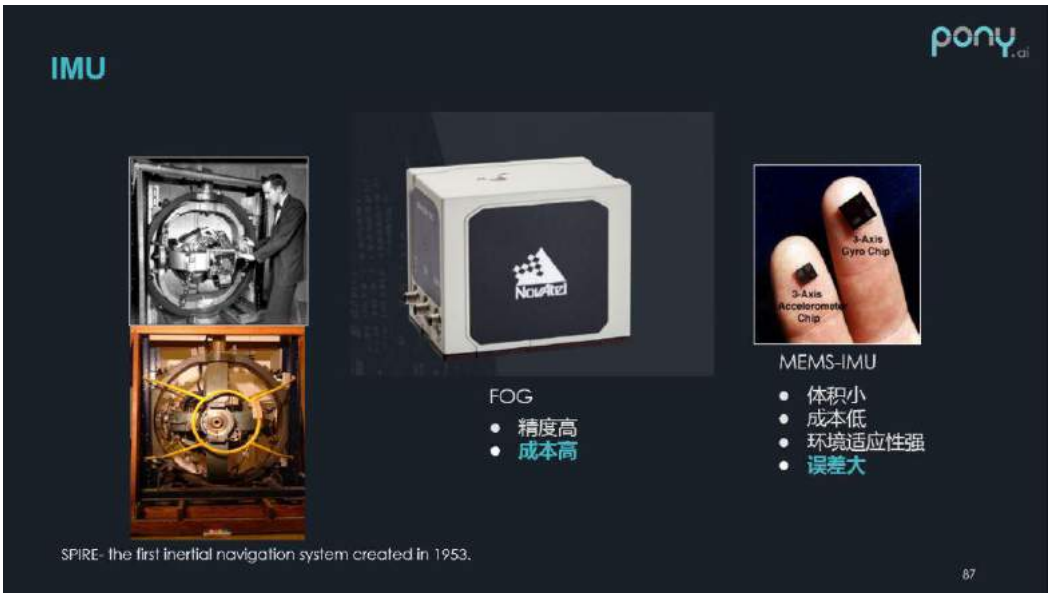
3. 高可靠性：整个定位的输出是感知，规划与控制的输入，如果定位系统出现偏差将会导致很严重的后果。

4. 自主完好性检测：因为我们系统的可靠性只能做到非常接近 100%，但是难以达到真正的 100%，这要求我们在没有办法提供准确的输出的时候，能及时的警告用户，采取措施，避免发生事故。对它的要求就是要保证较低的虚警率与漏警率。

为了满足自动驾驶对导航与定位的要求，采用的方法有：

- 惯性导航 INS
- 全球导航卫星系统 GNSS

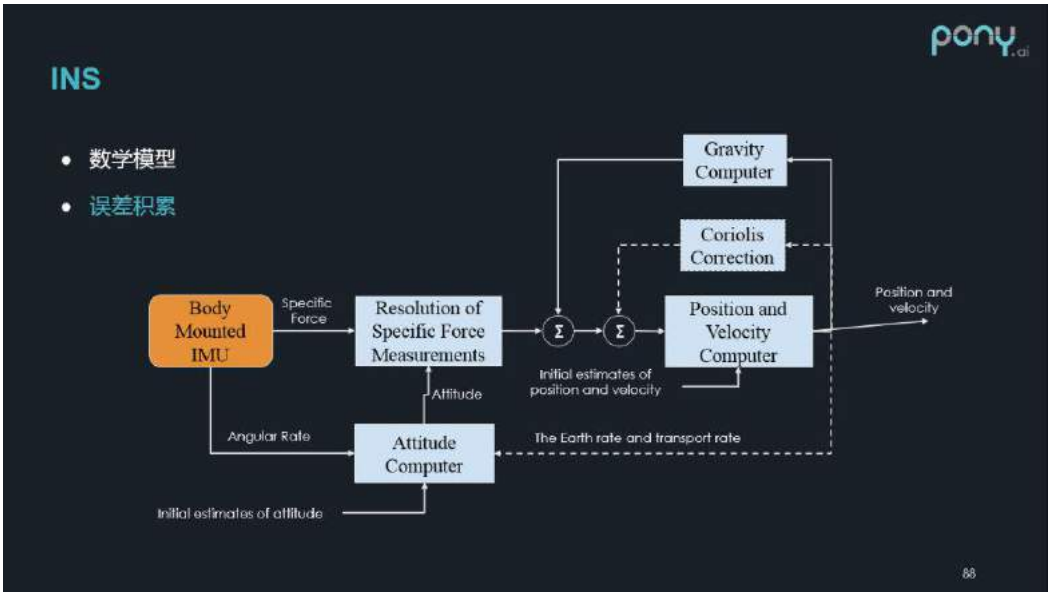
- 高精地图与线上激光 lidar 点云匹配定位
- 里程计
- 结合车辆运动特性的运动约束



接下来介绍下惯性测量单元 IMU ，目前自动驾驶常用的按照精度分为两类：

一种是基于光纤陀螺 FOG 的 IMU ，它的特点是精度高，但同时成本也高，一般应用于精度要求较高的地图采集车辆。

第二种是基于 MEMS 器件的 IMU ，它的特点是体积小，成本低，环境适应性强，缺点是误差大，如果把它应用在自动驾驶测试车辆中，需经过比较复杂的处理。



为了有 IMU 输出的原始数据得到导航定位输出，需要进行捷联惯性导航（SINS）解算，包括这样的几个模块：

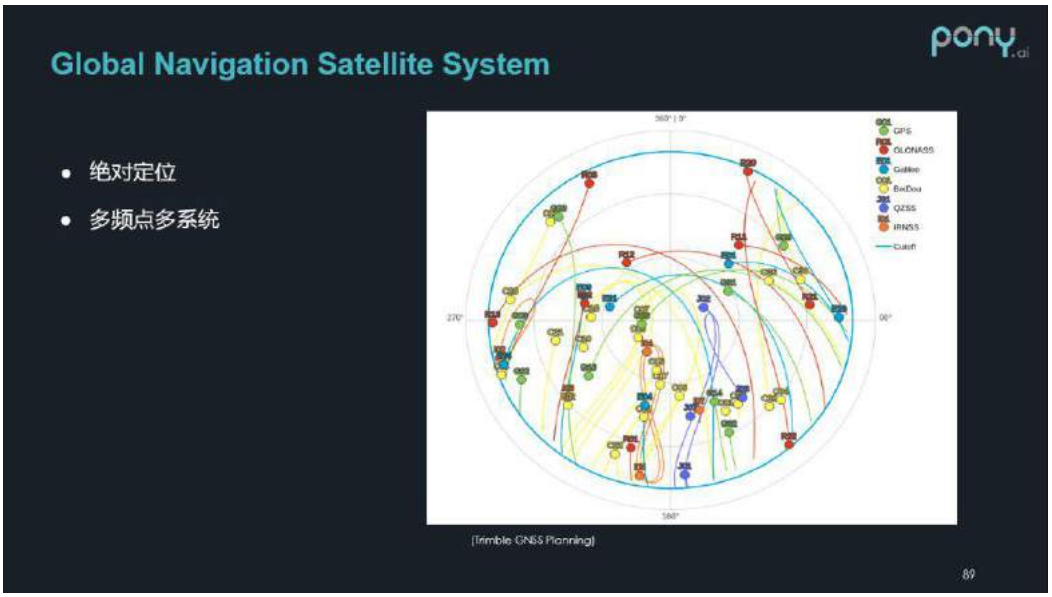
1. 陀螺输出角速度信息积分得到姿态

2. 采用姿态信息对加速度计输出的比力做转换，由载体坐标系到导航坐标系

3. 重力计算，有害加速度，地球自转角速度等补偿

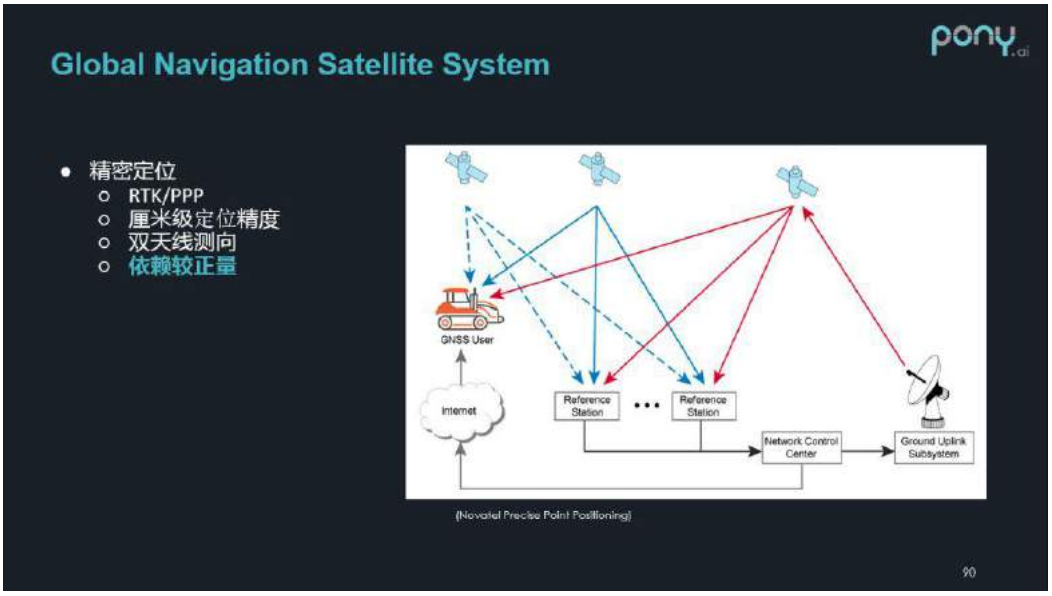
4. 加速度信息积分得到速度与位置

由于积分过程的存在，造成输出误差随着工作时间发生积累。



下面介绍下全球卫星定位导航，它是基于测定的绝对定位方案，误差不会随着工作时间以及载体运动发生变化。

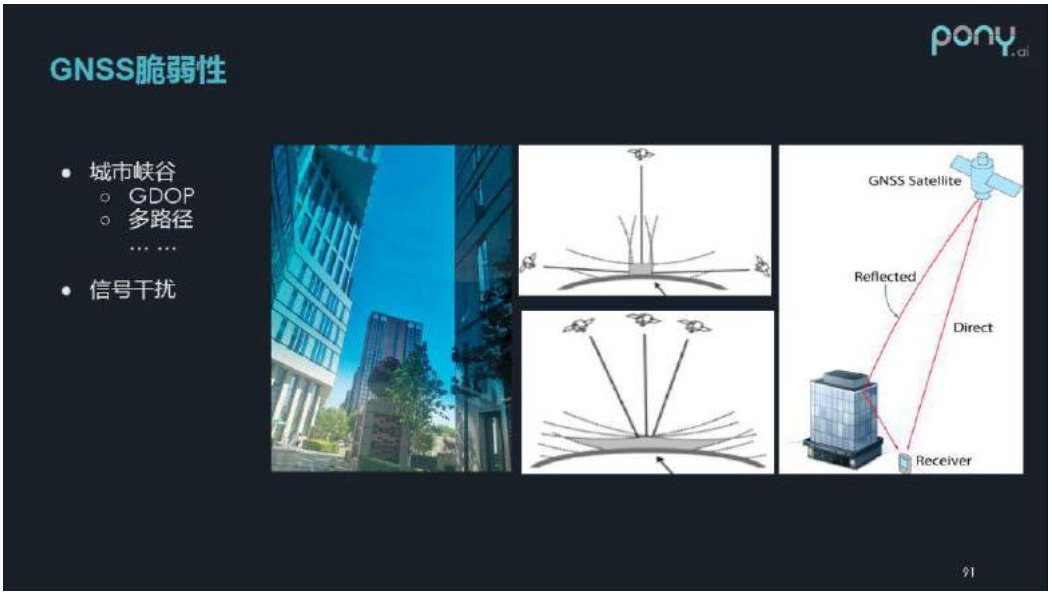
目前全球卫星导航的一个显著的特点是由单频单系统向多频多系统转变，比如我们国家的北斗导航系统，俄罗斯的 Glonass ，欧盟的 Galileo 等。这幅图是北京某一天所能接收到的卫星信号的星空图，可见基本可以保证在任意时间我们都能收到 35 颗以上的卫星。多频多系统在极大的程度上提高了我们导航系统的可靠性与可用性。



另外一个显著的特点是精密定位，在民用领域的广泛应用，尤其是基于载波相位动态差分的 RTK 技术在智能驾考，无人机，精密农业等领域得的了普遍的应用。在自动驾驶领域他的优点是：

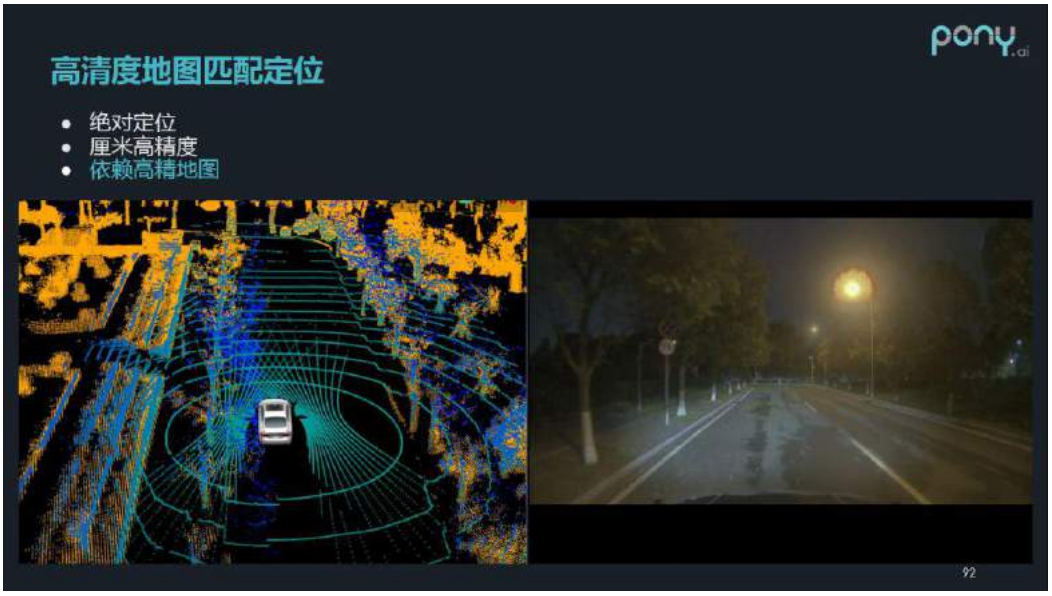
- ❖ 厘米级定位精度
- ❖ 双天线测向

精密定位在提高定位精度的同时，由于需要基站信息，这也增加了整个系统对于基站（基站服务提供商）和网络信号的依赖。

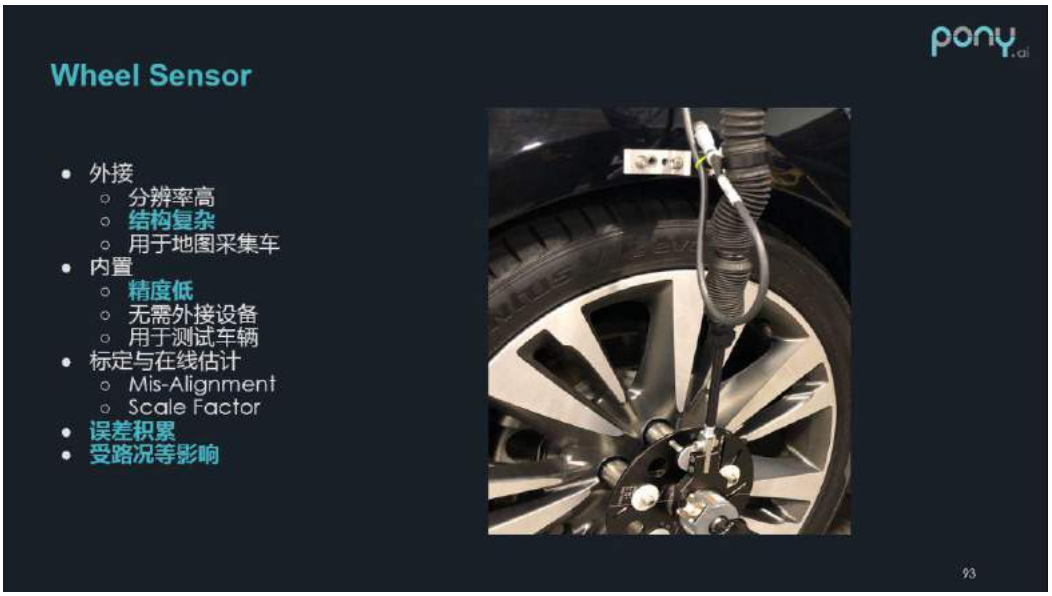


因为 GNSS 是基于卫星定位的导航，具有非常脆弱的特点：

我们在城市里经常会遇到城市峡谷，使我们没办法接收低仰角的卫星信号，极大的增加了我们定位的因子，增大了定位结果的不确定性。另外一个是信号干扰，车辆本身及加装设备等使车辆的电磁环境非常恶劣，如果处理不得当，会严重影响接收机的收星及观测值的质量。



接下来是基于高清度地图匹配定位，它是事先建立的高精度地图，配合我们线上的激光点云，可以做到绝对定位，厘米级高精度，它的一个显著的局限性是增加了我们对高精地图的依赖。

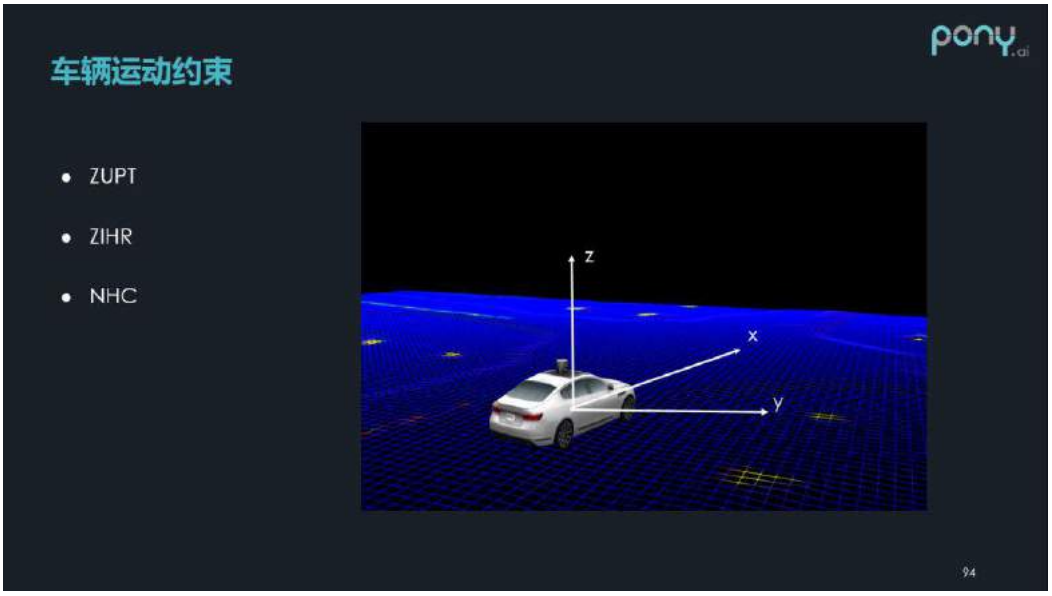


另外一个是轮式传感器 Wheel Sensor ，自动驾驶目前获得车轮的信息有两种方法：外接和内置。

外接，它的特点是分辨率和精度都非常高，缺点是结构复杂，可靠性难以保证，通常用于我们的地图采集车。

内置传感，它的特点是无需外接设备，缺点是精度低，误差比较大，如果用于测试车辆需要经过很多的处理才可以。

另外，IMU 的敏感轴，很难保证与车的运动方向是完全平行的，所以需要做一个精确的标定，除了轮速的输出，要得到有用的车速信息和里程信息，需要做一些推算，需要一些车轮的参数，而车轮的参数会随着我们的工况发生变化，所以我们需要对它做在线的估计。在这个过程中，我们的计算误差会随着运动的距离发生积累，且会受到路况的影响，比如轮胎打滑或者过减速带。



我们还可以结合车辆的运动特点做些约束。比如我们可以通过一些方法检测车辆是否处于静止，这样就可以假设在静止的时候速度等于 0，可以保证我们在极端的情况下，我们的定位结果不发生很大的误差。我们可以加入一些车辆横向或者纵向的约束条件加入到多传感器融合的系统中。

在获得传感器输出数据之后，我们整个多传感器融合定位，包括几个部分：

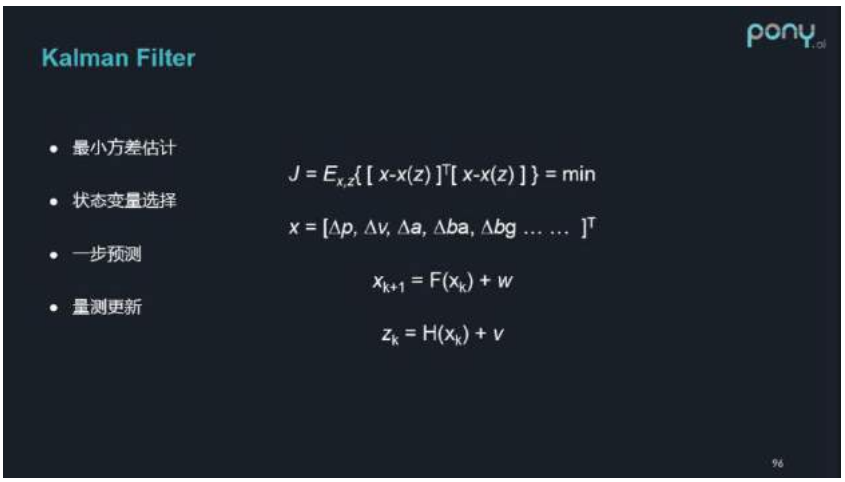
1. 数据预处理，将惯性导航解算，GNSS 的质量控制，雷达数据的误差补偿，还有基于轮式传感器的计算，以及在线估计和补偿。

2. 基于雷达数据和高精地图的匹配定位。

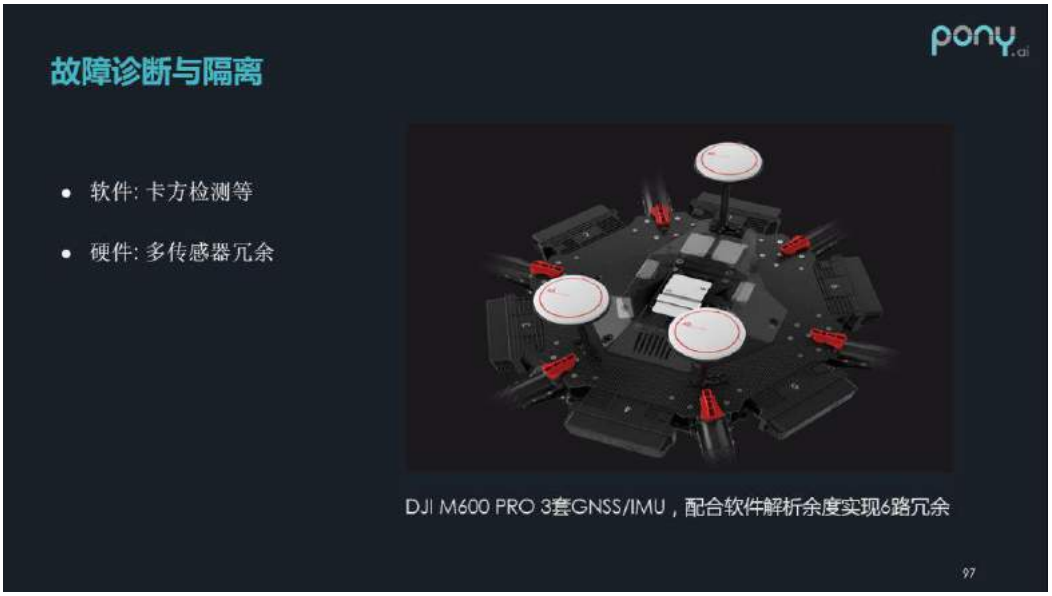
3. 四个核心模块：

- ZUPT/ZIHR/NHC，车辆运动约束部分
- INS Alignment，惯性导航初始对准部分
- Integrated，组合部分
- FDI，故障检测与隔离部分

4. 安全相关的模块，就是对于输出的完好性监测。

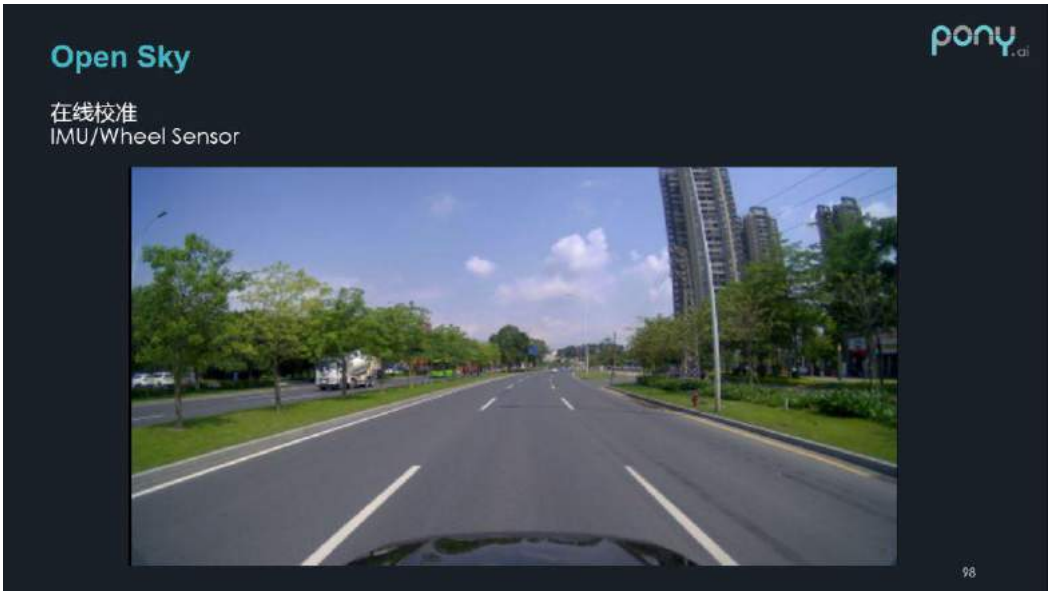


目前我们的导航定位优化方法，还是基于传统得卡尔曼滤波器，其优化的一个指标是使状态方差达到最小。在建立 Kalman Filter 模型的时候，首先要选择状态变量，目前我们还是基于导航参数误差+车载的传感器误差，作为状态估计。然后状态方程通过一步预测和量测更新，实现我们在时间域的递推。

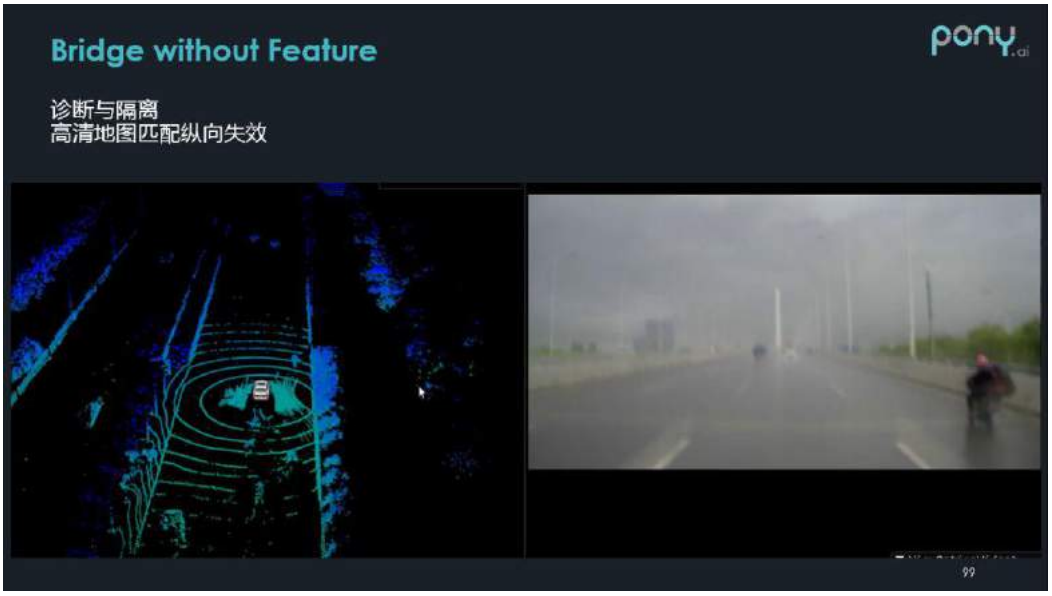


下面简单介绍下故障诊断与隔离，传统的软件方法有很多，比如卡方检测等，另外是硬件上的冗余。如果我们可以配备多套 GNSS/IMU ，我们可以配合软件解析余度实现多个冗余，来提高可靠性。

典型场景：

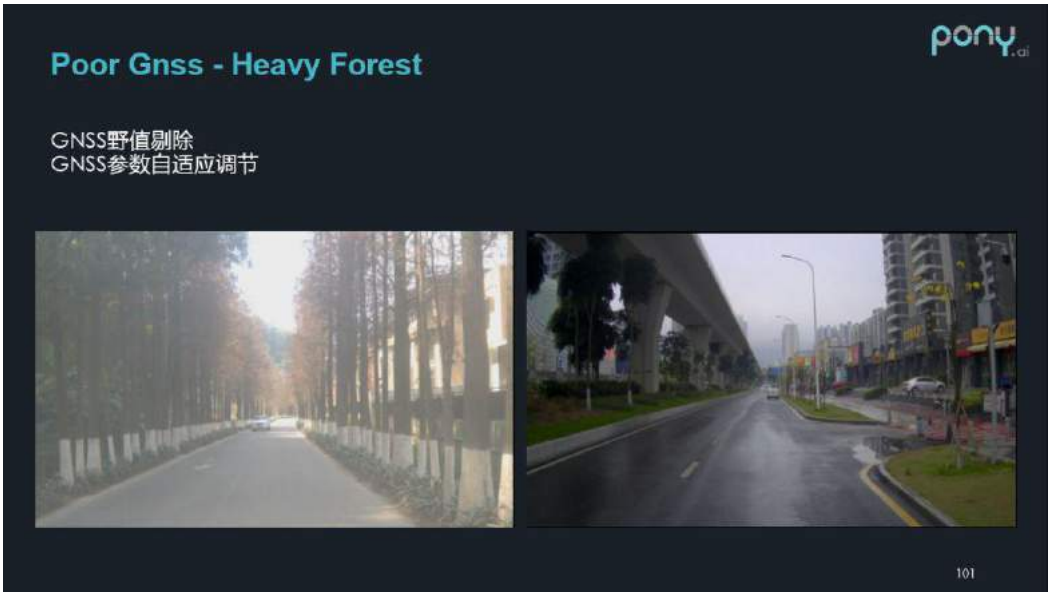


首先是一个非常典型的场景，视野非常开阔，这时我们可以利用，Gnss/Rtk ，Scan Math Pose 作为量测量校准 IMU/ Wheel sensor 误差，并作出补偿。

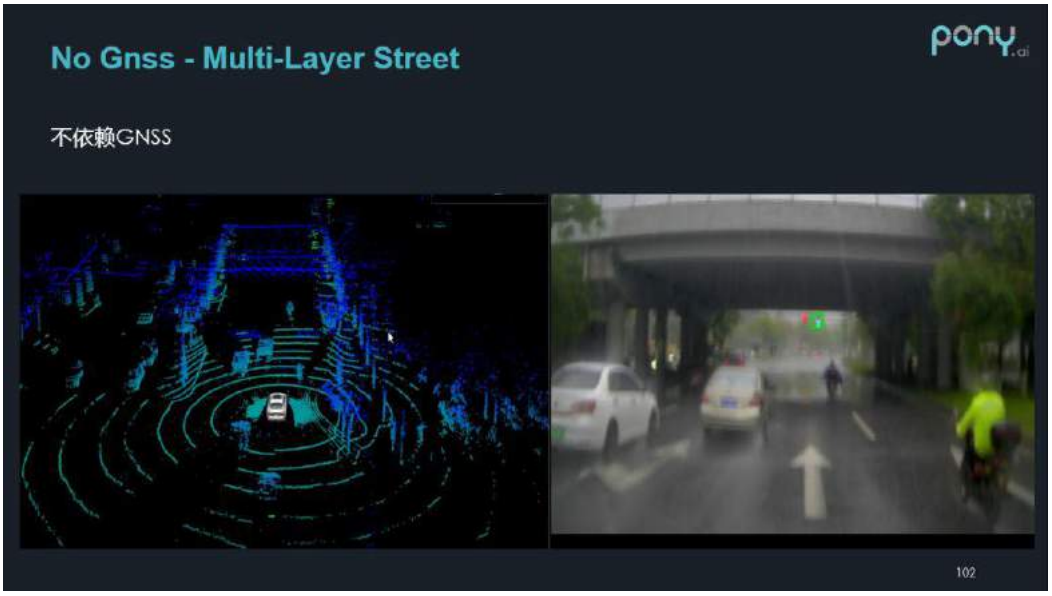


这个场景是我们的车辆运行在一个激光雷达纵向特征非常不明显的桥上，导致我们的高精定位在纵向上失效情况。解决办法是使用 GNSS，IMU，Wheel Sensor 信息，检测 scan match 纵向失效情况， 使其不影响定位系统输出。

上面这个场景是我们的激光雷达由于点云被遮挡，导致我们与高精地图匹配定位失败，这要求我们整个系统能不依赖于匹配定位，也能输出非常可靠的结果。



这俩个场景是对于卫星导航来说非常恶劣的场景，需要我们从传感器融合部分去处理，使其具有野值的剔除能力，以及对它参数的自适应调节能力。

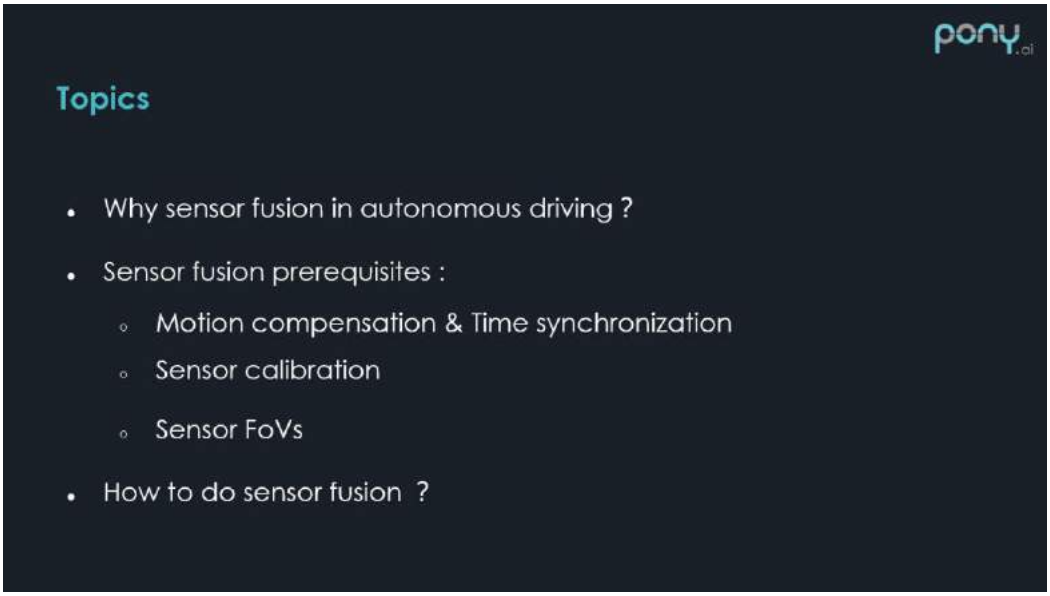


这个场景更极端，是车辆穿越高架桥的下面，导致我们整个卫星定位导航失败，此时需要我们的系统能独立于卫星导航，也能可靠的工作，得到可靠的结果。

自动驾驶多传感器感知的探索

嘉宾：刘博聪　编辑：Hoh Xil

Pony.ai 在多传感器感知上积累了很多的经验，尤其是今年年初在卡车上开始了新的尝试。我们有不同的传感器配置，以及不同的场景，对多传感器融合的一些新的挑战，有了更深刻的认识，今天把这些经验，总结一下，分享给大家，与大家一起讨论。



本次分享分为三部分：

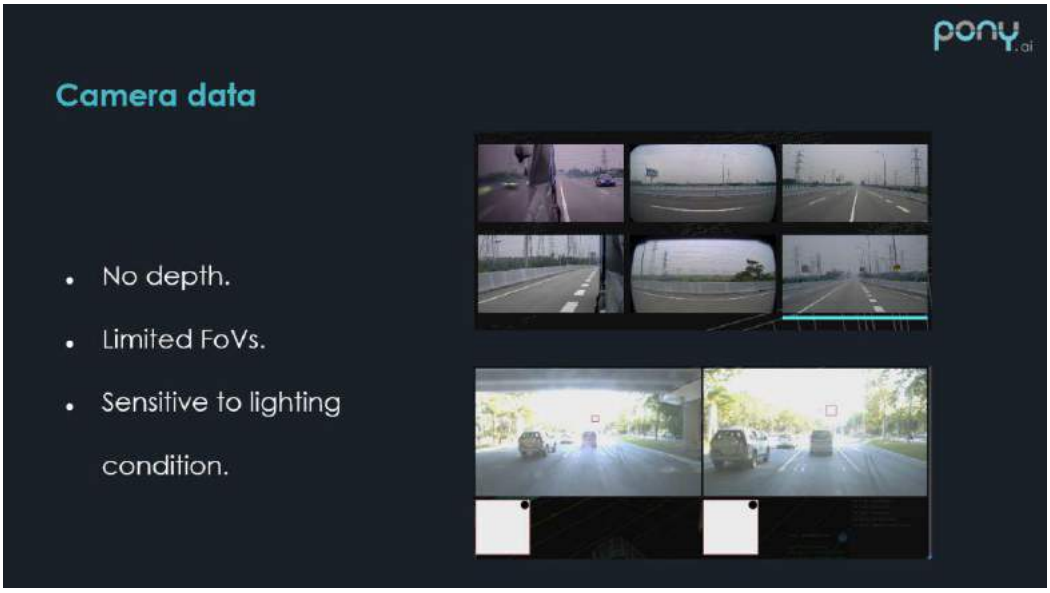
- 为什么需要多传感器融合？
- 传感器融合的一些先决条件

❖ 如何做传感器融合？

为什么需要多传感器融合？

首先，单一传感器在自动驾驶中，都有各自的挑战，所以先了解下常用的传感器的挑战是什么：

1. Camera data



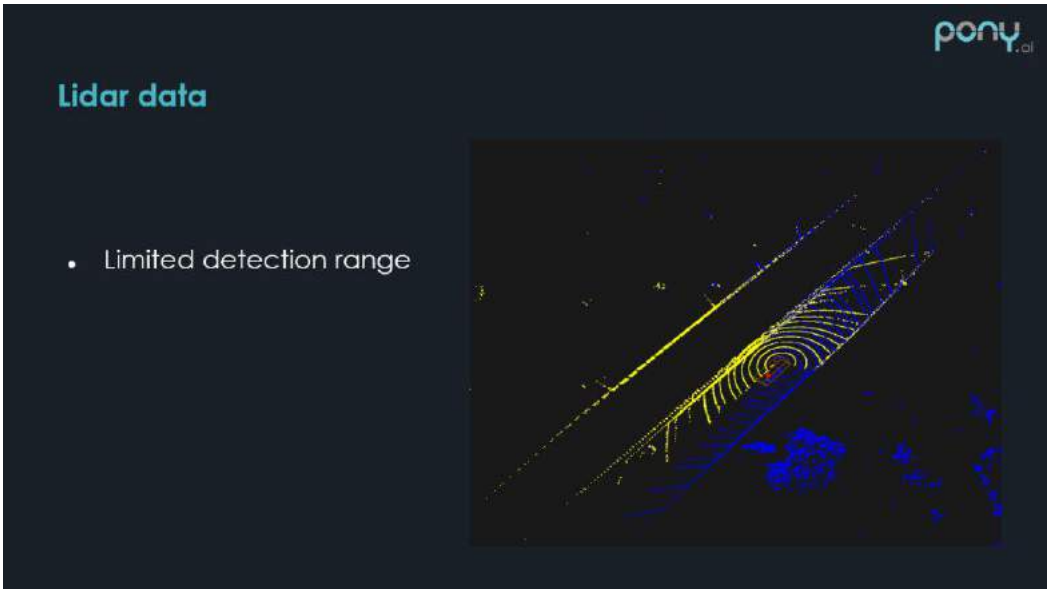
照相机数据遇到的挑战：

① 没有深度信息。

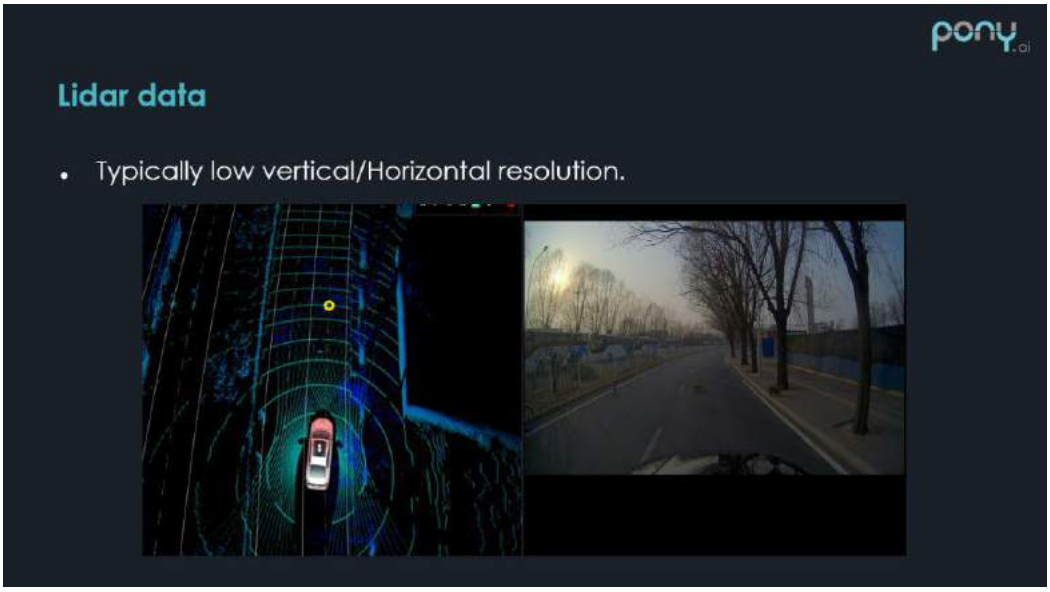
② 视场角有限，以卡车的传感器配置来说，需要比较多的摄像头，这里用到了 6 个摄像头覆盖了 270°的视场角。

③ 摄像头受外界条件的影响也比较大，（上图右下方）这是当车行驶到桥下时，由于背光，且光线变化比较大，导致无法识别正前方的交通灯。

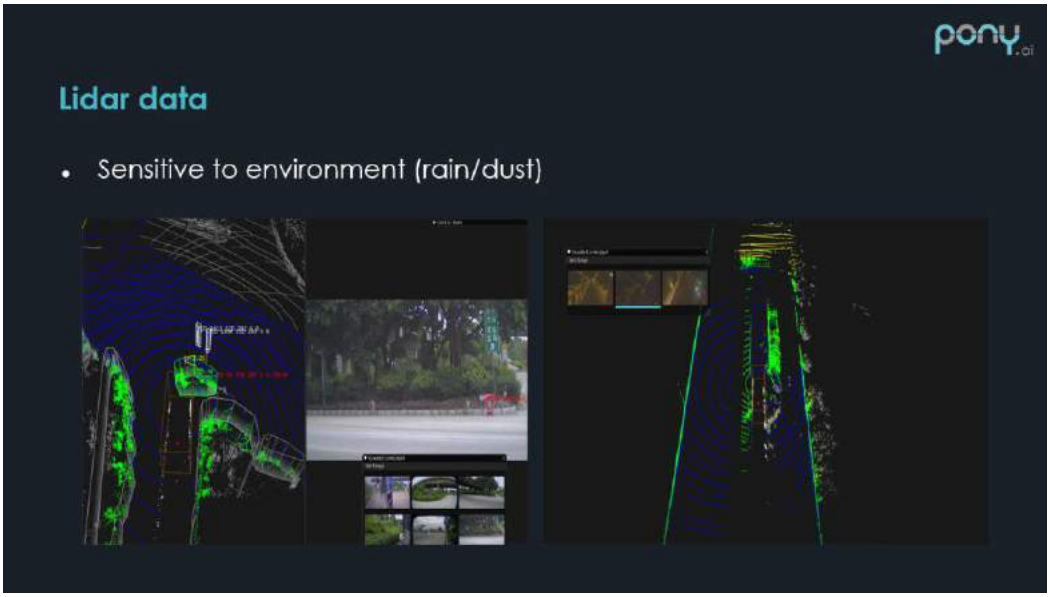
2. Lidar data



激光雷达数据的一个比较大的挑战是感知范围比较近，如右图所示，感知范围平均在 150m 左右，这取决于环境和障碍物的不同。

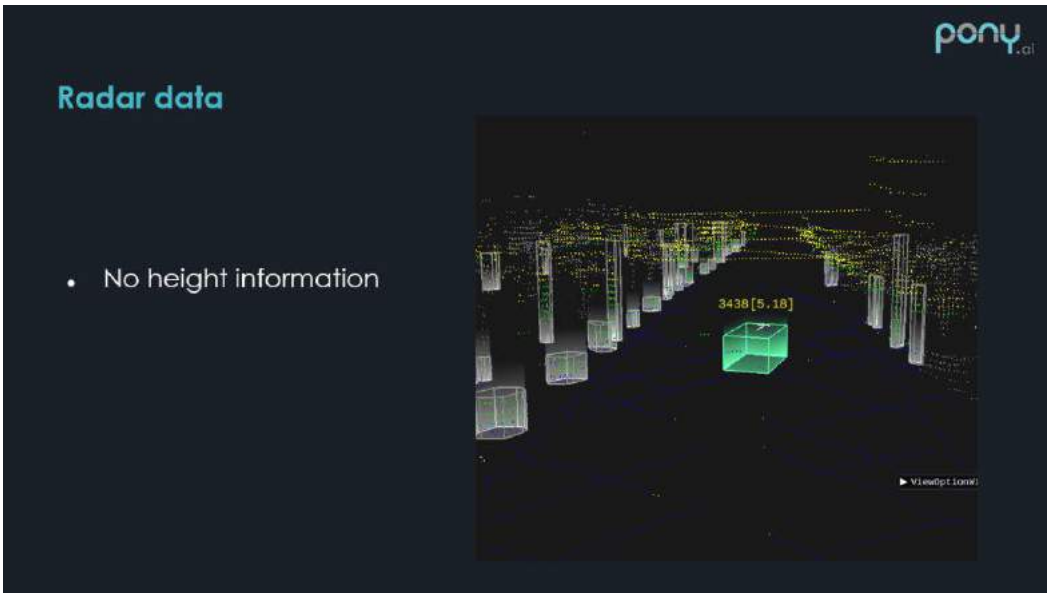


激光雷达在角分辨度上远远不如照相机，如上图，有三条小狗在过马路，在照相机上可以清楚的看到，但是在激光雷达上，采集到的点是比较少的，这样的场景在每天复杂道路路测的时候会经常遇到。



激光雷达对环境的敏感度也是比较大的，上图为 Pony 路测的时候经常遇到的虚拟噪点的 Case，在车经过建筑工地的时候，在图像上可以看到并没有任何的障碍物，但是在雷达上前面有很多的噪点，右边是雨天中的测试，车辆行驶中溅起来的水花，在激光雷达上都是有噪点的，如何去除这样的噪点，是我们经常面临的挑战。

3. Radar data



毫米波雷达，本身的一个挑战是没有高度信息，毫米波雷达能告诉你这个物体在哪个位置，但是不知道多高，这个 case 就是前面有一个比较高的指路牌，毫米波雷达知道这儿有个障碍物，但是不知道是悬空的。

4. Why sensor fusion

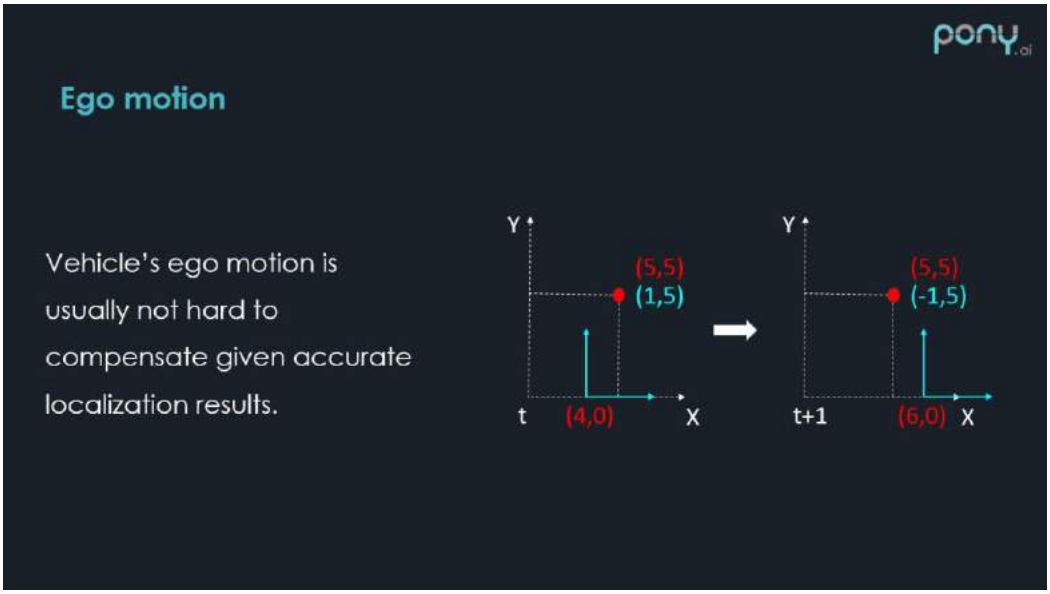
当看过了这些单一传感器在自动驾驶中面临的挑战，自然会想到，做多传感器融合，进行传感器之间的取长补短，来帮助整个感知系统效果的提升。这里的例子是 ,如何利用多传感器来提升感知的探测距离 ,当障碍物距离 150m 左右时，激光雷达的反射点已经比较少了，但是这时毫米波雷达和照相机还是比较稳定的。

当障碍物驶出 200m 的范围，基本上没有任何的激光雷达反射点了，但是 200m 取决于自动驾驶车辆本身的车速是多少，200m 的感知距离还是必要的，这时只能通过毫米波雷达和摄像头，来提升对障碍物的感知距离，从图中可以看到障碍物还是可以稳定识别出来的。

传感器融合的先决条件

1. 运动补偿 & 时间同步

① Ego motion



为什么做运动补偿？在自动驾驶传感器感知过程中，传感器采集数据，一般都不是瞬时发生的，以激光雷达为例，采集一圈数据需要 0.1s，在这 0.1s 内，本身车会发生一定的位移，障碍物也会发生一定的位移，如果我们不考虑这样的位移的话，我们检测出来的位置就会不准确。

位移有俩种，第一种就是车自身的位移 Ego motion。右边画了一个示意图，虚线部分可以认为是世界坐标系，红色的点代表一个静态的障碍物，它在坐标系中有一个稳定的坐标（5，5），蓝色部分代表车自己的坐标系是局部坐标系，（4，0）为这个坐标系的原点，在 t+1 时刻，这个坐标系移动到了（6，0）的位置，车沿着 X 方向向前移动了 2，在 t 时刻在车自身的坐标系下，障碍物的坐标是（1，5），在 t+1 是时刻，则是（-1，5）。如果不做车自身运动的补偿，静止的物体在 2 帧之间，测量的局部坐标是不一样的，就会产生错误的速度，因此，要去补偿车本身的位移，做自身的 Motion

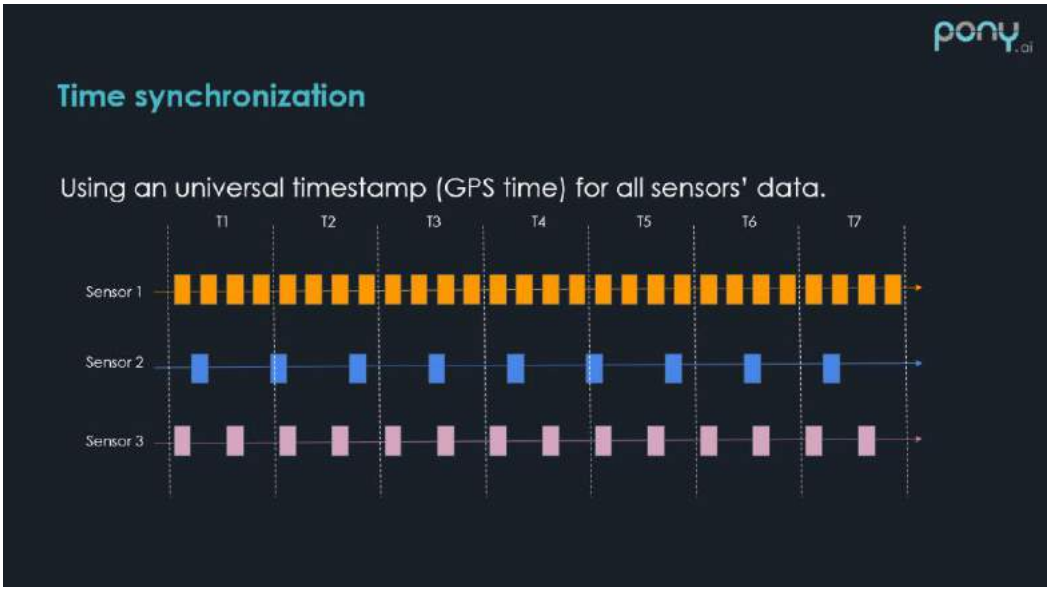
compensation 运动补偿。这个问题比较简单，因为车是有比较准确的定位信息的，它会提供这俩个时刻，车本身的姿态差距，我们可以利用姿态差，比较容易的补偿车移动了多少，那我们就可以知道这个障碍物其实是没有移动的。

② Motion from others

第二种要考虑的是运动物体在传感器采集的时间段内，运动物体发生的位移，相对于自身运动补偿，这是一个更难的 case，首先快速移动的物体，在激光点云里很可能会被扫到俩次，大家可以看下红圈内，尾部会有拖影。所以我们如何想办法消除对方车的 Motion，也是要考虑的。解决的方式有很多，现在激光雷达本身从硬件上也会有些配置，来缓解这样的现象，简单解释下，当你用多个激光雷达在自动驾驶车辆时，可以让激光雷达按照同样的方式一

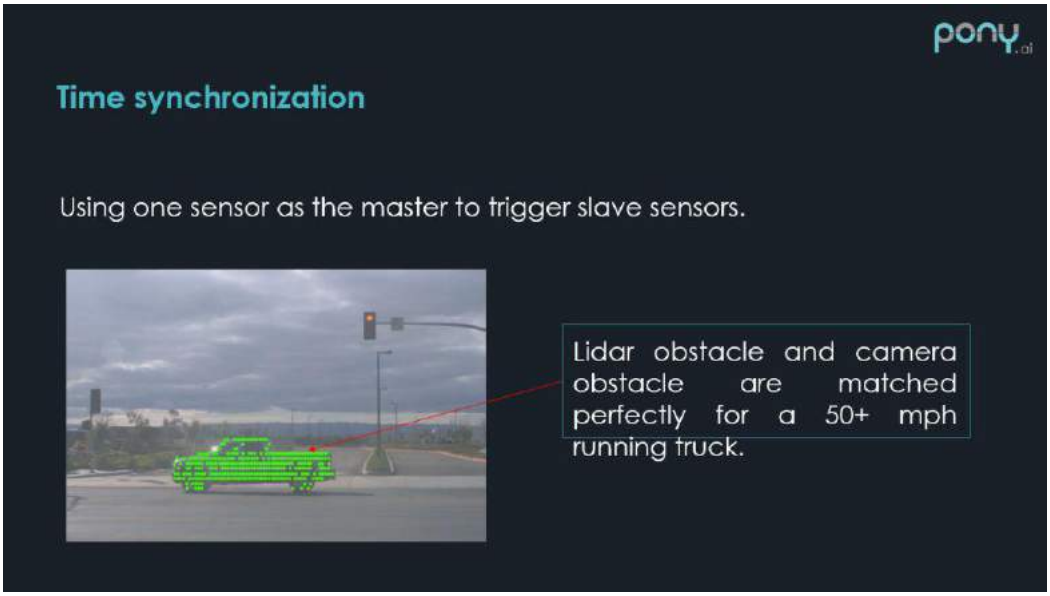
起转，在某一个特定的时段，特定的方向，应该扫到同样的东西，这样来减少快速移动的物体产生拖影这样的问题。

③ 时间同步

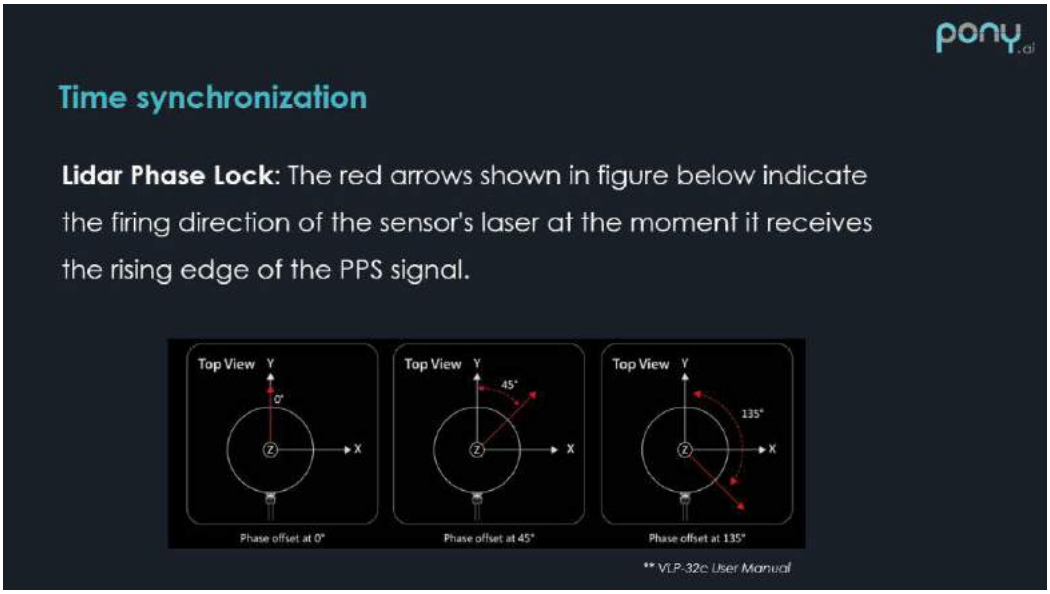


在很多自动驾驶车辆传感器中，大部分支持 GPS 时间戳的时间同步方法。这个方法比较简单，如果传感器硬件支持这些时间同步的方法，拿到传感器数据的时候，数据包中就会有全局的时间戳，这样的时间戳以 GPS 为基准，非常方便。但是，时间戳查询数据会有一个比较明显的问题，举个例子，图中有三个数据，三个传感器和时间轴，不同传感器是以不同频率来采集数据的，以传感器 2 为例，在 T1 时刻，传感器 2 有一个数据，在这个时刻，想知道对应的传感器 1 和传感器 3 的数据是多少，肯定需要去查找，查找的方式是找对应的传感器数据和传感器 2 时间差最近的数据包，然后拿过来用，

这就取决于查的时候，数据包的时间和 T1 时刻传感器 2 数据包的时间到底差多少，如果差距比较大，本身障碍物都是在移动的，这样误差会比较大。



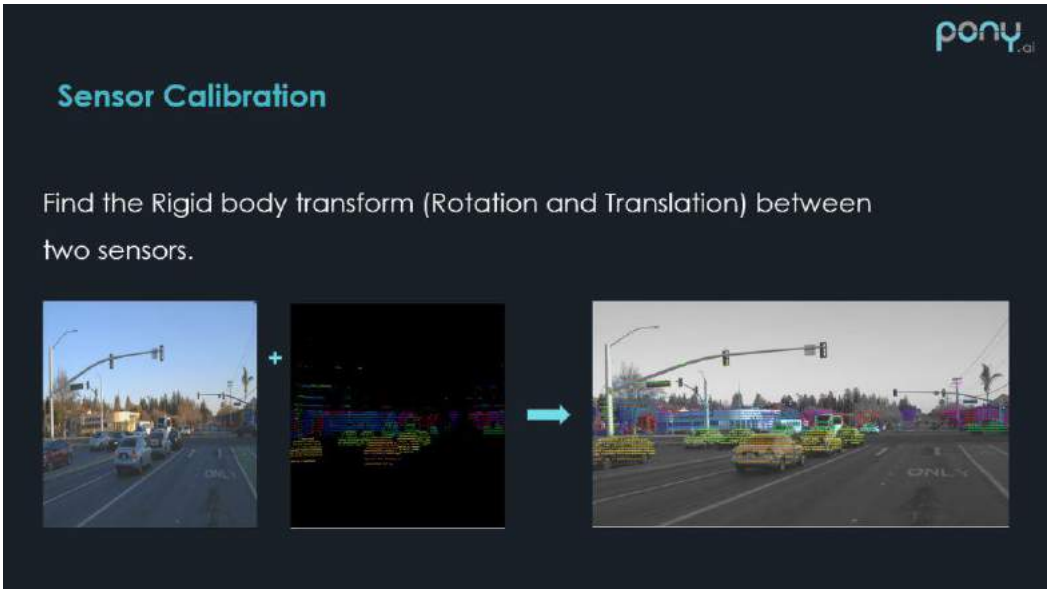
然后就有了第二种时间同步的方法，来缓解刚刚说的这种现象。就是主动数据同步的方法，比如以激光雷达作为触发其它传感器的源头，当激光雷达转到某个角度时，触发那个角度的摄像头，这样就可以大大减少时间差的问题，如果把这套时间方案做到硬件中，做到比较低的误差，那么对齐的效果比较好。如上图所示，这时激光雷达的数据就很好的和摄像头的数据结合在了一起。



刚才说到如果一个自动驾驶车辆用了多个激光雷达，激光雷达之间如何同步，减少扫到同样车在不同时间这样的问题，velodyne 是我们常用的一个品牌，支持一种 Phase Lock 功能，能够保证在某一时刻，所有的激光雷达的角度，都可以根据 Phase Lock 的配置，在固定的角度附近。这样如果用俩个前向的激光雷达都设置一个角度，在同一时刻，扫到的东西应该是类似的，这样一个快速行驶的车，被扫到 2 次的概率就会减少，当然这个办法也不能完全解决问题，比如有个人和我们的激光雷达以同样的频率一起转，那么在激光雷达扫描的点云中，人一直会出现，所以还要通过软件的方法，设置的一些规则或者模型来想办法剔除。

2. 传感器标定

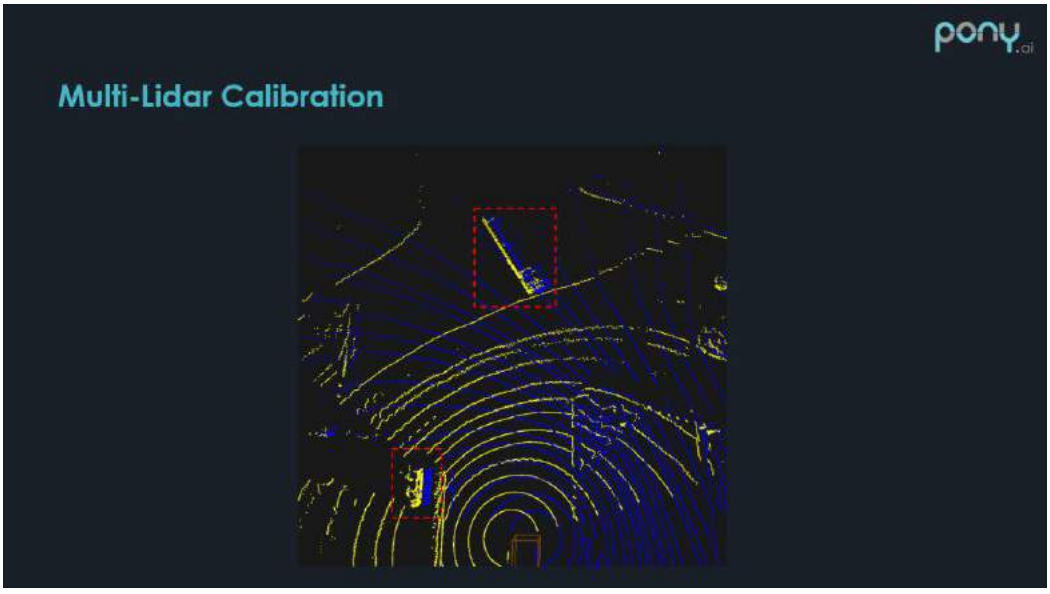
接下来是另外一个比较大的话题：Sensor Calibration 传感器标定。这里主要是指传感器外参的标定。



传感器外参其实就是刚体旋转，因为物体上的俩个点，在经过旋转和平移之后，两个点之间的 3D 位置是不会变的，所以叫刚体旋转，在自动驾驶和机器人上，刚体旋转还是比较常见的。传感器外参的标定就是要找到这样的一个刚体旋转，可以把一个传感器的数据和另一个对齐。相当于把一个传感器测量的数据从其本身的坐标系，通过刚体旋转，转到另一个传感器坐标系，这样就可以进行数据融合了。上图中，左边为图像，中间为雷达，如果有一个比较好的外参工具，可以把 3D 的点投射到 2D 图像上，所有的障碍物的点都可以对应上，相当于把 2D 上的像素都加上了深度的估计。这样在图像质量并不是很高的情况下，可以通过这样的方式把信息补回来。

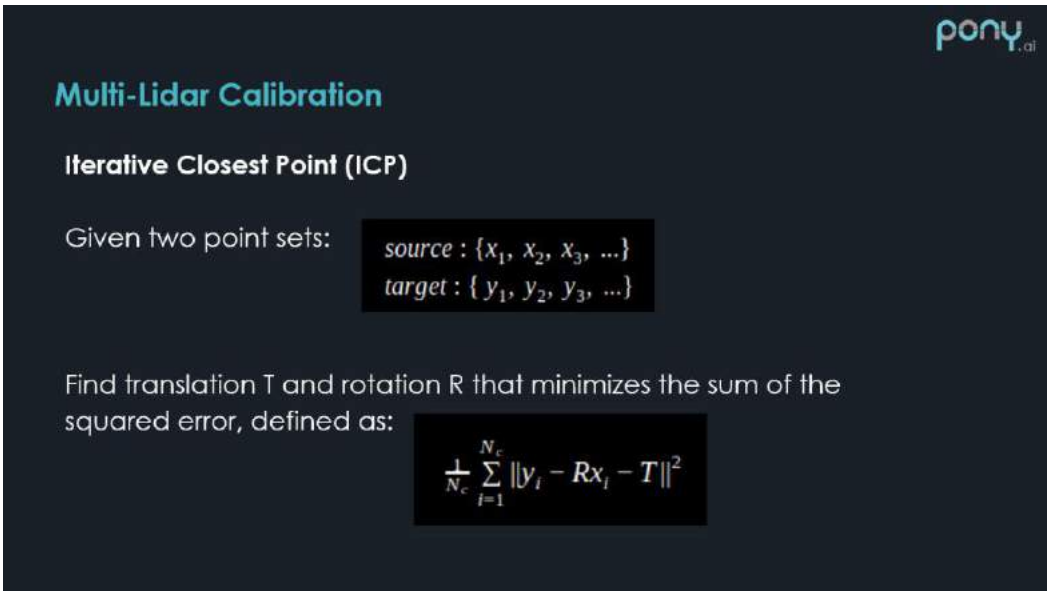
传感器的标定一般有俩种思路，第一种是有激光雷达的传感器标定，第二种是无激光雷达的传感器标定，之所以这么分，是因为激光雷达采集的数据是完整的 3D 信息，空间中的（x，y，z）都是准确的，并不会有信息的丢失，而照相机得到的数据，可以认为是极坐标系下的坐标，没有深度和角度，而毫米波雷达是没有高度的。所以，如果有这样的传感器能够提供完全的（x，y，z）坐标，以此为参照物，其他传感器和激光雷达做绑定的话，会更容易和更准确。Pony 是有激光雷达的，所以今天主要讲有激光雷达的传感器标定方法。

① Multi-Lidar Calibration



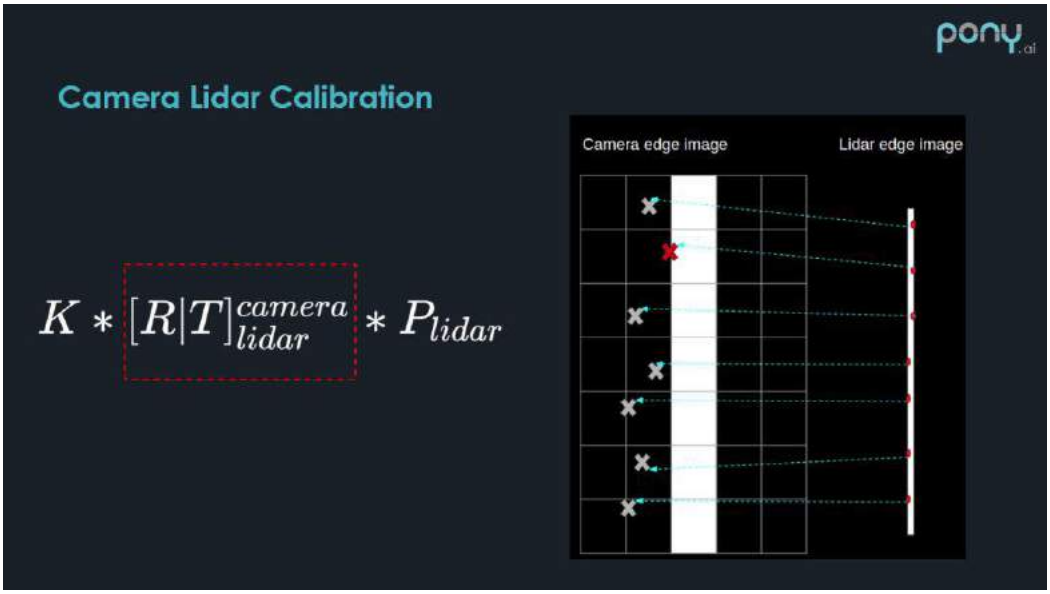
首先讲下多激光雷达是如何标定的，上图可以看到正好用到的是两个前向激光雷达，这两个激光雷达在前向 180°是有比较大的覆盖区域，如果对激光雷

达之间的旋转和平移没有比较好的估计，当把俩张激光雷达的点云放在一起进行感知处理的时候，在红框位置会发现存在比较大的分隔（黄线和蓝线分别代表俩个前向激光雷达），这种情况肯定是不想遇到的，所以需要把多个激光雷达做比较准确的标定。



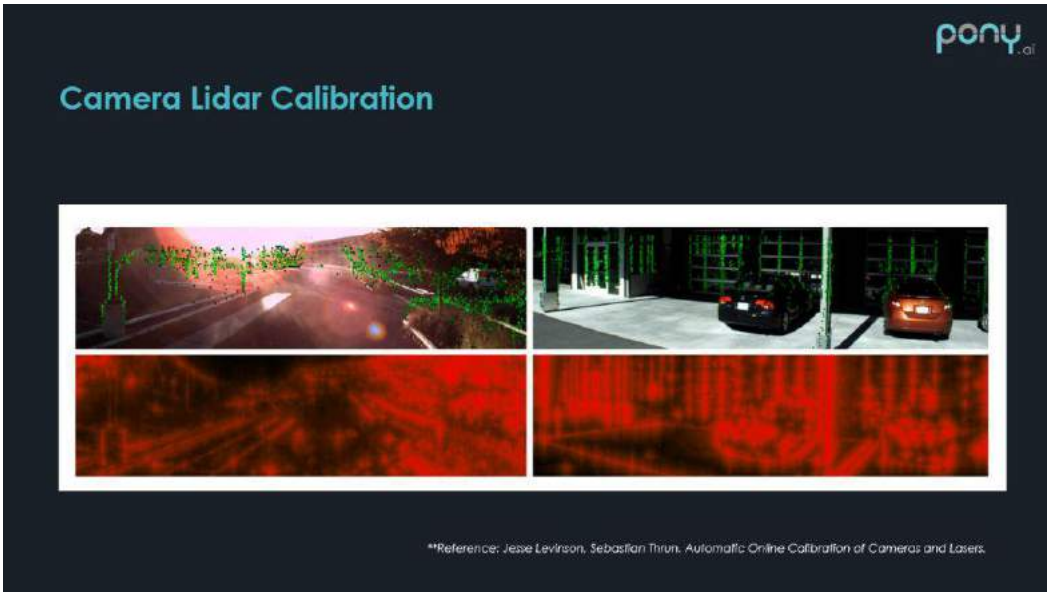
标定的方法是已知的，非常好解决的问题，因为激光雷达本身是有完全的 3D 信息，解决这样俩个数据集匹配的问题，就是 ICP（Iterative Closest Point）迭代式最近点方法。这个方法有很多的变种，感兴趣的同学可以百度或者 Google 搜索下。

② Camera Lidar Calibration



另外一个就是照相机和激光雷达之间的标定。照相机本身是没有距离信息的，那么如何去做标定？同样激光雷达是有 3D 信息的，可以通过标定的方式，把激光雷达投到图像的坐标系中，建立图像的像素点，和激光雷达投影后的点之间的匹配，然后通过某种优化方程，来解决这样一个匹配问题。举一个简单的例子，比如现在要选取一系列激光雷达检测出来的候选点，如何选这些点呢？这些点一定是在图像上比较容易能够识别出来的边界点。选取方法也比较简单，因为激光雷达有距离信息，只需要找相邻俩个激光点之间的距离差，就可以判断这样一个点是不是边界点，我们可以轻易的选出这样的候选点，通过这样的投影方式，红框是我们要求的标定参数，K 矩阵为相机本身的内参，通过这个数学关系，我们可以把刚才 3D 中检测的候选点，投到 2D 上 ,上图中的 X 就是投射后的位置。我们可以根据 3D 投影点和 2D 检

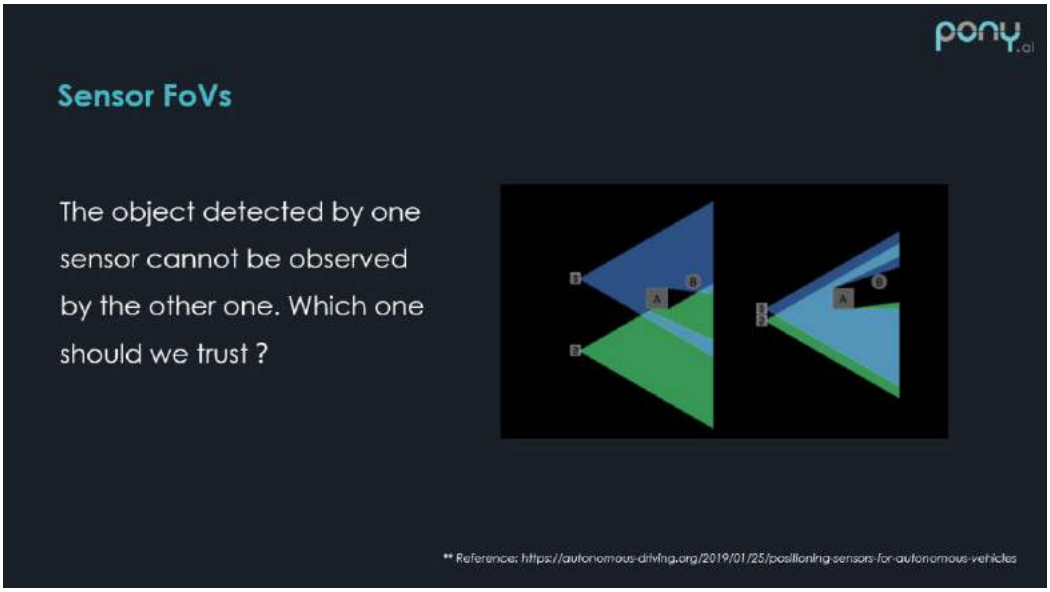
测的边界，进行匹配，然后根据他们之间的距离匹配程度，建立这样一个优化方程，然后解这样一个优化问题，来估计出 Calibration 的参数。



大家如果感兴趣可以参考这篇 paper：Automatic Online Calibration of Cameras and Lasers ,详细的讲述了其中的数值原理 ,可以看到绿色的是 3D 点投射到图像上，是一些边界点候选的区域，如果有一个比较好的标定结果，这些边界点会比较好的和图像匹配起来。

3. 传感器视场角

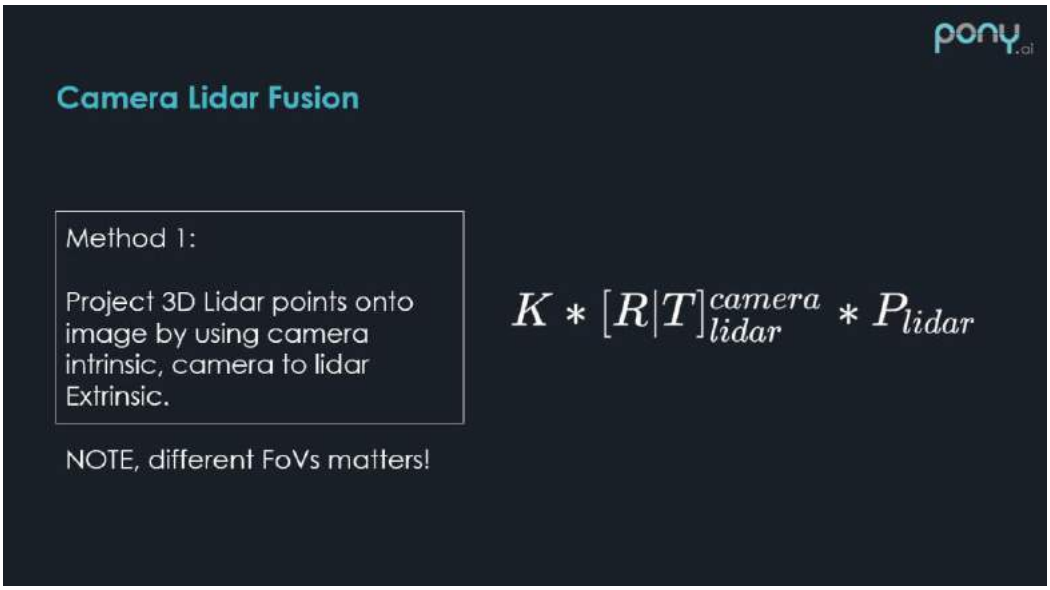
接下来看下传感器不同视场角带来的融合问题。



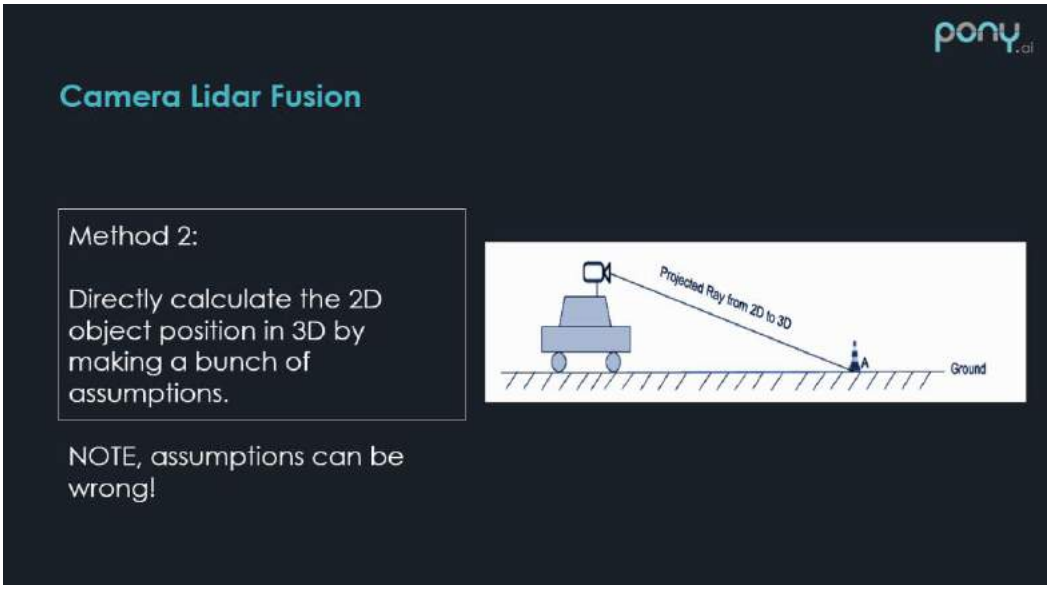
这里有一个简单的示意图，假设在这个位置上有两个激光雷达，它们有各自不同的视场角 ,但是前方有个障碍物 A 刚好在传感器 2 的视场角内把障碍物 B 完全遮挡了，障碍物 B 只出现在一个传感器检测的视场角内部，这带来的问题是：我们到底该不该相信这里存在一个障碍物？这是比较常见的问题，需要我们经过不断的路测，来完善。

如何做传感器融合？

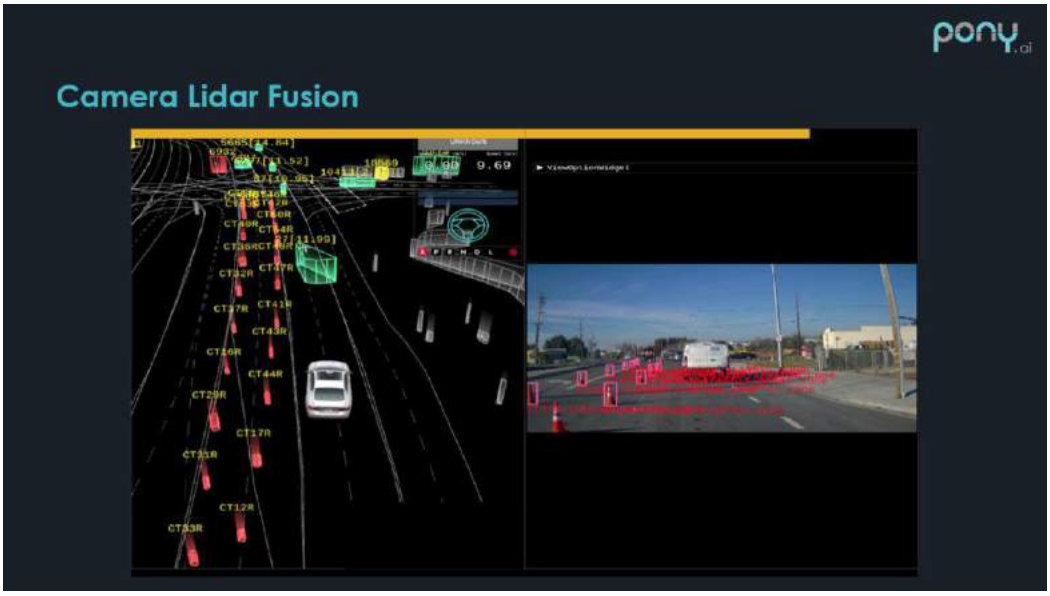
1. Camera Lidar Fusion



首先讲下照相机和激光雷达融合，方法 1 之前大概讲过，就是说激光雷达有（x，y，z）比较明确的 3D 观测，通过标定参数，通过照相机本身的内参，就可以把 3D 点投到图像上，图像上的某些像素就会打上深度信息，然后可以做基于图像的分割或者 Deep Learning Model。需要注意的是，多传感器的时候，视场角可能会不一样，可能会造成噪点或者漏点，这里比较推荐的方法是把照相机和雷达安装在一起，越近越好。

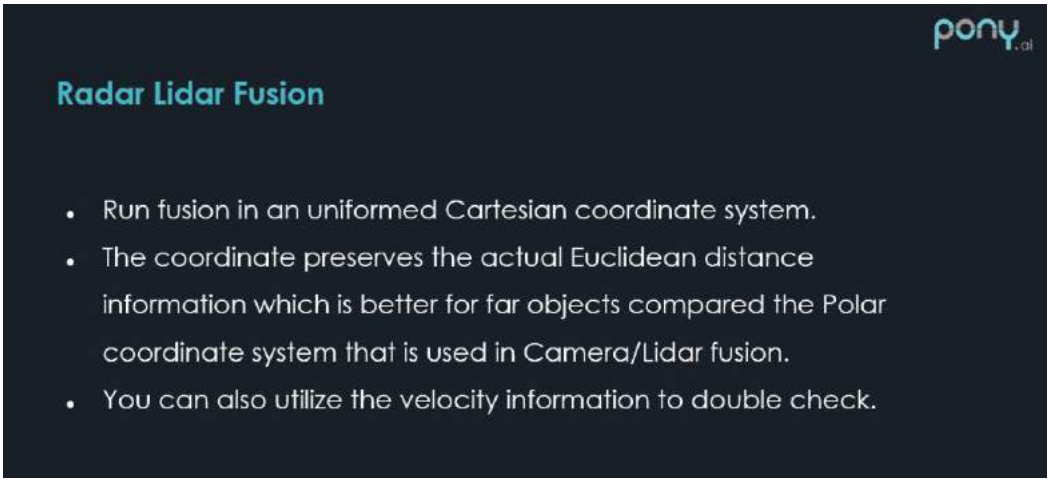


另一个比较直观的方法，是否能将 2D 检测出来的障碍物直接投影到 3D，然后生成这样的 3D 障碍物，这种方法，在做很多的假设条件下（比如障碍物的大小，地面是否平整），也是可以做的，如上图，相机的内参，车的位置高度，都是已知的，这时在 2D 上识别出的每个帧的障碍物，都可以还原成 3D 在照相机坐标系下的一条射线，然后找这条射线在 3D 坐标系下和平面的交点，就可以估计出 2D 障碍物在 3D 上的距离。



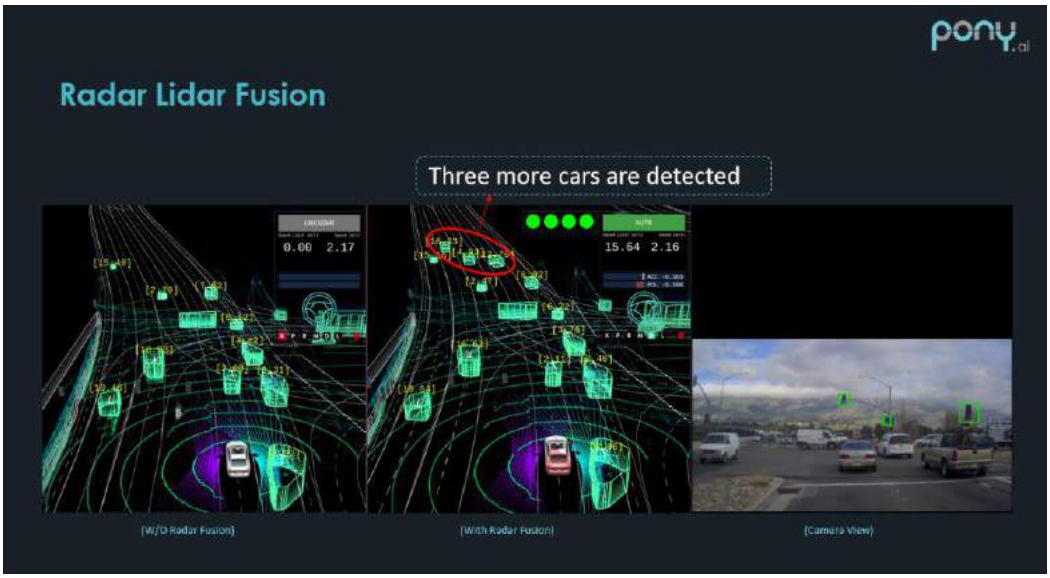
上图为 Pony 在建筑工地旁采集的数据，可看到这些路障都是直接生成到 3D 的（图中有个漏点，也是我们还需要努力提高的）。

2. Radar Lidar Fusion



至于毫米波雷达和激光雷达的融合方式就更简单了。因为在笛卡尔坐标系下，它们都有完整的（x，y）方向的信息，那么在普适的笛卡尔坐标系下，做针

对于距离的融合，而且毫米波雷达还会测速，对障碍物速度也是有一定观测的，然后激光雷达通过位置的追踪，也会得到障碍物速度的估计，这些速度的信息也可以用来做融合，帮助筛选错误的匹配候选集。



这是 Pony 激光雷达和毫米波雷达融合的效果，红圈里的障碍物是 radar 补充的。

当然，不同传感器之间融合的特例还是很多的，比如激光雷达和毫米波雷达融合的时候，可以看到，这个场景是前方有比较高的路牌时，毫米波雷达会在这个位置产生障碍物，恰好激光雷达也有噪音，因为恰好前方有车，这时在牌子底下也会产生噪点，所以激光雷达和毫米波雷达都在这个地方检测出来本不应该出现的障碍物，这时两个传感器都告诉你前方有个障碍物，只有摄像头说前方只有一个障碍物，这时该怎么办？（如果想了解 Pony 具体如何解决的，欢迎大家加入 Pony，我会告诉你答案 o(∩_∩)o ）

总结

总结来说，每个传感器都有自己的一些问题，传感器融合就是说我们要把这些传感器结合起来做取长补短，提升整个感知系统的精度和召回度，今天就分享到这里，谢谢大家。

“老司机”成长之路：自动驾驶车辆调试实践

嘉宾：贺星　编辑：Hoh Xil

随着自动驾驶技术的发展，一辆新车从被改装到上路需要经过的调试流程也有了许多提升。今天，我希望结合自己之前的调车经验来跟大家分享一下我们是如何将系统的各个模块逐步上车、调试、集成，进而将一辆“新手”车培养成“老司机”的。

自动驾驶简介

这是一段来自维基百科的关于自动驾驶的定义：自动驾驶汽车，又称无人驾驶汽车、电脑驾驶汽车或轮式移动机器人，为一种运输动力的无人地面载具。作为自动化载具，自动驾驶汽车不需要人类操作即能感测其环境及导航。

随着自动驾驶技术的发展，行业内涌现出了许多为了应对不同需求不同场景量身定制的自动驾驶汽车，这其中就包括中小型载人汽车，大型载人大巴以及大货车等等。那么怎么衡量一辆汽车的自动驾驶能力如何呢？有这样一张表：

这张表是由 SAE（国际自动机工程师学会，是横跨航空航天、汽车、商用车工程行业领域的顶级标准制定者，起着监督确保产品及服务安全、质量及有效的角色。）提出的。SAE 自动驾驶分级标准第一版 J3016 于 2014 年 1 月发布，最新的一次修订版发布于 2018 年 10 月，标准中 SAE 将自动驾驶技术分为 L0-L5 共六个等级：L0 代表没有自动驾驶加入的传统人类驾驶，而 L1-L5 则随自动驾驶的技术配置进行了分级：分别为辅助驾驶、部分自动驾驶、条件自动驾驶、高度自动驾驶、完全自动驾驶。

今天主要讲的是如何把一辆 L0 的汽车，通过我们的改装，让它具有 L4 和 L5 的能力，也就是我们今天所说的“老司机”。

新手上路

如何教人开车与如何调试自动驾驶车辆有很多共通之处。

这是一个比较经典的自动驾驶车载系统框图，真实的环境经由传感器系统被传递到感知模块，感知模块会基于收集到的原始数据和定位结果，给出对周围环境的感知和预判，然后规划模块结合感知输出结果和当前掌握的地图信息，对底层车辆下发精准的控制指令，而达到控制车的目的，这是一个完整的自动驾驶流程。那么这样一个层层依赖的车载系统是如何一步步在一辆车上被搭建起来的呢？我们可以用一个新手去参加驾校培训的例子来类比：

① 需要有车。“巧妇难为无米之炊”，不管是参加驾校培训，还是调试自动驾驶车第一步都是需要有车，只不过参加驾校培训，车辆是由驾校提供的。

② 需要有过硬的“身体”素质。通过体检是能够参加驾校培训的必备条件，而对应于自动驾驶系统来说就是，我们需要：

- 足够强大的计算平台
- 足够精准的地图和定位模块
- 足够高级的感知模块

③ 在具备了基本素质之后，就可以正式参加驾校培训了，我们将在教练的指导下通过不断的练习来掌握如何控制车辆前进后退，完成倒车急转等动作，这些驾驶技能的提升，对应于自动驾驶系统中的控制和规划模块。接下来我会对每个模块做对应的介绍。

1. 线控改装

线控，即不通过传统机械结构去连接方向盘和前轮，而通过导线，说得通俗点，就相当于你用一台模拟方向盘/游戏手柄玩赛车游戏。

那么，什么样的车适合被改装进而用来做自动驾驶呢？简单讲，不同的改装方案决定了可选择的车辆范畴，一个极端的例子是如果选择采用在驾驶位上放一个仿真机器人来实现自动驾驶车辆的话，所有人类可以驾驶的车辆都可以选择。当然因为在现有技术条件下，这个改装方案在系统响应和安全性上有比较大的问题，我们今天要的讨论是通过改装一些“底子”本身比较好的车辆来实现线控。

什么是“好的底子”？事实上，随着汽车辅助驾驶技术的发展，许多配备有高级辅助驾驶功能的汽车已经完成了底层线控的改造：

- ACC 能够通过自动控制车辆的油门刹车来保证车辆与前车始终保持一个安全距离；
- LKA 能够通过自动控制方向盘保证车辆始终行驶在车道内；
- Valet Parking 自动泊车系统能够在无人干预的情况下完成自动寻找车位停车入库的操作。

这些功能都是通过车辆底层控制协议来控制车辆的机械结构，我们只要想办法获得这些协议，就可以通过调用响应的接口达到我们的目的。由于这种改装方案对于原车的改动最小，响应的安全性也更高，目前被主流自动驾驶公

司广泛接受。右边两张图是 Pony.ai 申请北京路测牌照选用的两款车，而选择的原因也正是因为他们有比较好的改装基础。

在拿到一辆改装完成的车之后第一步，就是对线控能力做测试：

功能测试：是指车辆是否能够按照约定的协议在自动和手动之间自由转换，是否能够完成基本的加减速、方向盘，制动控制是否正常；

性能测试：主要考察车辆的响应/执行时间，反馈时间。就是说我们在对车辆下达指令之后，车辆经过多长时间机械层开始有响应，过了多长时间车辆会给上层反馈它完成了这个指令。因为这个指标与安全性非常相关，也是最受车出厂自身配置限制的一个指标。比如，一辆豪华小轿车和一辆普通大卡车的响应时间肯定是完全不一样的，大卡车的响应或者反馈时间会比小轿车长将近 100ms ,这时如果做自动驾驶的大卡车 ,就必须采取偏保守的规划策略。

2. 地图及计算平台

地图决定了自动驾驶能行驶的区域，通常来说地图采集会是选好车辆之后的第一步，拿我们去驾校学习举例的话，这就是我们熟悉场地的过程，我们需要仔细检查场地里有多少路口，S 形弯道从哪里开始，坡道的坡度多少，红绿灯的大概位置在哪里，做到这些都了然于胸了再去上车实际操作。

高精度地图的细节在上一次讲座已经有所介绍，这里不做介绍。

高精地图和定位在自动驾驶的应用

接下来重点是搭建车上计算平台，主要包含：稳定的供电系统，通信系统，存储系统和计算系统，在设计时要在保证可靠性和高性能的同时兼顾其灵活性，需要系统的软件与硬件有很好的协作。计算平台的好坏是所有车上系统的基础，计算平台的性能高低与车辆行驶的最终效果直接相关，决定了整个自动驾驶系统的上限。

3. 传感器与定位

标好地图和搭建完计算平台之后，就可以基于现在的情况进行传感器的选型。自动驾驶领域常见的传感器包括激光雷达，毫米波雷达和摄像头等，能够提供这些传感器的供应商也非常多，各家公司会基于自身需求做不同的考量，而考量的标准包括：传感器的成本/功耗/性能以及视野。

传感器的安装是感知和定位的基础，不同传感器由于自身特性对于安装朝向/方式会有不同的要求，比如有的需要与车体刚性连接，而有的需要有比较好的减震保护。

安装后要做的第一件事就是标定，简单讲标定包括内参标定和外参标定，内参标定是为了去掉一些传感器自身内部数据的一些误差，而外参是为了让自

动驾驶系统明白各个传感器被安装在什么位置，以便做数据融合和进一步的感知。

定位模块的作用是提供车辆的运动状态，包括车辆的位置，速度以及朝向等，关于定位在接下来的 talk 中会有同事详细介绍，这里不展开了。

自动驾驶复杂环境下高精度定位技术

4. 感知系统

在把传感器上车安装好之后，就可以进行原始数据的采集。这时会开着装好传感器的车到将来要进行测试的区域不断的测试，这时是纯人工操作，目的是采集足够多的道路数据，为感知做准备。数据采集之后会用于模型训练和算法迭代，每一版模型生成并通过评估之后都会在相关的感知模块进行上车

测试，通过不断的迭代感知系统就可以拥有障碍物和简单交通标志的识别能力。关于感知系统需要面对的一些问题以及思考会有其他两位同事在接下来的分享中为大家介绍。

自动驾驶多传感器感知的探索

传统方法和深度学习结合的感知策略探索

讲到这儿，我们的车有了基本的感知能力，虽然我们还没有控制车辆，但是已经可以对于车辆周围的危险进行一些基本的警示，这就是目前一些车辆 L0 级别功能的思路，包括 PDC（我们俗称的倒车雷达），FCW（前方碰撞预警系统），LDW（车道偏离预警系统）。当然，这些系统并不要求有高精度地图以及线控。

5. 控制系统

接下来就是要调控制，车辆控制分为纵向和横向两方面（纵向控制是为了通过油门和刹车来控制车轮转速，而横向控制是通过方向盘来控制车辆行驶方向）。在控制系统上车的初期我们也会按照先纵向后横向的方式去调试，场地一般都是在封闭道路上。在车辆通过横纵向独立测试之后我们会进行集成测试，先由人手动开一个任意轨迹，然后将其中经过的一些位置点记录下来当作控制模块的 input，通过不断调试让车辆能够自动沿着记录下来的轨迹行驶。这时，我们的车拥有了模仿能力。如果还拿学驾照比喻，此时我们已经可以完成教练演示过的操作，也就是说我们可以去考科目二了。

这就是 L1 和 L2 的思路，他们的共同点是没有强大的规划模块，但是基本完成了对感知系统和控制模块的掌握。

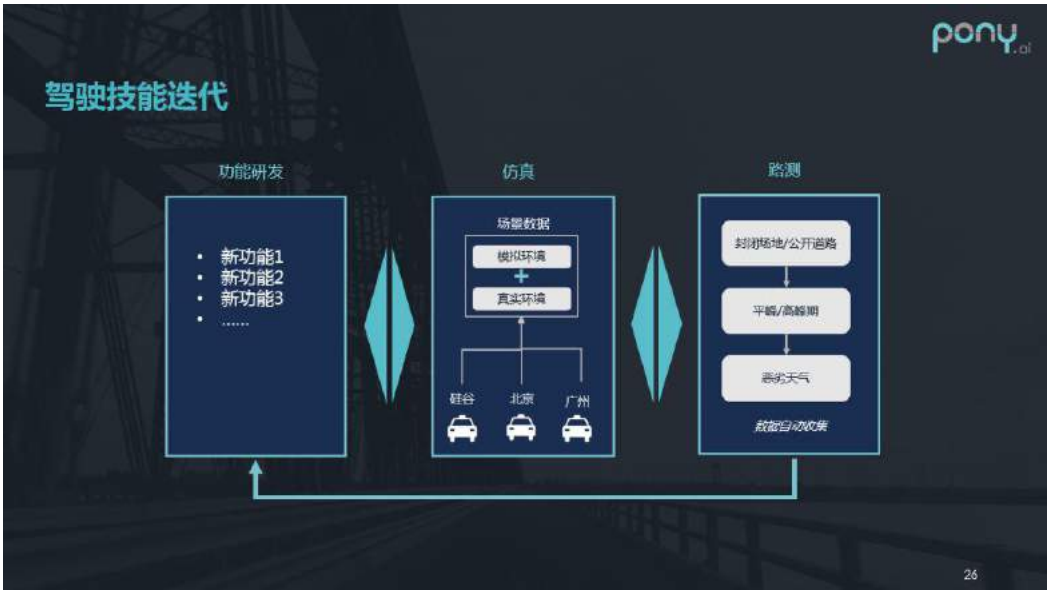
6. 规划系统

真实的路况远远比直线加速和简单循迹复杂，为了实现车辆真正的自动驾驶，我们还需要拥有足够强大的规划系统。简单讲，规划系统的作用就是基于当前车辆周围环境决定车辆接下来怎么走。这有点像驾校考试的科目三，教练会给学员下指令变道还是掉头，决定什么时候应该将车辆加速到 50km/h，这在实际自动驾驶车的调试过程中也是类似的，只不过我们会充当教练的角色，而我们的自动驾驶系统就是学员。

一些配备了一些 L3 级别辅助驾驶功能的汽车拥有了类似的简单的规划能力，比如自动泊车系统。车辆配备的功能可以完成的规划/决策越复杂，他的自动驾驶级别也就相应越高。

通过了前面的测试，也就可以说我们的系统考过了科目三，可以持证上路了。上图左边是我们常见的驾驶证，右边是北京颁发的自动驾驶路测牌照。但取得驾驶证/测试牌照与会开车并不等价，生活中大家周围都会有一些持证十多年但是从未开上过公开道路的“资深”司机，这说明如果要进一步提升能力成长为驾轻就熟的“老司机”，仍然需要不断地练习。

7. 驾驶技能迭代



这是 Pony.ai 内部功能开发的一个基本流程，一个新功能从 idea 被提出到代码实现基本需要经过以下流程：首先，在功能开发完成之后需要先在内部的模拟环境中充分测试，我们的模拟环境包含丰富的测试场景，这其中的测试场景有的来自于程序生成，有的来自于之前的真实路测。在完成仿真测试之后，需要进行实车路测，为了保证系统的平稳迭代和发布，我们会随着功能的稳定程度的提升一步步增加路测环境的复杂程度。在实车路测的全部数据都会被很好记录并被有效追踪，其中暴露出来的问题会被重新加入到仿真环境的测试样例中，直到现有版本通过所有测试，才可以进行下一次路测流程。只有当一个功能在之后的路测中被测试了足够长时间并且没有问题，我们才会将它作为稳定版本发布。

正是经过这样一步一步的迭代，让我们的车从最开始只能开直线、转弯、掉头，到最后能完成一些超车、加塞等高级的“老司机”动作。

成为“老司机”之后

考虑到自动驾驶的目标是用机器代替人的操作，在教会一辆车成为“老司机”之后理论上也可以教会一群车成为“老司机”，也就说，我们可以建立一支自动驾驶车队，而车队中的驾驶员开车一样驾轻就熟。建立这样的一个车队，对于整体解决方案的稳定性要求是很高的，它同时也带来了很多好处，除了数据采集速度成倍提升以外，可以同时在多辆车上测试不同的功能或者进行对照试验，算法的迭代效率也大大提升了。

另外，我们可以将教自动驾驶系统开小轿车的经验移植到公交车，大卡车上，就像人类驾驶员增持驾照一样，这同样是非常值得尝试的方向，也是我们在努力的方向。这就是今天的分享，谢谢大家。

安全舒适两不误：探讨优化算法在规划控制中的应用

嘉宾：孙浩文　编辑：Hoh Xil

导读：自动驾驶技术研发对于“安全第一”的追求是毋庸置疑的，但是这中间可能就忽视了舒适性。

因此，今天我想给大家分享的是，自动驾驶研发如何在兼顾安全性的同时，保证乘客体验的舒适。往细一点说，就是如何通过优化算法在规划和控制模块中找到安全性和舒适性的平衡点。

安全性和舒适性的定义

安全性的定义其实比较好理解，一般包括不发生碰撞和遵守交通规则（交通灯、停车标志、限速）。

而舒适性则受下面两个指标影响：

- Jerk，即加速度的变化率。当加速度的变化越小，乘客体感越舒适。
- Curvature rate，即曲率变化率。当曲率的变化率越小，乘客体感越舒适。

在二者的定义基础上，**“安全舒适两不误”也就意味着在遵守交规且不发生碰撞的基础上，控制好行驶时的加加速度和行进路线的曲率变化率。**

规划和控制的定义

规划（Planning）和控制（Control）是自动驾驶最底层的部分，**决定了车在路上如何行使**。

一般来说，规划所需要的输入（Input）信息包括：地图信息，起点终点，障碍物预判，交通标志以及感知信息，即周围的物体位置、大小、速度、方向等。

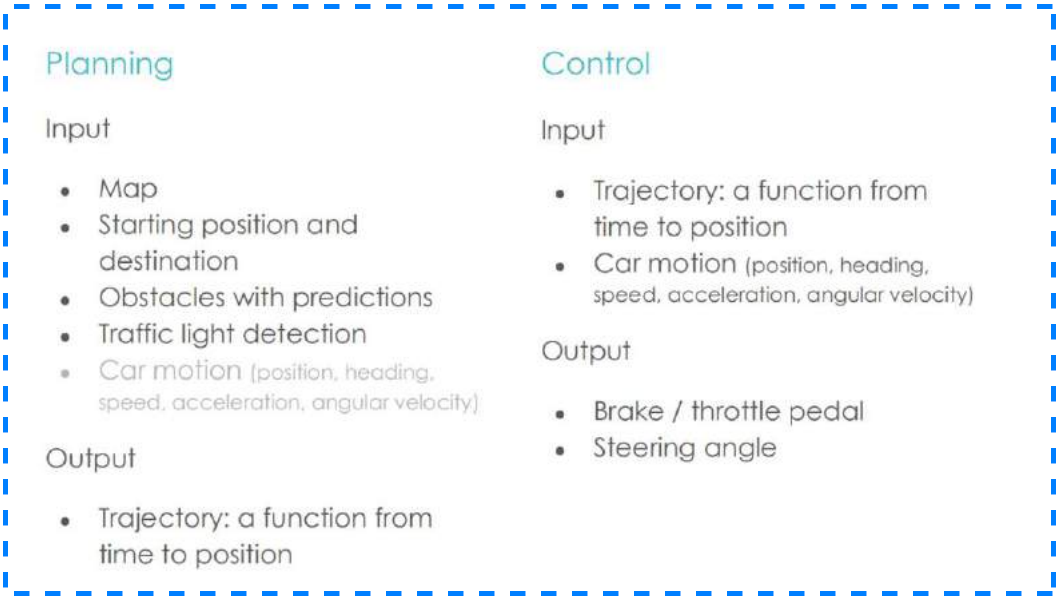
借助这些输入信息，规划模块可以输出（Output）一段时间的行驶轨迹（Trajectory）,即某一个时间无人车行驶到到某个位置的函数(Xt~Yt)。

而规划输出的轨迹正好是控制（Control）模块的两项输入之一，另一项输入是车辆自身的状态，包括车的位置、heading（行驶朝向）、速度、加速度、角速度等信息。

而控制模块会输出以下两类信息：

- Brake/throttle pedal，刹车和油门
- Steering angle，转向角

简单来说 ,控制的目的是让自动驾驶车能够精准实现规划模块给出的行驶指令。



上面就是规划和控制的简单定义。当然，这里我也留下一个思考（上图标灰的地方）：对于规划模块而言，是否必须知道车辆的实时位置信息？

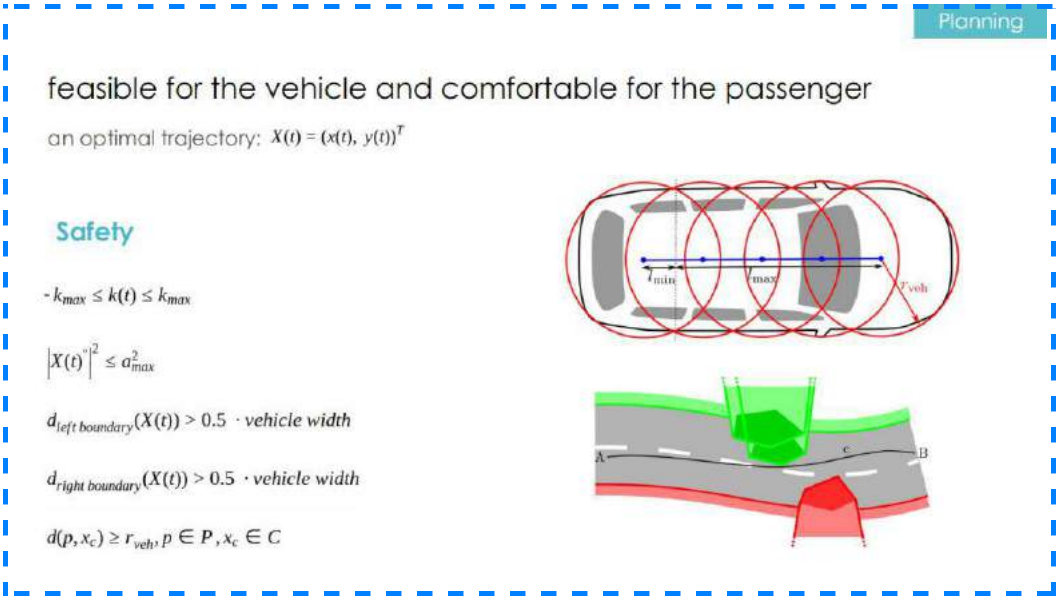
答案是，有无都是可以。规划和控制模块都能有相对应的做法和方案。

规划上，如何实现安全性和舒适性？

安全性

前面已经说到，规划（Planning）为的是输出一个 trajectory（时间到位置的函数）：X(t)=（x(t)，y(t)）T，而这个行驶路径需要同时保证安全性和舒适性。

实际上，规划（Planning）是一个优化问题。对于一个优化问题来说，我们需要通过设计 constraints （约束条件） 来保证行驶安全。



上图里列举了一些最基本的约束条件。

首先，规划模块不能忽略车的物理特性，比如方向盘的最大角度，对应的就是曲率的最大值、最小值等。

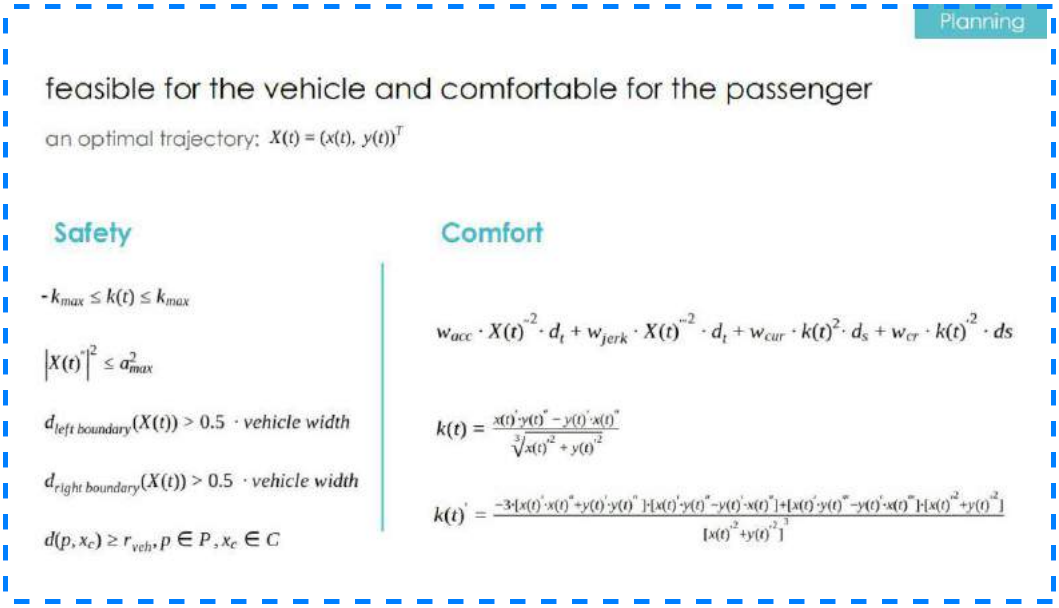
其次，明确车辆刹车和油门的极限，这对应的就是加速度的极限。

第三，当自动驾驶车在道路上行驶时，为了不出现压线的情况，车到左右 boundary（车道线）的距离也需要纳入规划中。

第四，自动驾驶肯定不希望车辆发生任何碰撞。我们可以通过取车轴上一系列等半径的圆来覆盖车身，并采用若干个凸的多边形表示车身周围的物体，那么，当车轴上所有的圆心到凸多边形的距离大于半径时，自动驾驶车就不会与凸多边形（障碍物）发生碰撞。

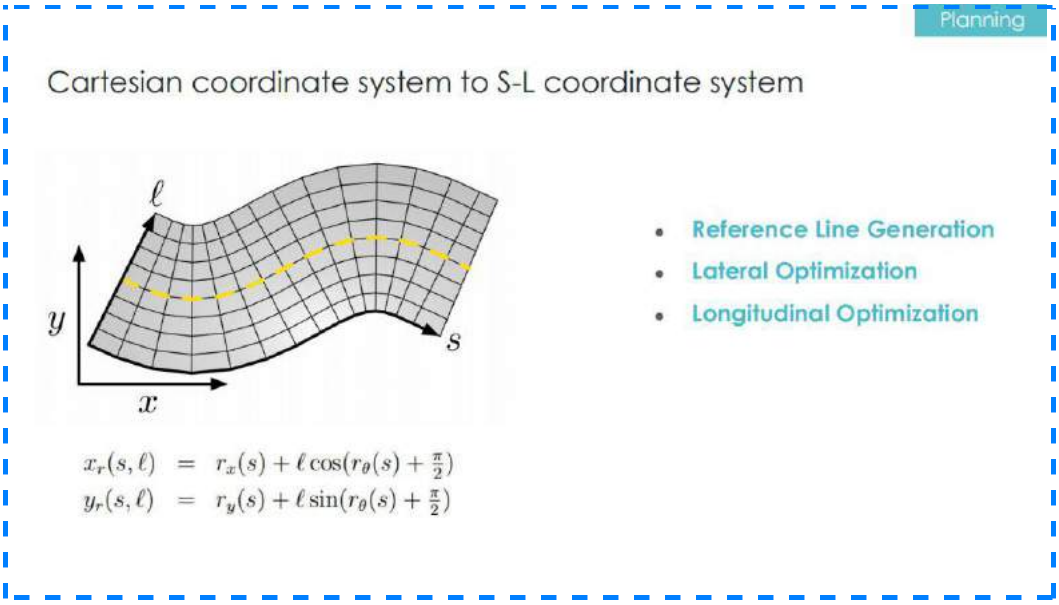
舒适性

介绍完安全性，下面介绍规划模块是如何实现驾驶的舒适性。前面提到过，舒适性越高，意味着加速度变化率越小，以及曲率的变化率越小。那么对应的 cost function（代价函数）就比较好设定了，即加速度+加速度变化率+曲率+曲率的变化率。



虽然整个模型设计起来比较简单，但是 constraints（约束条件）和 cost function（代价函数）的求解却比较复杂（如上图），尤其是系统实时求解 k（t）和 k（t）'的耗时会比较长，解起来也比较难。这时候我们可以换个思路，把问题转换下。

前面也提过，规划问题其实是为了解出一个时间到位置的函数，而 L4 级自动驾驶车一般会行驶在结构化的道路上 ,这也意味着规划模块具备地图的信息，那么问题就可以进行转化——把（x，y）坐标系转化为下图的（l，s）坐标系。



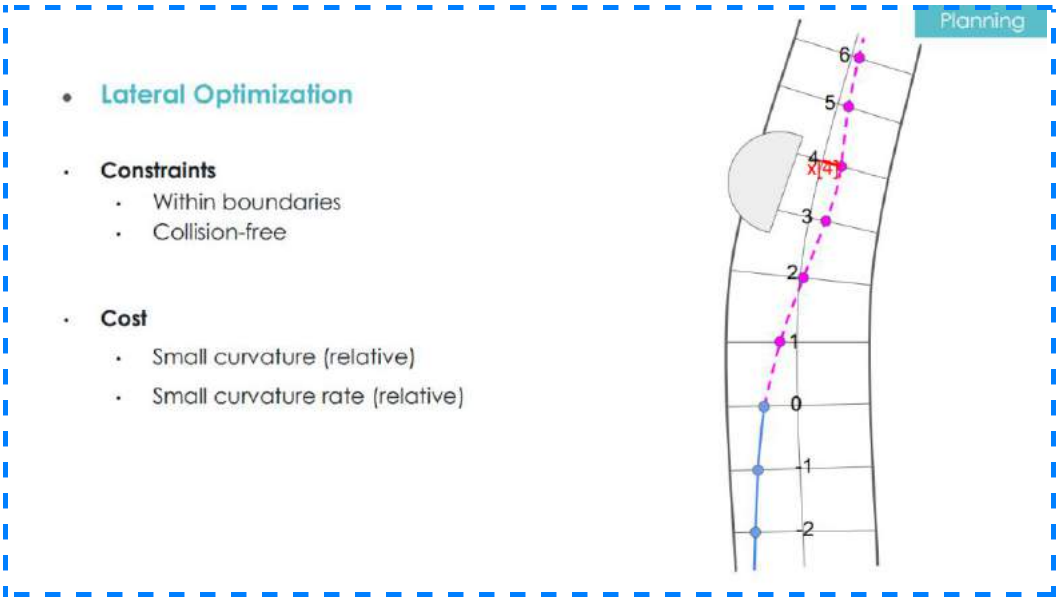
s 沿着道路中线（reference line）的方向，l 则是垂直于道路中线的方向。于是，问题就转化成了求解 s（t）和 l（s）的问题，我们希望得出，自

动驾驶车在 t 时刻应该走到 s 上的哪个位置；以及走在这个位置时，该车距离 s 的左右偏离是多大。

最终，一个复杂问题就转化成三个子优化问题：**一个横向优化、纵向优化以及中线问题（s 方向是车辆的纵向规划，l 方向为横向规划）**。

接下来我想仔细说一说这三个子问题。

子问题一：横向优化



上图中右侧展示了一条路，中间的线是 reference line（参考线），其中“0”点就是车当前所在的位置，“-1”，“-2”是车之前的坐标位置，接下来的“1，2，3，4”……就是规划模块的输出结果，也就是车未来的位置。

这条路的左右两边是 boundary（道路边界/车道线），图中的半圆对应着路面上的障碍物。横向规划求解的是正是每一点的 l 坐标，也就是到中心线的偏离量。

对于横向优化问题，我们同样可以从安全性和舒适性两方面考虑。

安全性方面，取决于横向优化问题的 constraints（约束条件），一般要求 l 不能太小，也不能太大，即不能压左边线，也不能压右边线，同时不能碰到左右两边的障碍物。

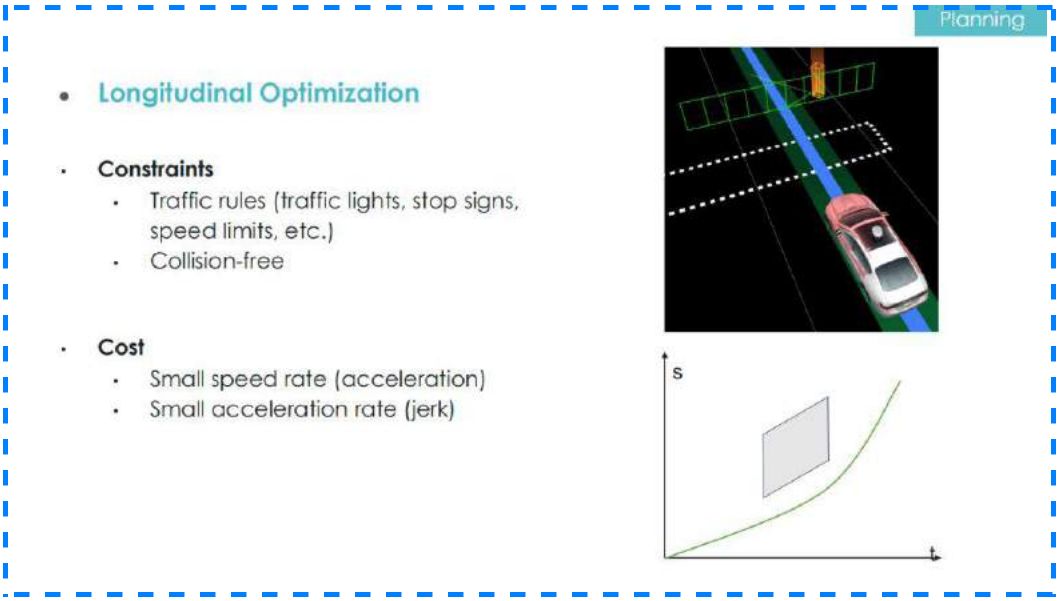
舒适性方面，我们则设计成 cost function（代价函数），一般要求路线的曲率越小越好，曲率变化率越小越好。如果我们的参考线是一条完美的参考线，那么曲率就可以用一个相对曲率来表示。

对于一个优化问题来说，cost function（代价函数）变成了标准的二次函数，同时，所有的 constraints（约束条件）都是一次的。

那么，这就成了一个标准的 QP（Quadratic programming，二次规划）问题，不仅如此，这还是一个 box constraint（闭箱约束）的 QP 问题，我们就能够非常快速求出优化的结果。

子问题二：纵向优化

纵向优化求解的则是 t 到 s 的函数。



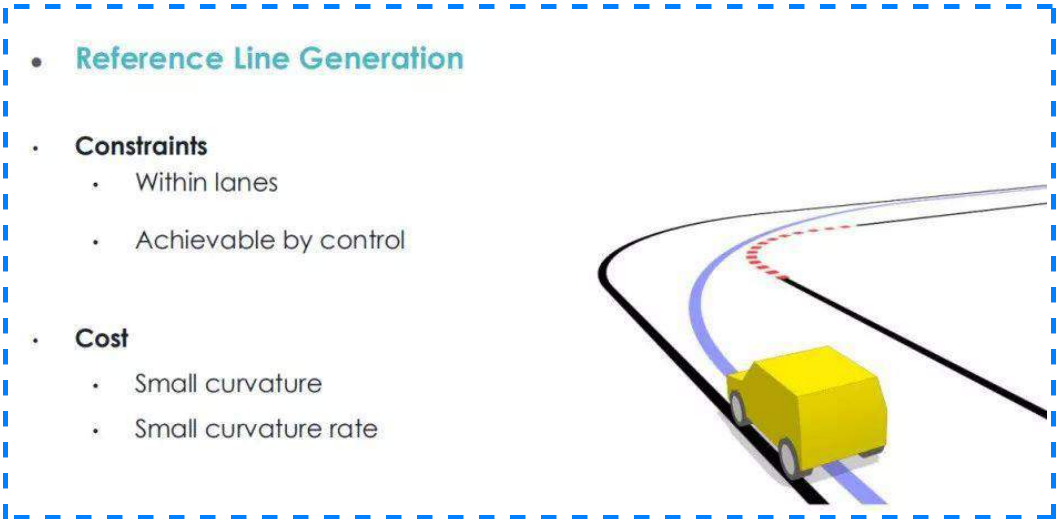
对于纵向规划的来说，安全性要求我们保证车辆不能发生前后方向上的碰撞。以上图右侧为例，此时有个行人正在过马路，规划模块会将行人的行为可以转换成（t，s）坐标，展示在坐标图里（上图）是一个平行四边形。

此时，面对这个行人，如果自动驾驶车选择让路，纵向规划解出来的曲线就应该在平行四边形的下方；如果选择超过这个行人，那么这条曲线应该从上面绕过去。

在纵向规划上，舒适性体现在加速度越小越好，以及加速度的导数越小越好。可见，纵向和横向规划涉及的 constraints（约束条件）和 cost function（代价函数）都非常相似。

子问题三：中线生成

同样，中线的生成也是一个优化问题，自然也会涉及到 constraints（约束条件）和 cost function（代价函数）。



中线设定的约束条件首先是不能压边界（车道线），同时要符合曲率的限制。对于 cost function（代价函数）来说，希望达到的目标是中线的曲率越小越好，曲率的变化率越小越好。这样来看，中线问题也是求解 k(t) 和 k(t)'的函数问题。

问题似乎又绕回去了，为了舒适性，我们还是需要求解非常复杂的方程吗？

并不是。由于中线是可以离线生成的，所以不需要在线计算，这些问题能放在服务器中进行并行的计算。

这里，我想增加一个值得思考的问题。前面提到，生成中线的前提是具备道路的信息，即道路左右的 boundary（车道线边界）。然而当自动驾驶车处在一个路口内，路面并没有车道线，这种环境下应该如何处理参考线呢？欢迎大家留言探讨。

控制上，如何实现安全性和舒适性

毕竟自动驾驶控制的是一辆车，而不是一个点，所以在详细介绍自动驾驶控制模块之前，我想先介绍一个模型。

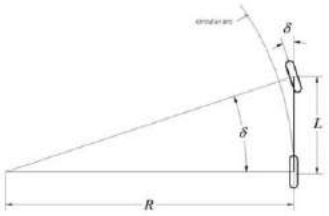
Bicycle model （自行车模型）

控制的模型有很多，而且非常复杂，需要考虑车辆的很多因素，包括引擎，变速箱，轮胎，悬架等等。为了讲好这个问题，我先给大家介绍一个最简单的模型 Bicycle model（自行车模型）。

Control

Bicycle model

Bicycle model simplifies the four wheel car by combining the two front wheels together and the two rear wheels together to form a two wheeled model, like a bicycle.

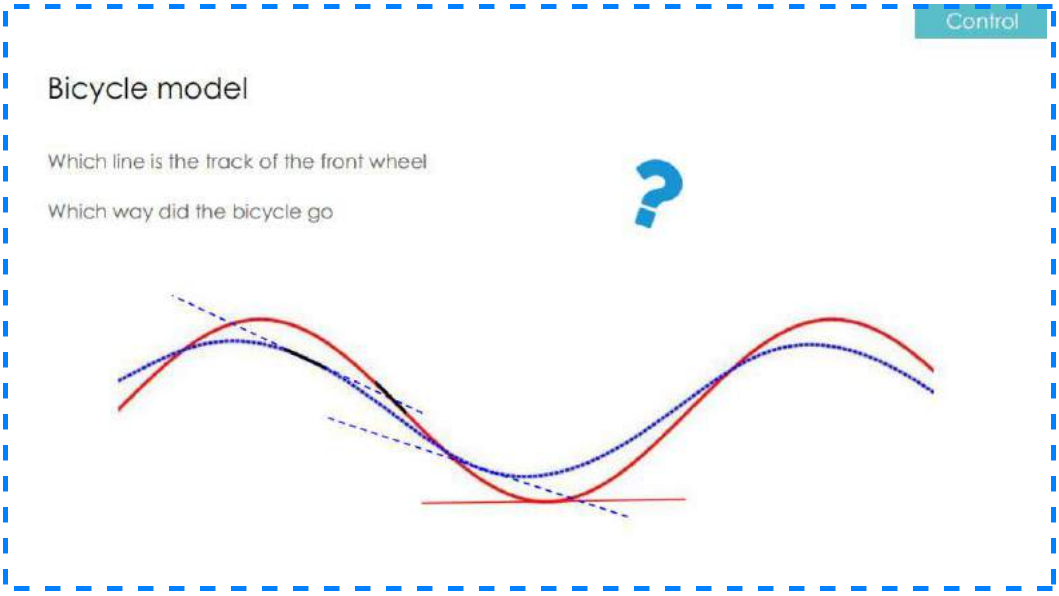


$$\tan(\delta) = \frac{L}{R}$$

首先，假设自动驾驶车辆的车宽为 0，可以想象成是两个前轮靠在一起，两个后轮靠在一起（上图）。

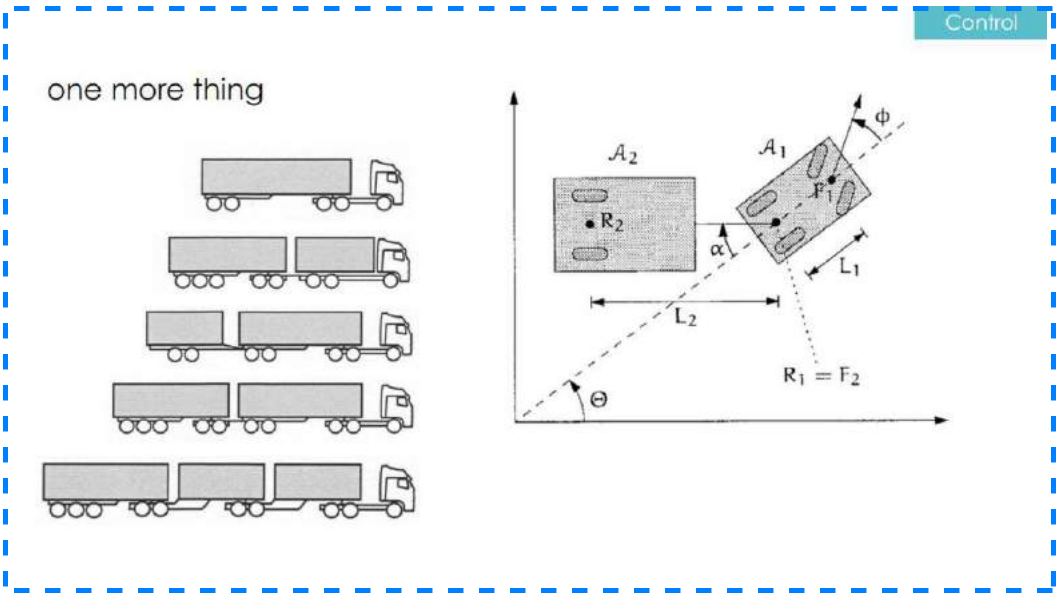
这个假设就意味着前轮的转角和转速都是一致的 ,当然现实中车辆的四个轮子的转角和转速都是不一样的 ,所以这个模型只是在做一个最简单的假设。基于这个假设，当得知前轮转角转了δ时，我们也可以通过几何关系算出后轴的转弯半径是多大。

前面把车假设成了“自行车”，下图展示了它的行驶轨迹，那么，哪一条是前轮的，哪一条是后轮的？车是往左边行驶还是右边行驶？



首先，基于自行车模型的假设，后轮的侧向速度为 0，所以后轮永远都是朝向前轮行驶的，在红线上画一条切线，会发现它没有穿过蓝线，所以它不是后轮。在蓝线画一条切线，它穿越了红线，因此蓝线是后轮的轨迹。

车往那边走呢？我们继续在蓝线上画一条切线 ,这条切线和这条红线还是有两个交点。实际上，交点和切点的距离就是车的轴距，并且由于轴距是固定的，所以这辆行驶的车就如图中黑线所示。这个问题的答案是：红线代表前轮，蓝线代表后轮，以及它是向右行驶的。

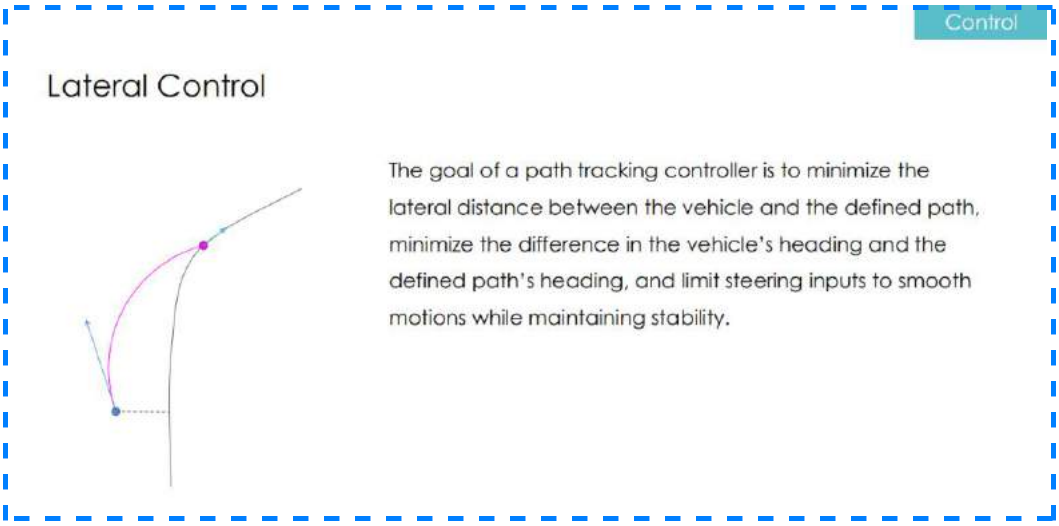


总结来说，对于只有两个轴并且通过前轮转向的车辆，一般可以转换成 Bicycle model（自行车模型）。但在真实路上行驶的往往不只是小汽车，比如还有货车，而货车可以分为两个部分，车头与后挂。

对于货车的控制，如果还是采用自行车模型进行假设（上图），则 A1 是车头，A2 是挂车，F1 为车头的前轴，R2 认为是挂车的后轴。在这个模型中，我们认为挂车的前轴和车头的后轴是同一个点 R1=F2，车头的轴距是 L1，挂车的轴距是 L2，当得知车头的前轴转了ϕ角度时，控制模块可以得到车头后轴的转弯半径。

那么 ,挂车的后轴轨迹会呈现出一条什么样的曲线？这个问题也欢迎大家留言讨论，毕竟现实生活中，路面上不止有一种车，自动驾驶从业者可以多思考下，各类车型的运动行为应该是什么样的。

横向控制（Lateral Control）



上图中紫色曲线是 Planning（规划模块）输出的目标轨迹，图中的蓝点是自动驾驶车此刻的位置，蓝色的箭头是车现在的行驶方向。

可以看到，自动驾驶车辆与规划轨迹存在一个侧向的偏差，因此横向控制的目标就是使车辆实际轨迹和目标轨迹的侧向偏差越小越好，同时车在某个时刻的 heading（朝向）与规划轨迹曲线对应点朝向的误差越小越好。

预瞄算法就是横向规划中常用的一种：首先会选定一个预瞄点，横向控制的目标就是到预瞄点时，侧向误差和 heading（朝向）的与目标轨迹的误差越小越好。这里的控制变量就是刚才说的曲率，所以横向控制又转化为了一个优化问题。

纵向控制（Longitudinal Control）

同样的，在纵向控制上，我们也可以选一个预瞄点，希望达到的目标是自动驾驶车速（v）与规划轨迹的误差越小越好，位移（s）与规划轨迹的误差越小越好，于是纵向控制也可转为优化问题。

总结来看，不管是对于 Planning（规划）还是 Control（控制），最终都可以转化为优化问题。

自动驾驶规划模块对舒适性和安全性的探索 ,重点就在于设计优化问题中的 constraints（约束条件）和 cost function（代价函数）。

而控制模块则在于预瞄点的选择 ,控制模块最终的目标就是自动驾驶车更精准地跟随规划轨迹 ,当预瞄距离越近 ,控制就越精准 ,当预瞄距离越远，控制量更加 smooth（平滑），体感可能更加舒适，但误差可能越大。

最终，我们都能坐上既安全又舒适的自动驾驶车。今天的分享就到这里，谢谢大家。

——文中部分图片来自网络。

打造最可靠的自动驾驶基础架构

嘉宾：莫璐怡　编辑：Hoh Xil

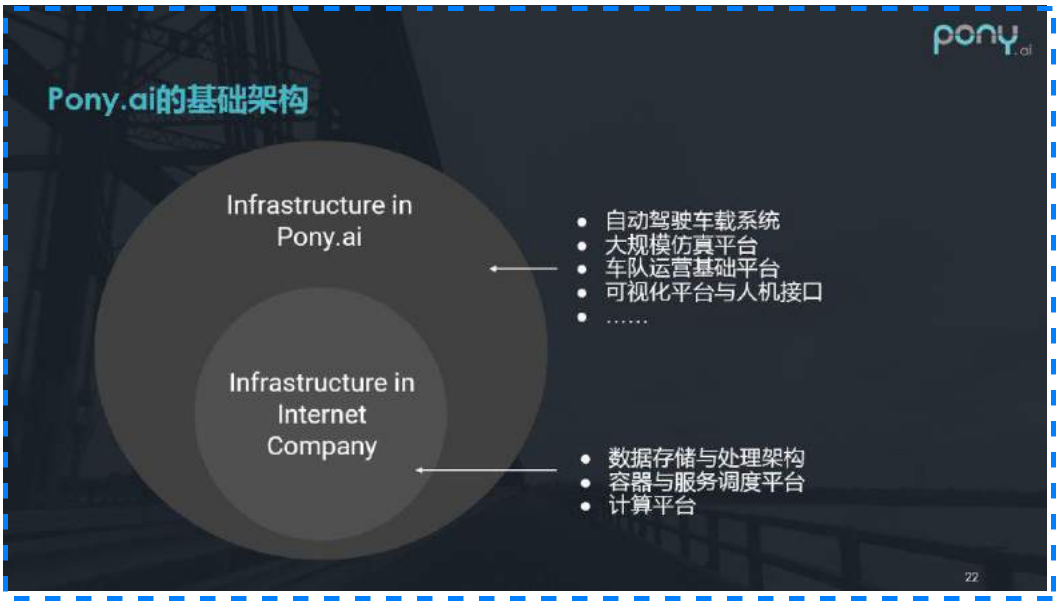
导读：本次分享的主题为**打造最可靠的自动驾驶基础架构。**主要内容包括如何做 Pony.ai 自动驾驶系统的基础架构，涉及到的技术困难，以及我们是如何克服的。

首先先了解下传统互联网公司的基础架构：

- 数据基础设施，会包括大规模的数据库、分布式的文件系统；
- 计算平台，可能会需要大量的服务器、大数据平台、容器的管理机制；

- Web 服务管理，同时还会有各种各样的 Web Service，不停的迭代来满足新的业务发展。

这是传统互联网公司要做的事情，但是对于自动驾驶公司和 Pony.ai，在这样的架构基础上我们还会做哪些事情？



这是 Pony.ai 的基础架构，包含了所有传统互联网公司要做的事情，除此之外，还需要做如下事情：

自动驾驶车载系统。如何支持各种各样的 AI 技术、算法，如何控制车辆，这都依赖于自动驾驶车载系统来完成。

大规模仿真平台。Pony.ai 每天至少会跑 30W 公里的仿真测试（很多自动驾驶公司一年跑的里程可能只有百万级别），这点对于自动驾驶测试来说非常重要。

车队运营基础平台。Pony.ai 要打造自己的移动出行服务，需要基础平台来支持 Robotaxi 的运营。

可视化平台与人机接口。可视化平台是帮助我们了解系统到底是如何思考、运作的，或者当测试工程师做各种测试的时候都依赖于可视化平台；人机接口，自动驾驶车辆最终是要提供出行服务，是有乘客在里面，这时会有一个可视化的界面，来告诉乘客车所感知的周围环境，以及接下来的驾驶操作等等信息，同时还会提供人机交互的功能，让乘客也能控制车辆，比如输入目的地，或者需要停车等等。

……

Pony.ai 的目标是打造自动驾驶移动出行平台，我们希望可以在不同的城市，可以提供大规模自动驾驶车辆的运营，那么我们的基础架构会面临以下挑战：

车辆数量的增加。目前广州已经有几十辆车在进行测试，同时还在不停的增长着；

运营区域的扩大。刚开始只是在很小的区域进行测试，目前已经在几百平方公里的区域进行测试；

数据量的增长。我们有很多的传感器，以及车辆和运营区域的增加，都使得数据量的增长非常非常非常大；

工程师数量的增长。目前 Pony.ai 有广深、北京、美国四个 office，工程师的数量每周都在增长，所以导致模块数量和内部代码的数量也在增长。

所有的这些增长都要求我们的技术栈是具有可扩展性的，来满足快速增长所带来的挑战。

刚刚讲了整个基础架构，其中重要的一点就是车载系统，在讲车载系统之前，先简单介绍下自动驾驶系统：

传感器及其他硬件：激光雷达、高分辨率摄像头、毫米波雷达、GNSS/IMU、运算平台，我们可看到图中标了不同的颜色，目前这些传感器是通过 Supplier Partner 来得到的，我们自己不做传感器，我们需要去购买他们的产品，但是购买之后需要做数据进一步的分析和整合，然后做后面的处理，然后对于运算平台除了 supplier 的一些应用外，我们自己也会做一些优化。传感器主要要做的事情就是接收真实世界的数据，然后传递给 Pony.ai 自动驾驶系统中。

自动驾驶系统：首先，要做传感器融合，进行时间同步，将多传感器的数据融合在一起；然后是感知模块，用来感知周围的环境有什么样的障碍物和物体；接下来会进行行为预测，预测这样的障碍物或物体之后的行为会是什么样的；然后才到我们的决策规划模块，按照之前的预测来决定之后车辆的动作，如急刹车、让路、超车等动作；最后，就是我们的控制模块，他会按照决策规划模块，告知我们的系统要怎么做，然后决定怎么踩刹车、油门，怎么打方向盘。

车辆：我们本身是不造车的，所以车辆是由 OEM 提供的，但是整个控制的算法，是我们自研去做的。

除此之外，还有高精地图与定位模块，以及数据与系统架构（数据的处理，以及控制数据在不同模块的流动）。

这里介绍的是各个模块，但最后把他们串联起来，靠的是我们的自动驾驶软件系统，这就是自动驾驶的车载系统。很多自动驾驶企业使用的是 ROS 的一套工业系统，而 Pony.ai 是从第一行代码开始，写了一套 PonyBrain，自研的多层次自动驾驶车载系统，最主要的做的事情有：

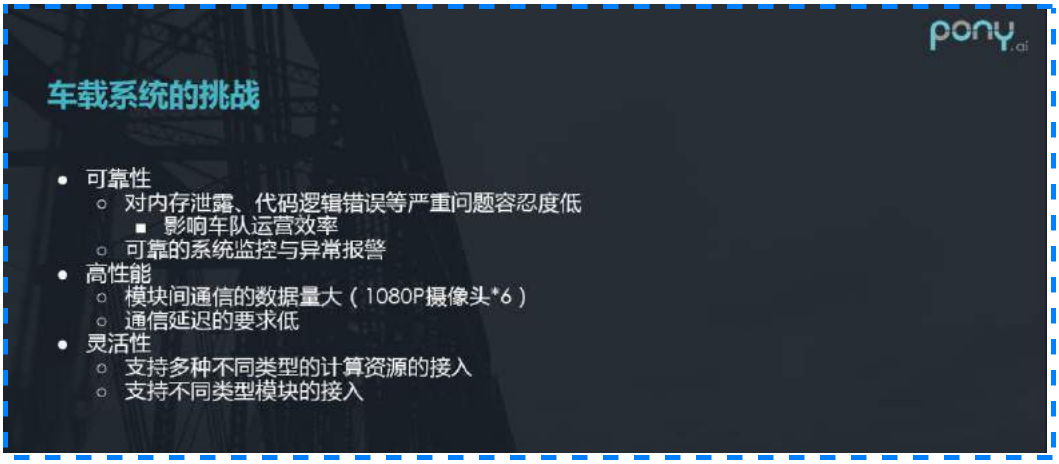
多模块的调度运行。所有模块的调度运行都是 Pony.ai 自己去做的。

模块间的消息通信。如何把数据从激光雷达传递到传感器融合的模块，再把融合的结果放到感知模块中，然后感知的数据怎么告诉行为预测、决策规划等等模块，以及如何拿到高精地图与定位的信息。

车载计算资源的分配与管理。对于自动驾驶来说反应速度是非常重要的，这就需要我们对内存、CPU、GPU 等有足够的优化，做到定制化的车载计算资源分配与管理。

日志记录。同时我们需要完善的日志记录，我们所有的测试数据回来都需要一整套的 Pipeline 去做自动化的分析，然后帮我们评判出有意义的数据，给到测试工程师或者研发工程师，进行进一步的分析去使用，然后进一步提升我们的模型。

监控与报警。保证了我们自动驾驶的安全性。



车载系统的挑战：

① **可靠性：**车载系统必须足够的可靠，不能有任何的内存泄露、代码逻辑的错位，这种都是零容忍的，一旦发生了这样的事情，对整个自动驾驶系统来说是非常严重的事故，是有可能影响到安全性的，对于 Pony.ai 自动驾驶系统技术的发展来说，安全永远是我们的第一位，所以所有影响安全性的事情，我们都是零容忍的，同时他也会影响车队运营的效率；所以我们还需要系统监控与异常报警，一旦系统出现任何问题，我们需要及时提醒安全员，做出车辆接管的操作。

② **高性能：**满足模块间通信的海量数据压力，同时实现低延迟。

③ **灵活性：**支持多种不同类型的计算资源的接入，以及不同类型模块的接入，需要有灵活的系统来支持计算资源的高速迭代。

车载系统的实践：

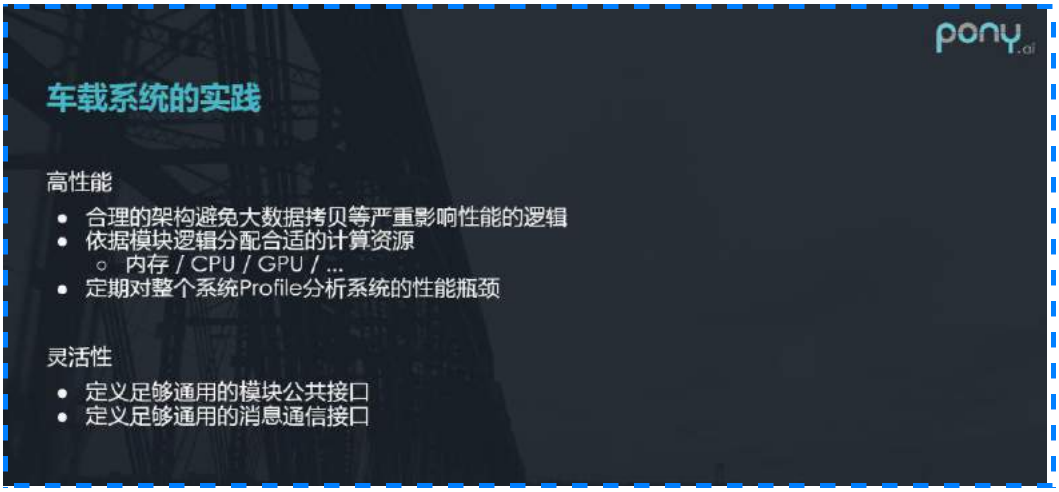
可靠性：

① 代码质量要求高：对于可靠性来说我们有非常严格的 code review 和 unit test，相信这是在国内互联网公司不太容易见到的一件事情，虽然会非常耗时，但是对可靠性的提升是有非常大的帮助的。

② 合理使用工具帮助发现问题：同时我们也会使用非常多的工具，如静态分析、ASAN 等等，来做离线的分析，来保证系统的可靠性。

③ 多重系统可靠性检查：包括系统启动前校验，系统运行时实时监控，系统运行后数据分析等。

④ 这是我们的持续集成与发布的平台：对于每一次代码的修改，我们都会进行仿真测试；然后对于研发的迭代，我们每周会有 Release 版本的更新，保障版本的稳定性，同时，刚刚我们整个测试包括封闭，半封闭，高峰期的测试，整个测试流程怎么持续集成与发布，也是保证系统可靠性的一种方法。



高性能：

① 合理的架构避免大数据拷贝等严重影响性能的逻辑。

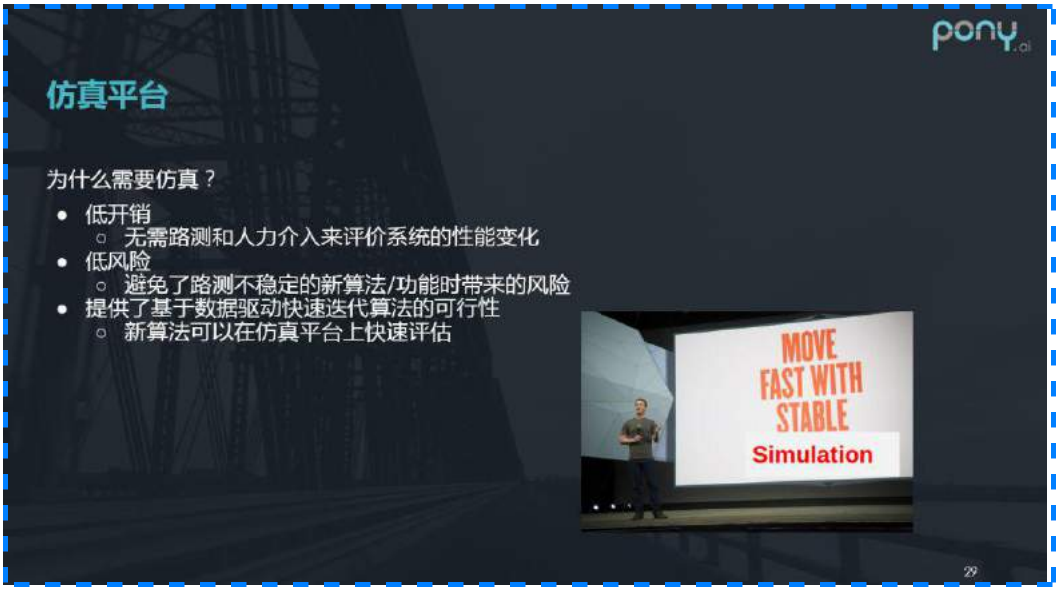
② 依据模块逻辑分配合适的计算资源，如内存、CPU、GPU 等。

③ 定期对整个系统 Profile 分析系统的性能瓶颈。

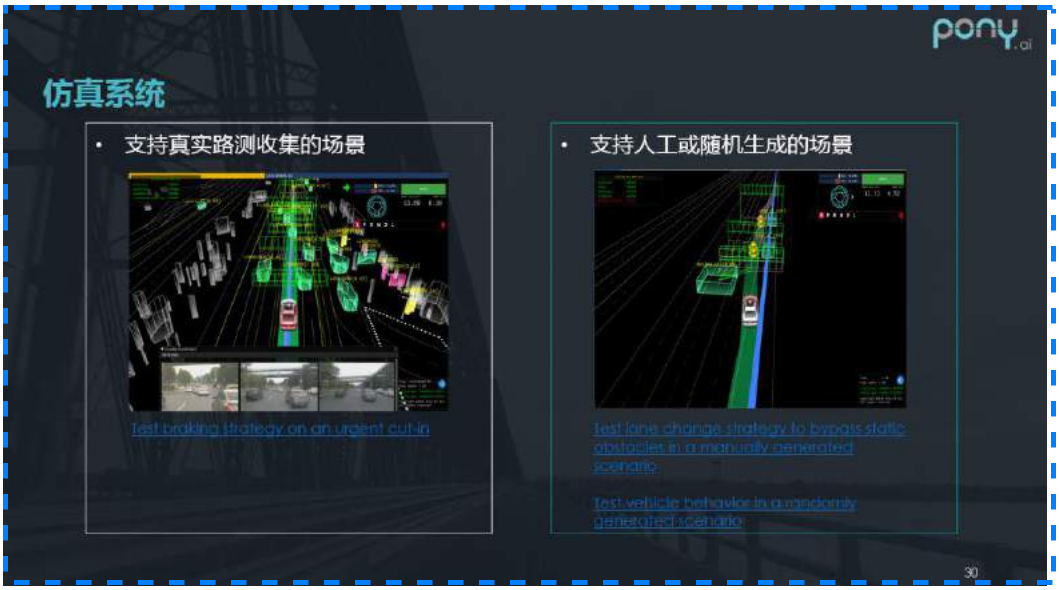
灵活性：

① 定义足够通用的模块公共接口。

② 定义足够通用的消息通信接口。



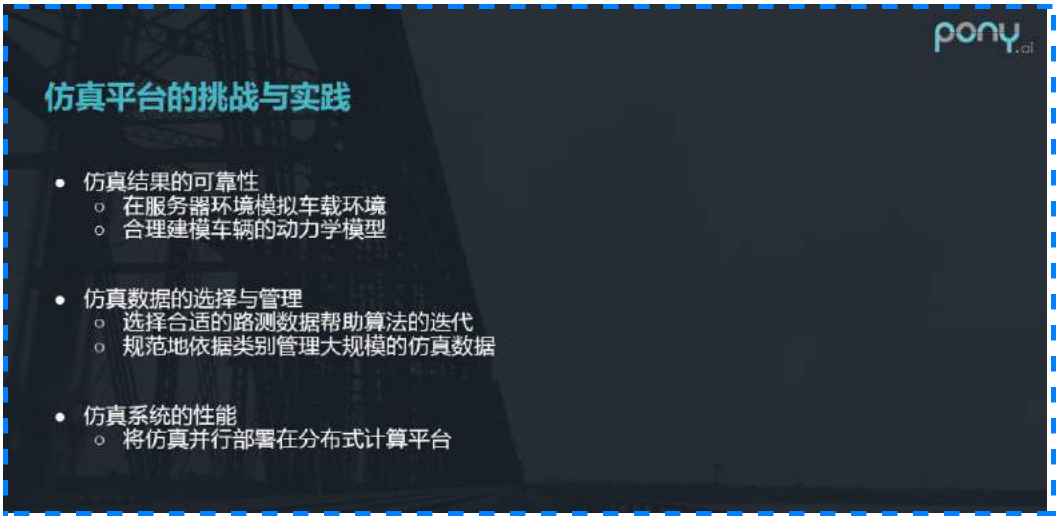
为什么需要仿真系统？因为仿真系统可以使得我们车还没有上路的时候，就已经做了大规模的自动驾驶测试，无需路测和人力接入就可以评价系统的性能变化；由于没有进行路测，不会引起路面事故；同时，仿真系统还提供了基于数据驱动快速迭代算法的可行性，新的算法可以先在仿真平台上做验证，一些具体的指标和测试的信息都会在仿真平台上有所体现。



仿真系统数据的俩个不同来源：

① 支持真实路测收集的场景，我们的路测数据非常的多，数据回来之后，通过 Data Pipeline 自动更新这些有意义和有意思的场景，我们会根据当时的场景改动相应的模块，然后会在仿真系统重跑当时的场景，来判断新的方法是否 work；

② 支持人工和随机生成的场景，这样的一些仿真的场景，也是非常的重要的，因为虽然我们在做大规模的路测，但是不代表可以遇到所有的场景，很多场景无法在路测中收集到，这就需要我们通过人工去创造这样的场景出来，给我们的系统一些样本，来学习如何处理这样的场景，保证我们新的 feature 在这样的场景不会出现问题。

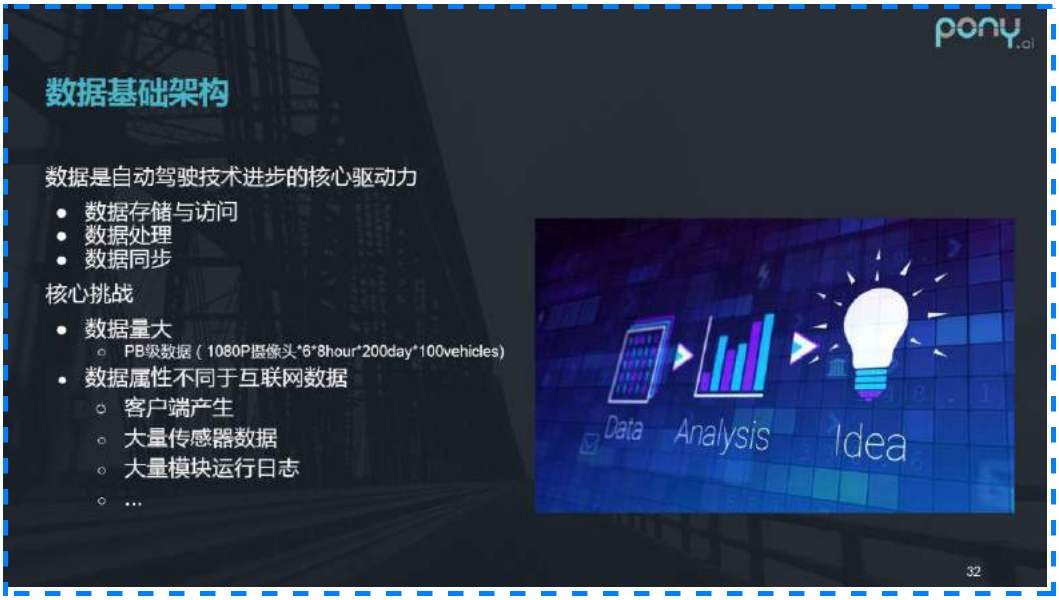


仿真平台的挑战与实践：

① 仿真结果的可靠性：首先仿真的结果必须是可靠的，如果不可靠，用它检测出来的结果是没有任何的意义的。整个仿真是在服务端模拟车载环境跑的，同时在服务端构建车辆动力学模型，保证测试的数据足够可靠。

② 仿真数据的选择与管理：当然我们会选择合适的路测数据来帮助算法的迭代（这里的选择不是人工的选择，是全自动化的选择，帮我们在茫茫数据中挑选出有意义的数据）；另外，我们还会规范的依据类别管理大规模的仿真数据，比如感知模块的一些改动，到底需要测试哪些数据，才会更加的体现这个改动带来多少影响，这里我们会有内部的一个分类，我们不会对所有的数据进行无差别的仿真（这样做意义不大）。

③ 仿真系统的性能：我们将整个仿真系统并行部署在分布式计算平台中，这才可能满足我们单天 30W 公里以上的仿真测试，并且这个数据还在不断增长。



数据基础架构：

数据是自动驾驶技术进步的核心驱动力，没有数据，我们就看不到现在如此多的测试车辆在进行路测，数据本身有几个重要的点：

① 如何存储海量的数据，如何支持快速的访问。

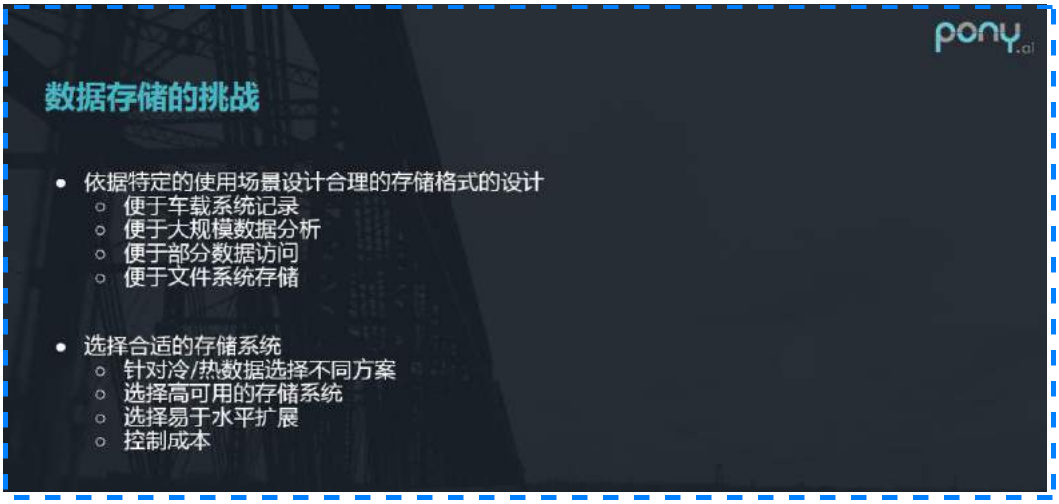
② 如何进行数据处理。

③ 如何进行数据同步，如何把不同区域、路测数据、车载数据同步到数据集中，如何让不同办公区的工程师都可以使用这些数据，对数据同步来说是一个很大的挑战。

核心挑战：

① 数据量大：我们有 PB 级别的数据，这里只是以摄像头为例，还包括其他传感器数据，以及系统运作的中间数据等等。

② 数据属性不同于互联网数据：我们的数据由客户端产生，有大量的传感器数据、大量的模块运行日志，这与互联网数据有本质的区别，所以对整个数据架构的要求也是不一样的。



数据存储的挑战：

① 依据特定的使用场景设计合理的存储格式的设计：以便于车载系统记录、大规模数据分析（数据回来之后，需要有方法进行分析，找出有意义的数据）、部分数据访问、文件系统存储（如何高效的利用文件系统）等。

② 选择合适的存储系统：

- 针对冷/热数据选择不同方案
- 选择高可用的存储系统
- 选择易于水平扩展，因为车辆规模是不停的在变大的，运营时间越来越长，数据的增长速度是远超想象的，所以需要易于水平扩展的存储系统。
- 控制成本，不能用过于昂贵的设备。

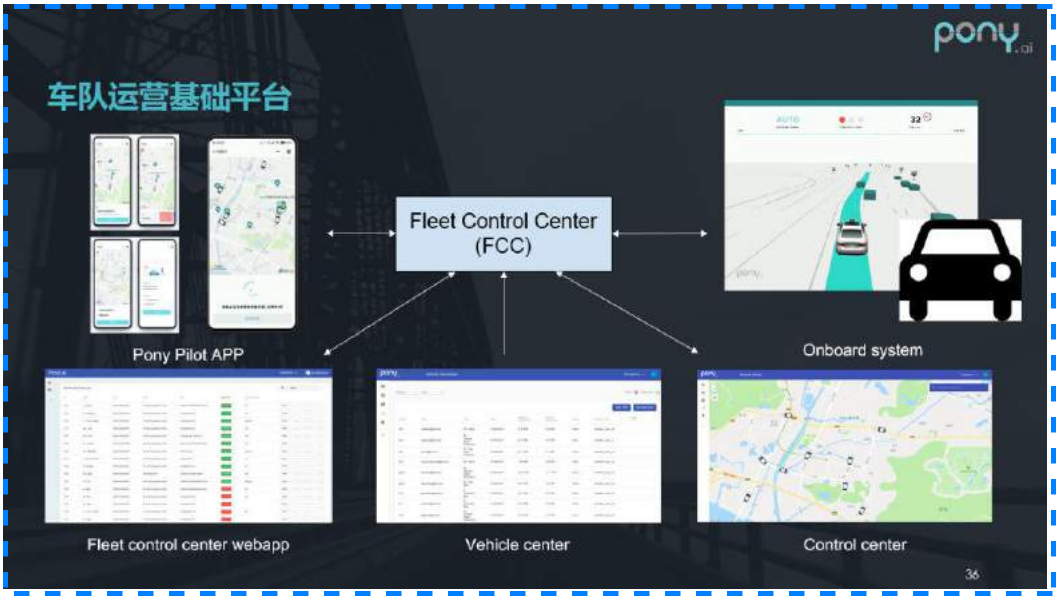
数据处理可以帮助收集性能指标，有 MPI（平均每次接管所需里程）、模块运行效率、乘客舒适度体验等，还有就是路测有趣场景的挖掘，如接管、急刹、感知算法识别、不合理的变道策略等用于模型训练和仿真。

数据处理的挑战：

① 减小数据采集到处理的全流程时间：如何以最快的速度把数据从车传到中间处理系统，Data Pipeline 运行完之后，上传到数据中心，这里面我们做了非常多的工作。

② 依据不同类型数据处理任务选择合适的处理系统：计算量要求比较高的我们选择 CPU 密集型系统来处理；更多的会是车载的数据，我们会选择 IO 密集型系统进行处理。

③ 通用的任务定义以支持灵活的添加新任务：帮我们检测出来更多有意义的数据。



车队运营基础平台：

我们有一个 Pony Pilot 项目，在我们广州所有的内部员工都可以使用，同时在北京和美国加州，也有同样的服务已经上线，那么支持这样的服务，我们需要做哪些事情：

- Fleet Control Center，车队控制中心
- Pony Pilot APP
- Onboard system

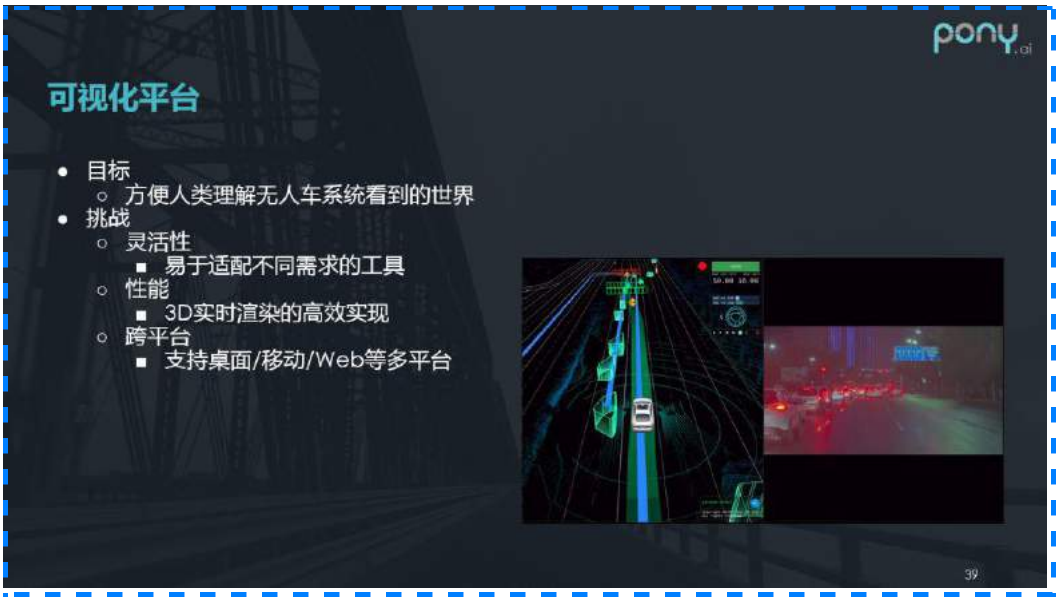
- 各种各样的 webapp，帮助我们观察整个车队的运营情况，帮助管理测试的车辆和人员。

车队运营基础平台的挑战：

需要支持复杂需求变化的 web 框架，同时我们有大量的 web service 的部署与管理，这都需要我们去完善 web 服务通用组件，例如部署工具、日志记录平台随时排查问题、监控平台保证所有 service 平台的高可容性。

容器与服务调度平台：

通过 Kubernetes 来帮我们做各种各样的服务调度和集群支持。



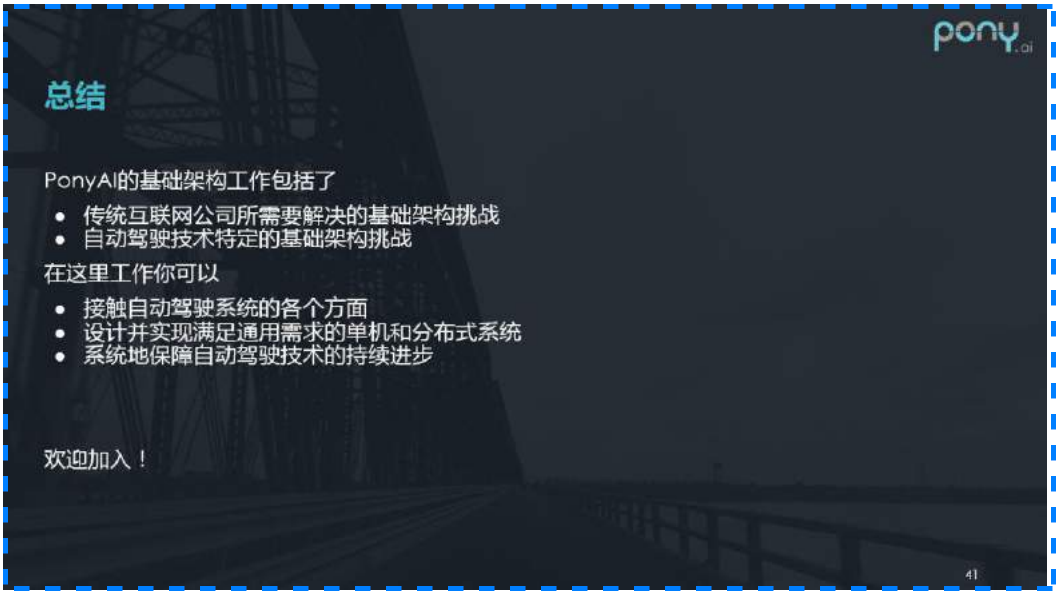
可视化平台：

① 目标：方便人类理解无人车系统看到的世界

② 挑战：首先，需要足够的灵活，易于适配不同需求的工具；其次，需要有高性能的现实，如 3D 实时渲染的高效实现；最后，支持跨平台的可视化框架，如桌面系统、移动系统、Web 等多平台。

人机接口：

方便乘客使用的用户界面，同时可以看到自动驾驶是如何了解世界，如何做决策，如何规划之后的行为等等，给乘客更多的信息和信任。



总结：

① Pony.ai 的基础架构工作包括：

- ❖ 传统互联网公司所需要解决的基础架构挑战。
- ❖ 自动驾驶技术特定的基础架构挑战。

② 在这里工作你可以：

- ❖ 接触自动驾驶系统的各个方面。
- ❖ 设计并实现满足通用需求的单机和分布式系统。
- ❖ 系统的保障自动驾驶技术的持续进步。

这是一个非常有意思的 team，里面有很多有意思的工作，非常欢迎大家与我们一起来工作，推动整个自动驾驶的发展，谢谢大家！

从硬件角度剖析自动驾驶，为什么说它是复杂的系统工程？

嘉宾：彭健　编辑：Hoh Xil

今天，我主要会向大家分享下自动驾驶在硬件技术方面的考虑，特别是面临的挑战和一些应对决策。

自动驾驶发展简述

首先，我给大家分享个小故事。

2004 年，DARPA 举办了第一届 “DARPA”挑战赛（无人车挑战赛）。根本没有一支参赛队伍完成这次比赛任务，这件事对来自斯坦福大学的 Sebastian Thrun 刺激很大。

于是，他在 2005 年组建了自己的无人车队，并且成为这项挑战赛上的第一支完赛队伍，获得奖金 200 万美元。这笔奖金虽然巨额，但事实上 Sebastian Thrun 在这个项目的投入远远大于 200 万美金。

看到上面的照片，人们的第一反应，或者说最引人瞩目的，就是车顶上的一排传感器，当时采用的激光雷达传感器品牌是 Sick（施克）。

这是 Sebastian Thrun 当时在无人车后备箱工作的一张图片，这些大大小小的设备体现了自动驾驶对于计算能力和信号处理的需求。

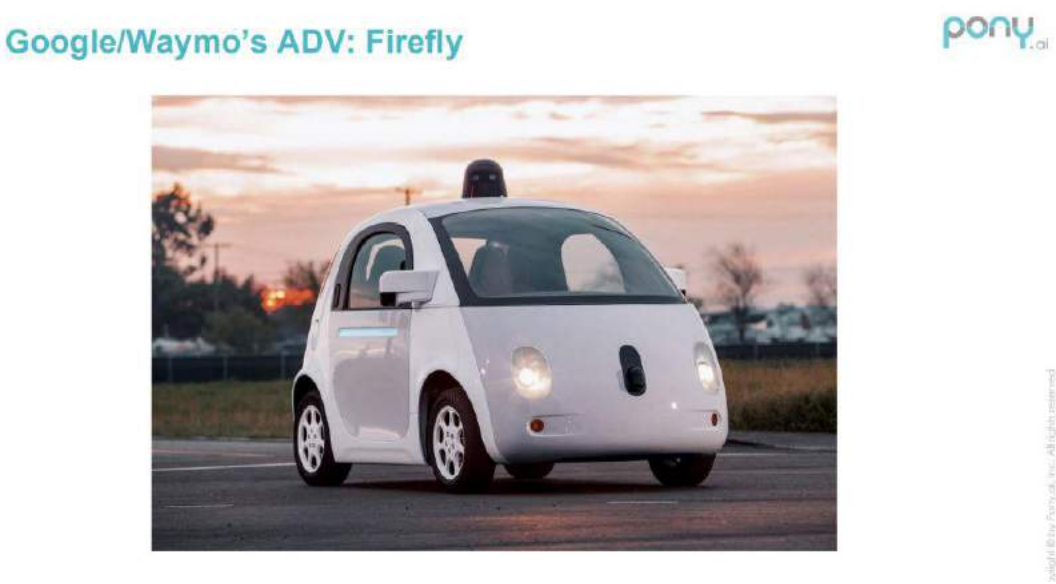
Sebastian 团队另外一大贡献就是在完成挑战赛后，发表了一篇著名的 paper（论文）《Stanley: The Robot that Won the DARPA Grand Challenge》，我相信几乎每位自动驾驶的从业者都看过这篇 paper。

2007 年，DARPA 举办了另一种路况的挑战赛——在城市的道路中完成自动驾驶。最后，这场比赛的冠军由来自卡内基梅隆大学的团队摘得，他们的无人车使用的激光雷达传感器变成了 Velodyne。

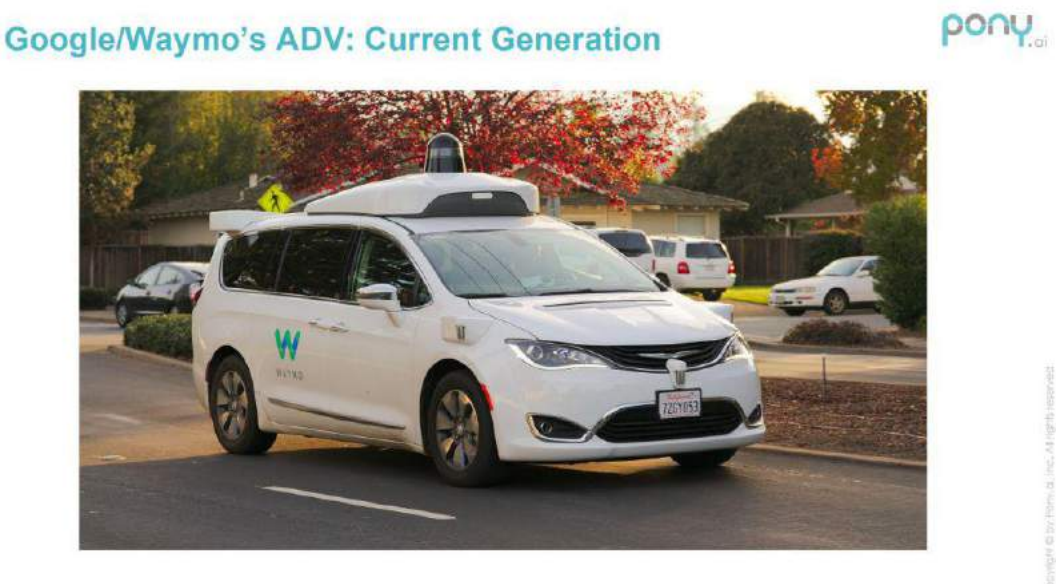
自动驾驶行业发展到今天，我们会发现世界做自动驾驶主流的人才很多都来自这些高校。

了解了自动驾驶简单背景，接下来，我们可以看看不同公司制造的自动驾驶汽车。

这是 Google（Waymo）早期的一辆自动驾驶车，它的顶部采用的传感器也是来自 Velodyne。

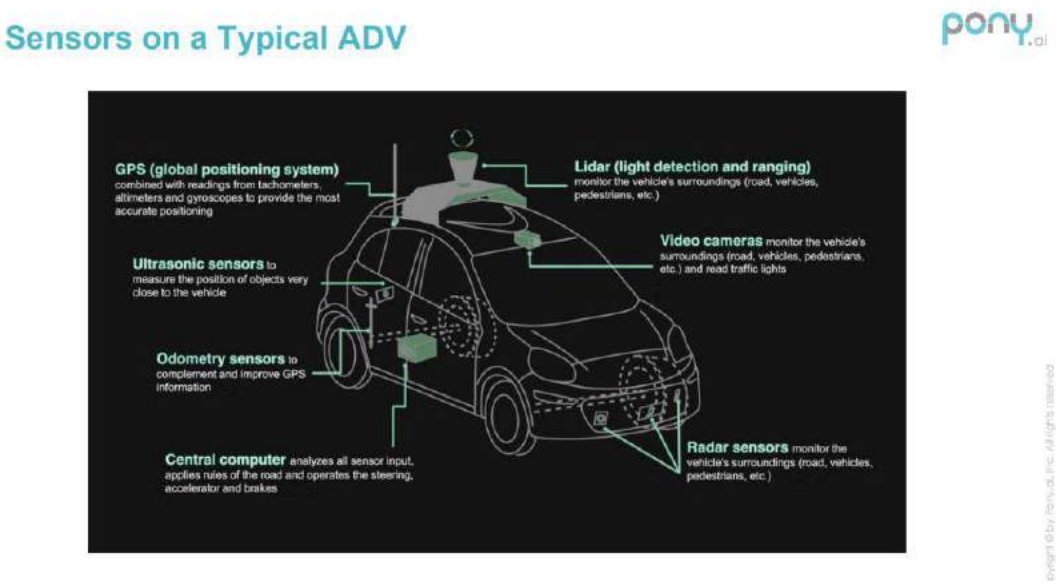


这是 Google 在 2014 至 2017 年间制造的一辆真正的“无人车”——Firefly（萤火虫），它没有驾驶舱和方向盘，但是由于当时实现真正“无人”的困难，所以 Google 后来就不使用了这款车型了。



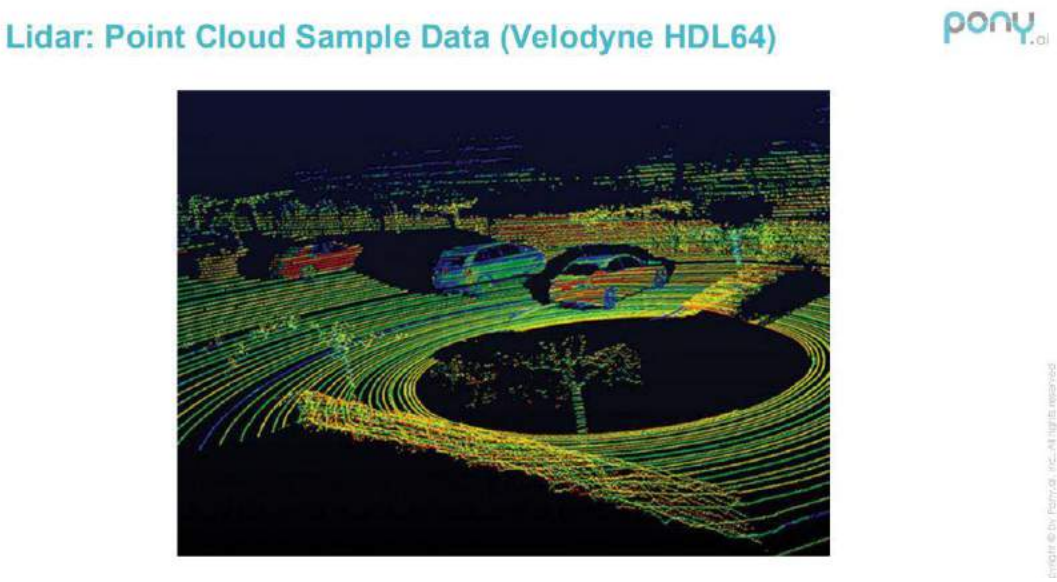
上图则是 Google 现在的自动驾驶车，他们的硬件技术思路是，硬件系统和软件系统都自主完成。其中硬件部分包含传感器、计算平台、信号方案、供电等等，都是 Google 自主定制的。

综合自动驾驶车外形变化历史来看，普通车和自动驾驶车最大的区别就是自动驾驶车上装载了各种各样的传感器。



一辆自动驾驶车辆(Autonomous Driving Vehicle, ADV) 一般都配有以下这些传感器：Lidar 激光雷达， Video Cameras 相机， GNSS/IMU 定位系统。

相机大家都见过，我就不介绍了。下面我简单说一下 Lidar 和 GNSS/IMU 这两种传感器。



激光雷达（Lidar）是做自动驾驶必备的传感器，上图展示了 Velodyne HDL64 扫描的结果。这是一款机械式激光雷达，总共会有 64 条扫描线，它具备测量精确距离的能力。

比较常用的 **GNSS 卫星导航定位系统**包括：GPS、BeiDou、Galileo、GLONASS，由于 GNSS 更新速率较慢，单独使用无法满足要求，所以还需要有 **IMU（惯性测量单元）**支持，它的更新速率较快。GNSS 和 IMU 结合，可以满足自动驾驶要求。

Computing Platforms

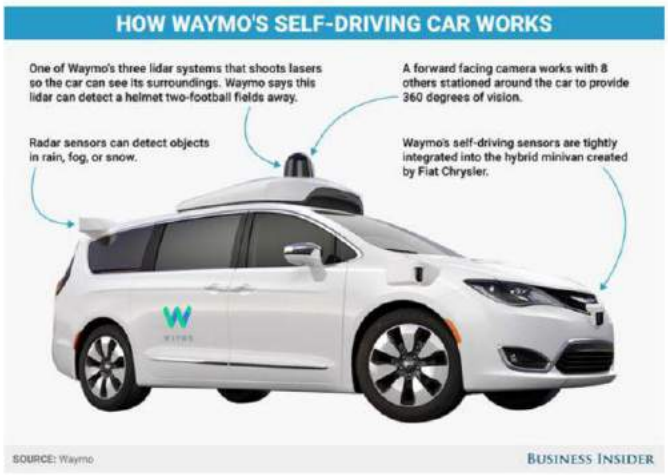
Computing platforms on autonomous cars:

- Varies greatly from different companies
- Heavy computation
  - Demanding power and thermal requirements
- Withstand vibration and heat (auto grade)

至于计算平台，根据计算量的不同，计算平台的复杂程度也各不相同。另外，越复杂的计算平台耗电量越大，所以也需要安装散热系统。

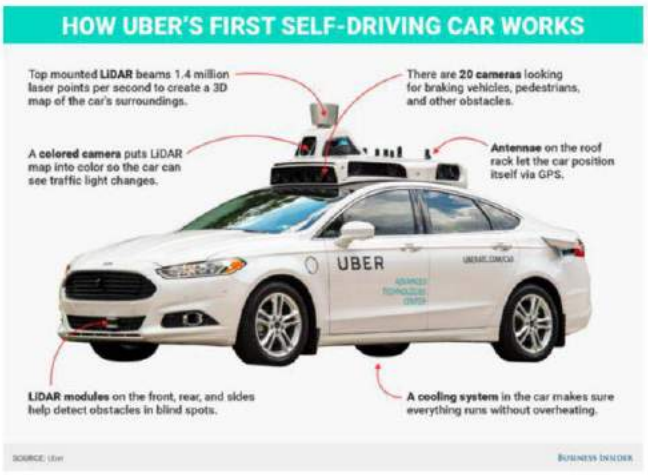
下面两幅图，分别展示了 Google 和 Uber 自动驾驶车辆所采用的传感器等硬件设备，包含激光雷达、相机、毫米波雷达、散热系统等。

Google/Waymo's ADV: Current Generation



Uber's ADV

pony.ai



自动驾驶硬件技术上的挑战

正如前面所说，一辆实现 L4 级自动驾驶功能的无人车无法离不开硬件技术的支持。也因此，自动驾驶是一项系统工程，除了出色的软件系统，也需要关注硬件设备在机械方面和电子方面的要求和挑战。

机械方面

首先是自动驾驶车的**“颜值”**，这是人们在视觉上最直接的反馈。如果一辆车的外观不够吸引人和赏心悦目，它在市场上就可能缺乏竞争力。

其次，传感器的增加需要满足**尺寸和重量的限制**。例如，绝大多数地下停车场限高 2.2m，这就要求车顶加装雷达后的车辆高度最好不要超过 2.2 米。绝大多数车顶承重不能超过 50-60 公斤，以及后备箱空间也有限制等等。

第三，**温度范围**也有要求。因为车内温度的变化比较大，所以对车载电子设备的温度适应能力要求比较高。在仪表盘上使用的设备温度要求范围是-40 摄氏度到+90 摄氏度，而在发动机舱内的要求是-40 摄氏度到+125 摄氏度。

第四，**热量管理**。这既包括散热问题，也要考虑严寒天气的适应能力和另外加热的问题。

第五，**冲击和振动**。车辆在行驶的过程中，免不了会产生的或弱或强的冲击和振动，这也成为硬件技术需要解决的任务。

最后一点，**防水与环境防护**。自动驾驶对精度和敏感度的要求极高，这也要求硬件团队能够针对各种类型天气和环境采取相应的解决方案，提高应对能力，以达到安全和稳定性的要求。

这些天气和环境包含：

- 雨雪、冰雹天气
- 大风、雷暴天气
- 霜、雾
- 雾霾（激光雷达对 PM10，即 10 微米以下的颗粒物非常敏感。）
- 湿度、露水
- 灰尘
- 紫外线
- 高原（海拔越高，空气越稀薄，硬件散热能力随之下降。）
- 盐的腐蚀性（例如海滨城市内吹拂的海风中往往含有盐份，并具有腐蚀性。）

电子方面

自动驾驶采用的硬件设备在电子方面也有许多注意事项和要求。

首先，**温度条件**。基于硬件设备类型，一般可以分为五种温度要求：

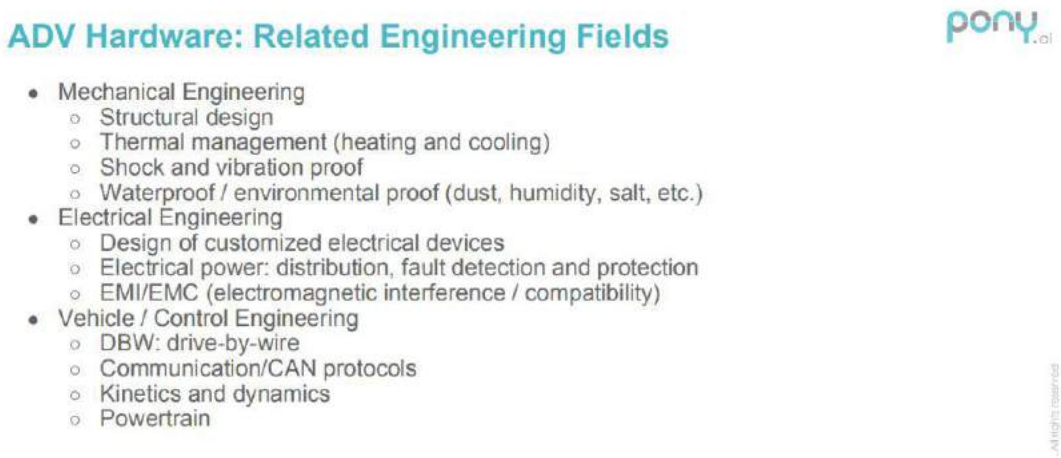
- 商品级：一般要求是 0℃~85℃
- 工业级：-40℃~100℃
- 汽车级：-40℃~125℃
- 前舱级：-40℃~125℃
- 军用级：-55℃~125℃

其次，**应对恶劣的环境**，包括冲击、振动、防水、自然环境等。

第三，**供电能力有限的挑战**。绝大部分车辆的供电功率少于 1KW，好一点的车型可以做到 1.5KW，除非像 Waymo 一样自己改造供电，可以做到 3KW 供电能力。

最后，**恶劣的电磁环境的挑战**。这种环境下，自动驾驶的硬件技术需要做好抗干扰。

自动驾驶相关工程领域



前面也提到过，自动驾驶是一项系统工程。除了与感知、地图、定位、规划与控制有关的软件模块，了解与自动驾驶有关的工程领域，能更好帮助从业者认识这个行业。

第一个领域自然是机械工程，涉及以下几个重要领域：

结构设计

热管理，即把产生的热量排出去

冲击和振动防护

防水/环境防护（灰尘、湿度、盐等）

第二个相关领域是电气工程，包括：

设计电子器件和电路

电力供应：配电、故障检测和保护

EMI/EMC（电磁干扰/兼容性）

第三是车辆工程与控制工程领域，一般涉及：

DBW（Drive By Wire）：线控驱动

通信/CAN（控制器局域网总线）协议

Vehicle Kinetics and Dynamics（车辆运动学和动力学）

动力系统

04

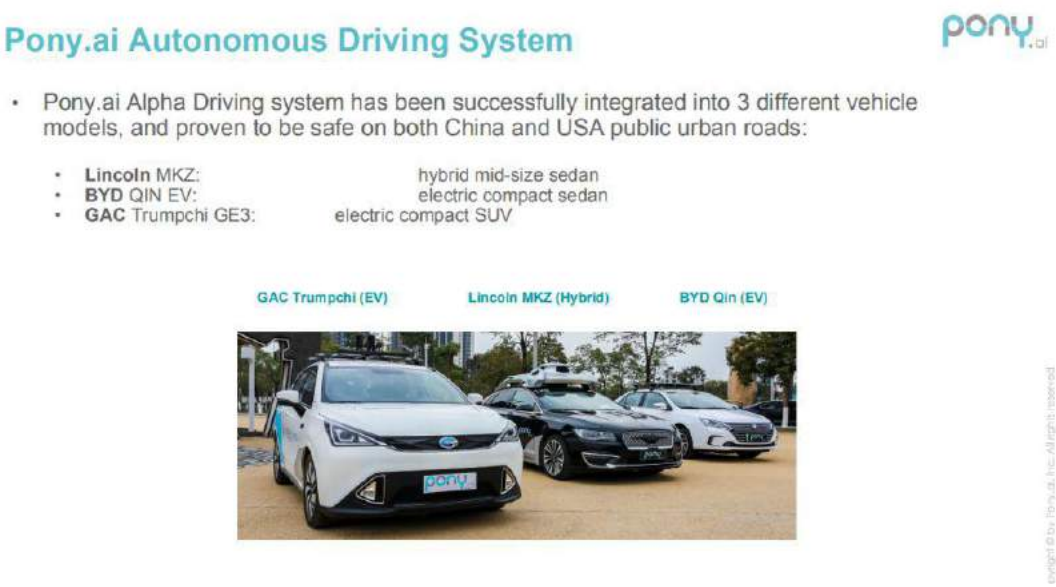
小马智行自动驾驶系统 PonyAlpha

了解了整个自动驾驶硬件技术的背景和挑战，这一部分我想为大家介绍小马智行研发的自动驾驶系统和硬件平台。

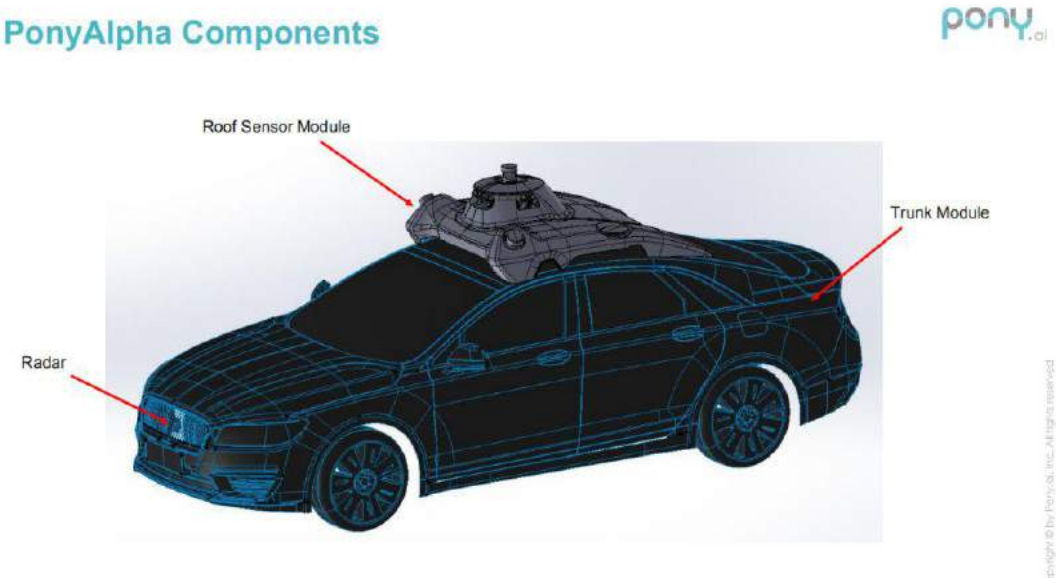
目前，小马智行已经迭代了三代自动驾驶硬件平台。

2017 年 4 月，小马智行打造了第一台自动驾驶测试车（上图），第一代硬件平台采用的传感器包括了 Velodyne HDL64 的激光雷达。照片里站在车旁的分别为小马智行两位联合创始人 CEO 彭军（右）和 CTO 楼天城（左）。

第二代平台采用了 32 线激光雷达，传感器的体积明显缩小。第三代方案则是去年 9 月在世界人工智能大会上发布的 PonyAlpha。



在近三年的时间里（编者注：该场技术演讲发生在 7 月 6 日），Pony 已经在三种不同的车型上搭载小马智行自动驾驶软硬件平台，分别包括上图左侧的广汽传祺电动车、中间林肯 MKZ、右侧的比亚迪秦（编者注：今年 8、9、10 月，小马智行分别宣布在丰田雷克萨斯 RX、广汽 Aion LX 和现代 Kona SUV 搭载了小马智行自动驾驶软硬件方案）。



PonyAlpha 硬件平台有三个组成部分：车顶模块，后备箱模块和毫米波雷达。

毫米波雷达部分相对比较简单，这里就不介绍了。先说第一个组成部分：车顶模块。车顶安装了许多传感器，主要负责解决自动驾驶的感知问题。我们要求传感器（感知系统）的覆盖范围为 360°全景，并且长距离（200m），因此采用了 6 个相机，包括广角和中长焦相机。另外还有 3 个激光雷达（探测距离在 100m~200m）。

从左前方向的视角观察硬件平台：顶部的柱体是 32 线主激光雷达，16 线激光雷达放在侧面，正前方采用三个摄像头，左侧面安装了一个摄像头。

这是左后方的视角：安装了 GPS 天线，有朝后看的摄像头，也可以看到 32 线主激光雷达。

整体来看，Ponyalpha 车顶模块包括：

◆ 车顶行李架结构，安装在自动驾驶车辆上；
◆ 传感器，包括 3 个激光雷达、6 个摄像头和 GNSS/IMU；
◆ 线束；
◆ 车顶外壳起到了装饰罩的作用，也改善了空气动力学性能，并为传感器提供保护。

硬件平台第二组成部分则是后备箱模块。后备箱模块一般包含以下子功能：

第一是车辆集成硬件，包括 PDS（Power Distribution System 配电系统）和直流/直流转换器。

第二是计算堆栈，包括硬盘、计算单元（CPU、GPU 和 FPGA）和传感器接口。

第三是通信设备，包括 4G/5G 路由器、网络交换机、以及和车辆通信的接口。

结论

关于自动驾驶硬件技术的分享到了这里，我最后做一个总结。

自动驾驶技术研发确会面临许多硬件挑战，从机械类（结构、热、冲击/振动、环境）到电气（温度范围宽、恶劣环境、功率有限）再到安全第一的目标。

也因此这个行业需要许多硬件工程师投入其中，解决这些挑战，比如机械工程师，电气工程师，控制工程师，嵌入式软件工程师，传感器工程师（ISP、摄像头、激光雷达、雷达、GNSS/IMU）等。

发展至今，小马智行的硬件技术进步飞速，我们也会为实现更多功能、更好的性能、更紧密的集成而不懈努力。谢谢大家！

大数据篇

Sophon ：Hulu 智能 OLAP 缓存层技术实践

嘉宾：罗安宁　编辑：董亮亮

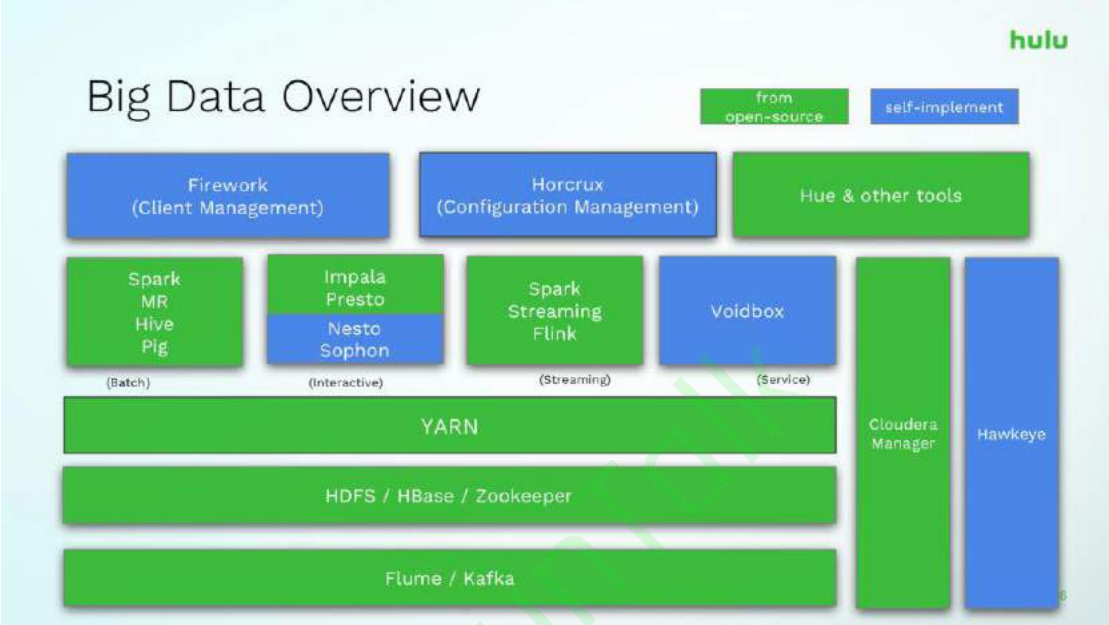
首先介绍下 Hulu，Hulu 是美国本土的互联网专业视频服务平台，主要以电影电视剧数字提供商提供的视频为主，同时有自制电视剧，还有实时的电视直播，Hulu 在 2018 年的订阅用户数达到了 2500 万，增长 48%。

本次演讲内容主要分为四部分：第一部分介绍 Hulu 的大数据架构；第二部分介绍 Sophon，Sophon 是一个轻量的 OLAP 中间缓存管理层，它的下层主要依赖于 OLAP 计算引擎—Impala；第三部分介绍 Impala 在 Hulu 的一些应用实践；最后是一个全文的总结。

1. Hulu 的大数据架构

Hulu 大数据架构分成 4 部分，架构最底层有两个自建的数据中心，它主要用于公司内部的大数据平台，包括大数据的存储，离线和实时计算，以及 OLAP 查询引擎等。同时

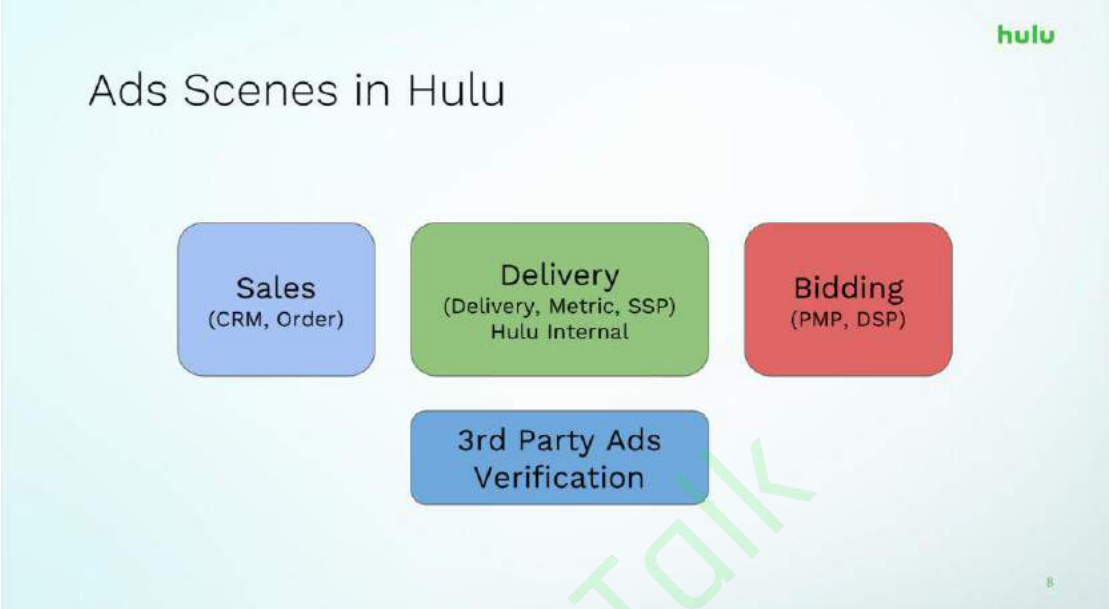
也有一部分的 data 和 data mart 存储在 AWS 的 S3 上，使用一些 Cloud Native 的查询引擎，例如 snowflake 等；基础设施层的上层是一些数据团队，包括广告，Metrics 团队等，最上层提供一些广告售卖和一些报表服务等。



Hulu 大数据的基础技术架构如上图，整体上是比较拥抱开源的，数据采集使用的是 Flume、Kafka，数据存储用的是 HDFS 和 HBase，资源调度用的是 yarn，离线计算用的是 Spark、MapReduce、Hive 等，在线计算主要用 Impala、Presto 以及一些自建的引擎，Presto 是一个 FaceBook 开源的 MPP 架构 OLAP 查询引擎，在 Hulu 主要用于 Ad Hoc 的查询，Impala 是 Cloudera 开源的 MPP 架构 OLAP 查询引擎，主要服务于一些自建的数据仓库，我们自研的主要是 Nesto 和 Sophon，Nesto 是一个能够查询嵌套数据结构，基于 MPP 架构的查询引擎，Sophon 是一个 OLAP 缓存的管理引擎，同时提供查询和路由的功能，再往上就是统一的客户端管理，包含统一的配置管理，自动更新等。我们的监控，除了采用 Cloudera 的自带监控外，还有自研的监控组件

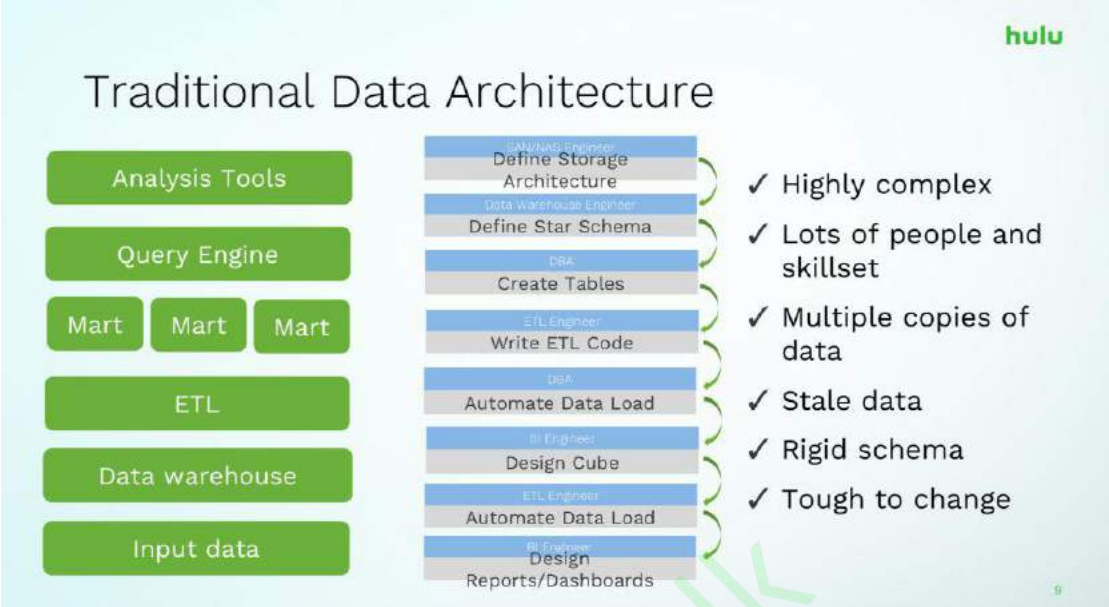
Hawkeye，主要是对 Cloudera 监控的补充，它可以用来统计整个 HDFS 和各种计算引擎的指标。

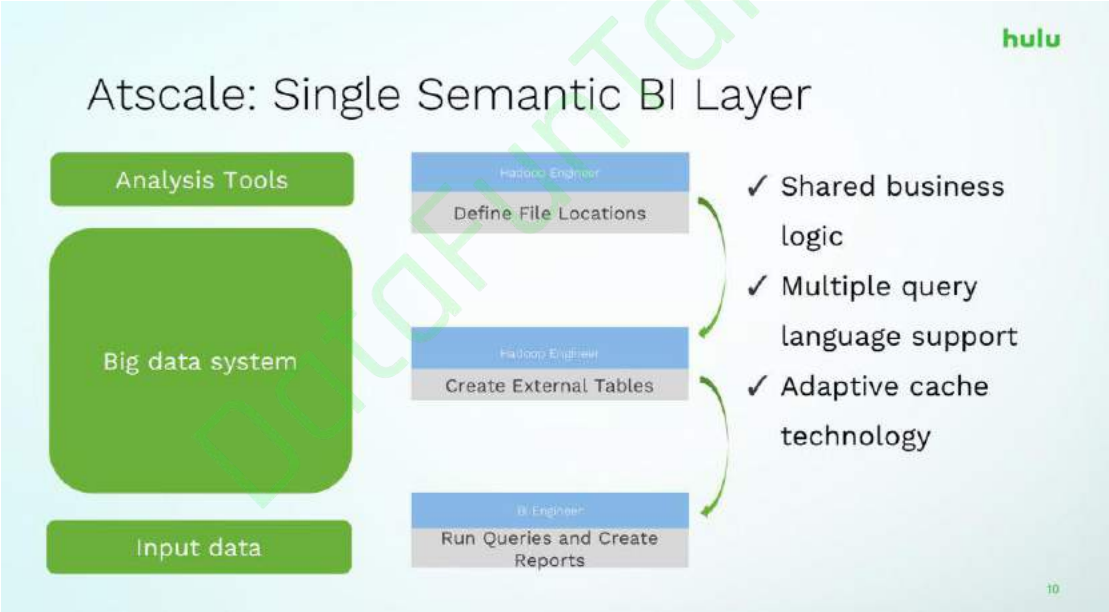
2. Sophon



在介绍了整个 Hulu 的大数据平台架构以后，我们来具体看一下今天要演讲的 Sophon。首先介绍下 Sophon 的研发背景，Hulu 广告的数据集市比较复杂。Hulu 主要以一些质量比较高的电视剧为主，所以它的广告是受到很大限制的，主要以展示类广告为主，这种广告一般是提前售卖的方式，它不是一种实时竞价的方式，提前售卖意味着今年 10 月份可能去谈明年一年广告售卖的单，这个单会非常非常大，在明年一年的投放中会细分成小的订单，它会整合到广告售卖订单系统中，售卖周期会很长，预先售卖的方式不可能把广告售卖的十分完整，总会有一部分广告位或多或少，是通过实时竞价的方式售卖出去的。因此整个系统会产生很多的事实表，包括广告的投放，广告的交易等。这些事件都会形成很多 fact 事实表，以及非常多的 dimension，包括订单的具体信息，

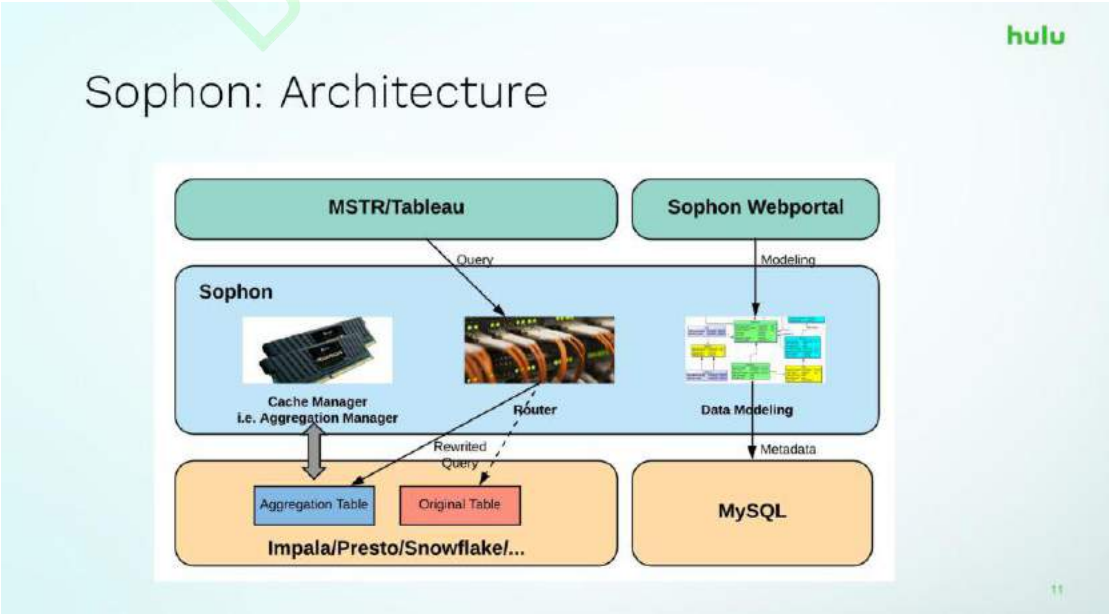
用户的具体信息，年龄，所在的州等，这样就会形成很多的事实表和维度表在你的数据集市 Data Market 中，如果用一个传统的方式对上层提供服务的话，那么你是需要一个很长的 pipeline 来处理这些数据。



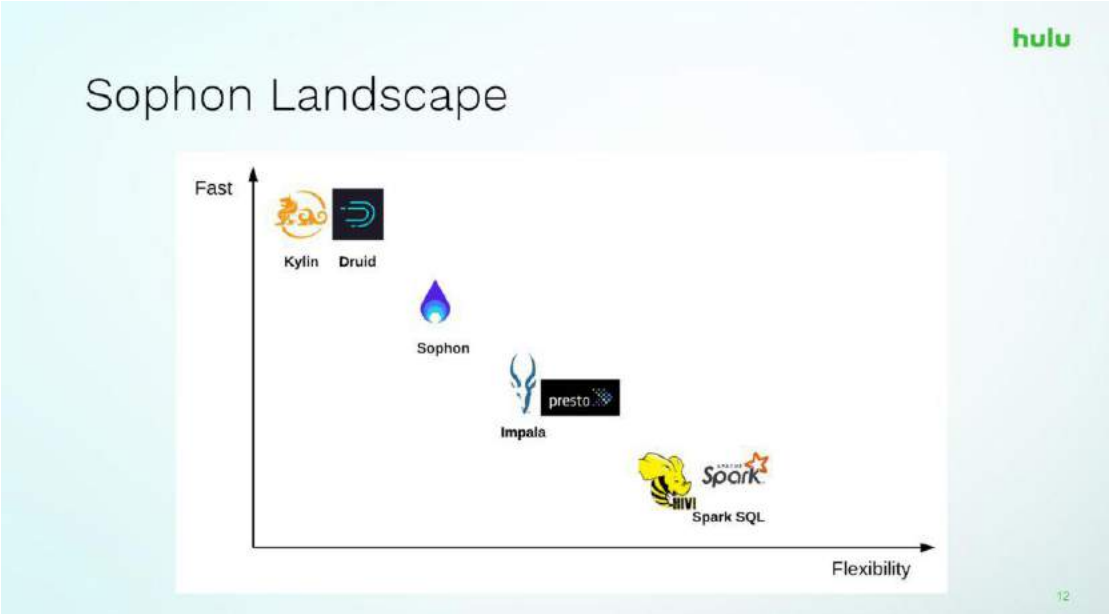


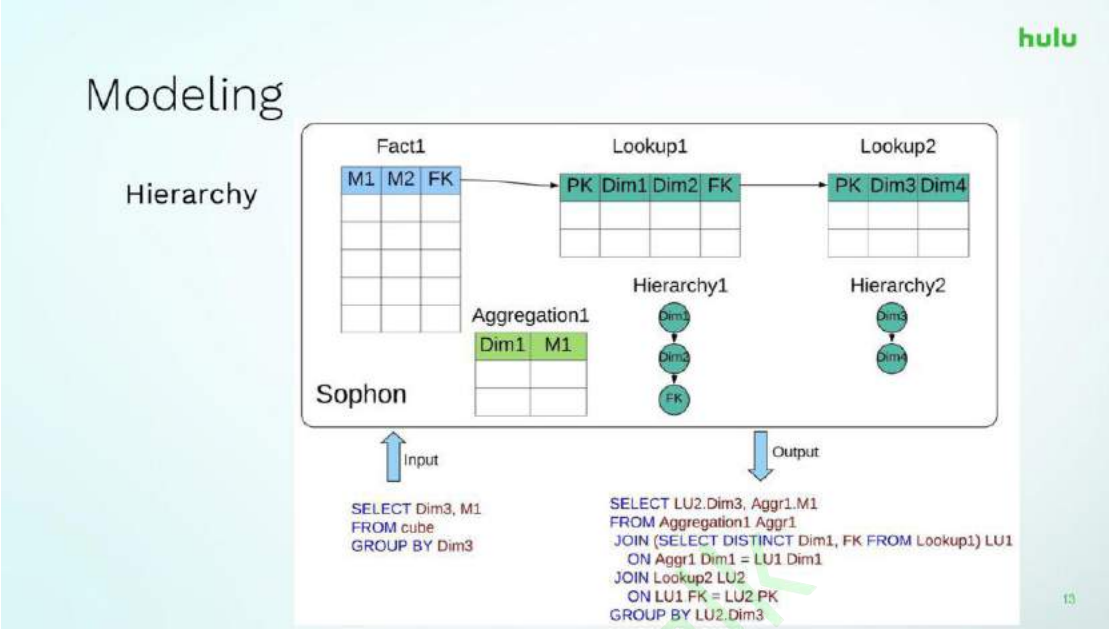
同时对于不同的事实表和不同的领域，需要建立不同的数据集市，有时比如你关心广告的售卖情况和具体的投放的关系时，就需要用相关的事实表和维度表来形成特定的数据

集市，不同的数据集市之间可能复用了部分相同的表，导致它们之间存在一定程度的耦合关系，这样当维度表的关系一旦发生修改的话，就意味着你上层的在不同 pipeline 的逻辑都需要修改，整个处理流程非常长。还有就是需要由不同领域技术背景的人去服务你的各个层次，因此我们希望这个过程可以更加一致，更加统一，更加灵活。再有就是来自 bussiness 的需求，比如当数据分析师用到一些新的维度组合时，你原有的数据集市可能满足不了，那么当你去重新建立一些 pipeline 去满足这个新的需求时，时间周期可能会非常久。所以我们考虑能不能把这些东西统一起来，把所有的数据关系集中到一起，当你有新的需求出现时，我能利用现有的模型和数据，在一个适当的响应时间内返回结果，然后系统能自动的优化 cache，加速查询。在这个方面我们尝试做了改进，把由下到上的过程统一到一起，把它结合到一个过程，就省掉中间各种流程。因为本质上那些数据之间的关系不随数据集市的不同而改变，既然不变的话，就能够构建一套系统在知道这套关系以后能自动的根据上层的需要去生成对应的数据集市需要的数据，直接去服务于用户，因此这就是我们尝试做 Sophon 的初衷。



从 Sophon 整个的框架可以看到，它上层对接的是 Bussiness UI 工具，比如 MSTR 和 Tableau，而对于数据建模人员来说，它提供一个界面去编辑和配置数据之间的关系是什么。它只去管理和重写 SQL，不会真正去做任何的查询,这样可以保证它是一个非常轻量的引擎。真正的查询工作它会下推给 Impala 或者其它一些数据引擎去实现。那么有一个问题，当我的数据表很原始的时候，我把很大的 fact 表接过来，查询速度会很慢，需要对整个过程进行加速，因此我们做了许多预聚合运算，存储了许多预聚合表，Sophon 并不会把这些预聚合的表缓存到内部，而是直接存储为 Hive 表，且记录所有表之间的 join 关系。这一点和 Kylin 不同，Kylin 将预聚合记过存储在 HBase 中，本身并不记录预聚合表与其他维度表之间是否能够 join。虽然 Kylin 能够配置维度之间的函数关系，但是只能适用于有限的几种情况。另外，Kylin 只支持 Snowflake 这种单 fact 的模型，在这种情况下，它可以把速度提升到很快毫秒级别，但牺牲了它的灵活性。因此，Kylin 和 Druid 更倾向于把我的查询提升到非常快的程度，达到毫秒级别，一旦没有命中的话 ,就会发送给后端的 Hive 引擎 ,这个速度就会有很明显的下降。对于 Sophon 来说，如果没有命中聚合表，但是 select 的维度和聚合表存在 join 关系，Sophon 会把两者 join 在一起做联合查询。一般来说 ,聚合表的规模远远小于 Fact 表 ,当这个 join 下推到一个 MPP 架构的查询，它的优化还是很可观的，即使我们没有在完全命中缓存的情况下，一个十几分钟的查询在这种情况下会缩减到几秒到十几毫秒，也可以非常明显的被上层感知到，所以它的适用范围会比这些查询引擎广一点，但是它的缺点就是会比 Kylin 这些引擎慢一点，到不了毫秒级的查询，因此使用场景会有一些区别。





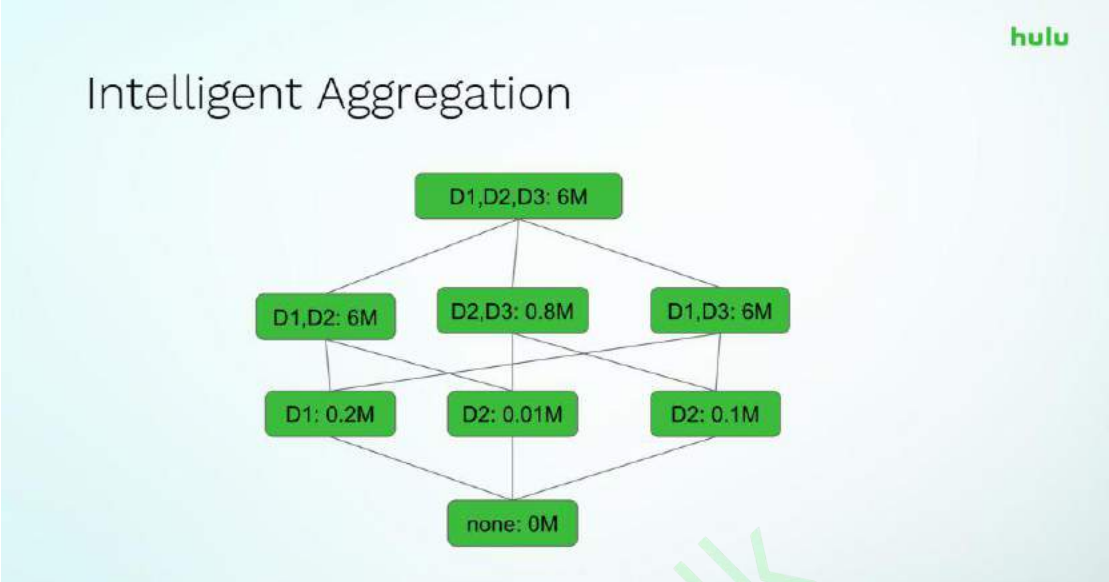
上面介绍了 Sophon 是做什么的，下面我们来介绍一下 Sophon 建模的一些细节。

Sophon 最主要的功能就是表与表之间的 Join，下面针对 Join 给出一个例子，如上图所示，我们有一个 Fact 表和几个 Lookup 表 join 到一起，在 lookup 表中，我们配置了维度与维度之间的层次结构关系，类似传统数据库的函数依赖，比如 Dim1 是一个 hour，Dim2 是一个 Month，你的 FK 可能是一个 year，同时系统中预构建了一个聚

合表，它只命中了维度 Dim1 和度量 M1，然而，查询语句并没有完全命中聚合表，那么在一个 Kylin 或者其它的系统中就直接放到后端的查询引擎计算了，但是对于 Sophon 来说，可以把 Lookup1 和 Lookup2 结合起来，把这两个表 join 到一起，这样聚合表就可以在这种情况下使用，这样的另一个好处是，当 Lookup2 的 Schema 结构发生了变化，预聚合 Aggregation1 同样是完完全全可以用的，不需要去做任何的重建工作。当然 Kylin 也支持 Derived Column 的特性，但是有很多的限制，像这样的例子无法适用，由于 Sophon 所记录的原始表之间的关系中，join 链往往比较长，很多情况都无法通过 Derived Column 这种方式来优化。

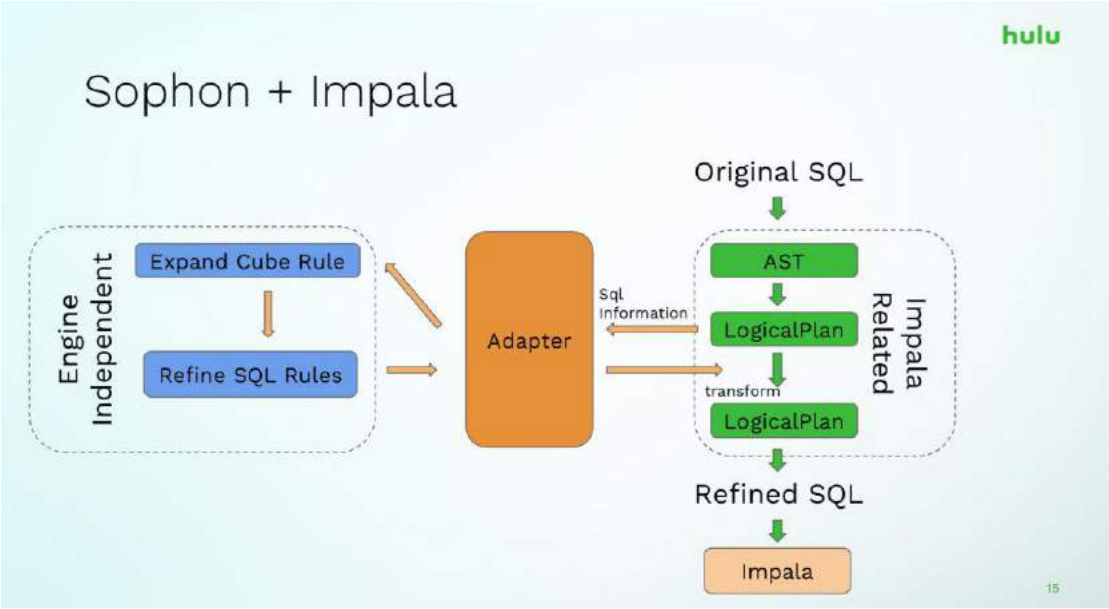
另外一个功能是，我们希望聚合可以自动构建。聚合本身就是一个计算 Cube 的过程，在 Kylin 里面，决定聚合哪些维度组合，需要人工探索，尤其是当 Dimension 维度规模比较大的时候，Cube 膨胀的规模会变得非常大，但是 Sophon 所遇到的场景本身维度就非常多，当你想通过人工去找出里面的规律，会非常困难。在 Sophon 里面，我们通过使用一个贪心算法去决定去建哪些聚合，具体来说就是每当我们选择一个节点物化一个维度组合生成缓存的时候，它对查询 Sophon 的所有 query 都会产生一定程度的优化（没有被优化的认为优化程度为 0），总体的查询代价就会减少，因此只要我们建立一个模型来计算查询的代价降低了多少，并且每一步选择一个代价减少最多的节点计算，就可以逐步通过这种贪心的方式得到一个比较优的结果。这种方法已经被证明与最优解之间的差距在一个有限范围内。综上可以看出，如何选择代价是一个关键的问题，直观上讲，代价应该和 join 的规模成正相关，当你 join 的两张表越大，代价就越大；如果聚合表的规模很大，那么就会耗费更多的存储空间，查询优化的效果也不经济，这种

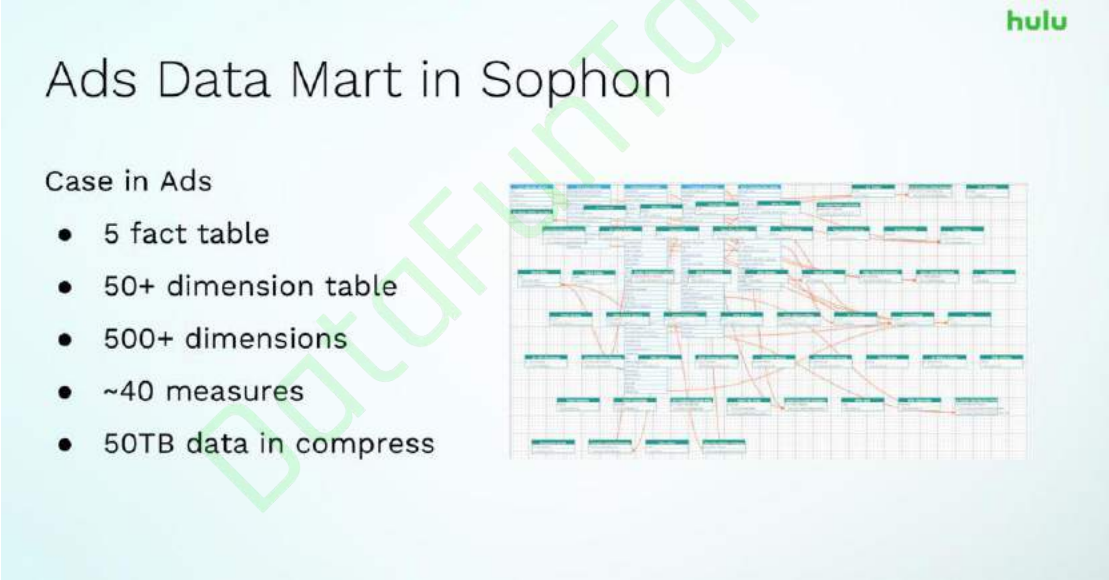
情况需要引入额外的惩罚。最后的问题是，这个算法本身的复杂度跟 cube 的复杂度是成线性关系的，假如说有 n 个维度，那么复杂度将是 O(2^n)。为了减少这个规模，我们一般需要去做一个聚类，来确定线上哪些维度是经常在一起查询的，然后再根据这样一个贪心的算法，去进一步的找出我们物化哪些节点，使得最终的代价最小，收益会最大。



最后是如何将 Sophon 和后端引擎融合到一起，这方面最重要的是将如何根据预先定义的模型选择聚合表和 join 路径与如何对输入的查询进行改写和优化分离开来。可以从下图看到，首先系统将输入的原始 SQL 翻译成逻辑执行计划，然后从逻辑执行计划中提取具体是要查询哪个 Cube，哪些 dimension 和 Measure 信息。然后将这些信息传递到右侧的优化系统中，这一部分与 SQL 采用什么样的语法没有任何关系。之后，系统根据这个信息去判断应该怎么去展开去命中需要的缓存，是否能命中聚合表，根据不同的选择去决定怎么把这个表展开，展开之后再去决定在这种场景下有哪些优化规则可以用的上，然后通过一个适配器生成改写 SQL 的规则，之所以这样做是希望 Sophon

可以不仅支持单一查询引擎，尤其是在当前背景下，希望以后可以支持一些 Cloud native 的引擎，比如 S3 上的 Snowflake 引擎，目前的趋势就是，以后的一个查询，一部分 1 年以内的数据可能在 In-house 的集群里，一年以前的数据在云上，我希望可以自动将这两部分数据的查询结果结合起来 ,这个时候可以由 Sophon 对这两部分数据进行管理和 Merge 结合起来，希望上层看到的是一个统一的视图，具体的底层细节数据是在云上还是在云下对于上层用户是无感知的。





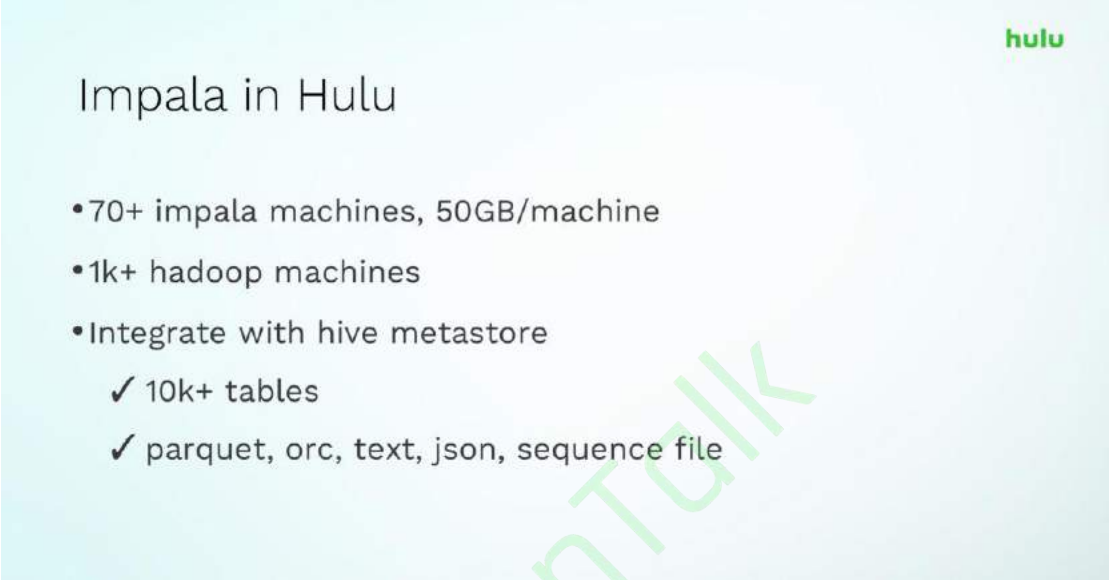
下面介绍下 Sophon 在广告数据集市的数据规模，有 5 张事实表，50+维度表，超过 500 个维度，数据量压缩以后有 50TB 的数据，大概是 10 倍的压缩比例，压缩格式是 ORC，这个是一个数据集市的量，目前我们要解决的场景就是这个，大部分的查询在秒级或者十几秒内。

简单总结下 Sophon 的特点：首先，它很轻量，结构简单。其次，它比较智能，它会自动的帮你完成很多事情，数据工程师只需要去关注建模，之后它便能够根据用户的查询

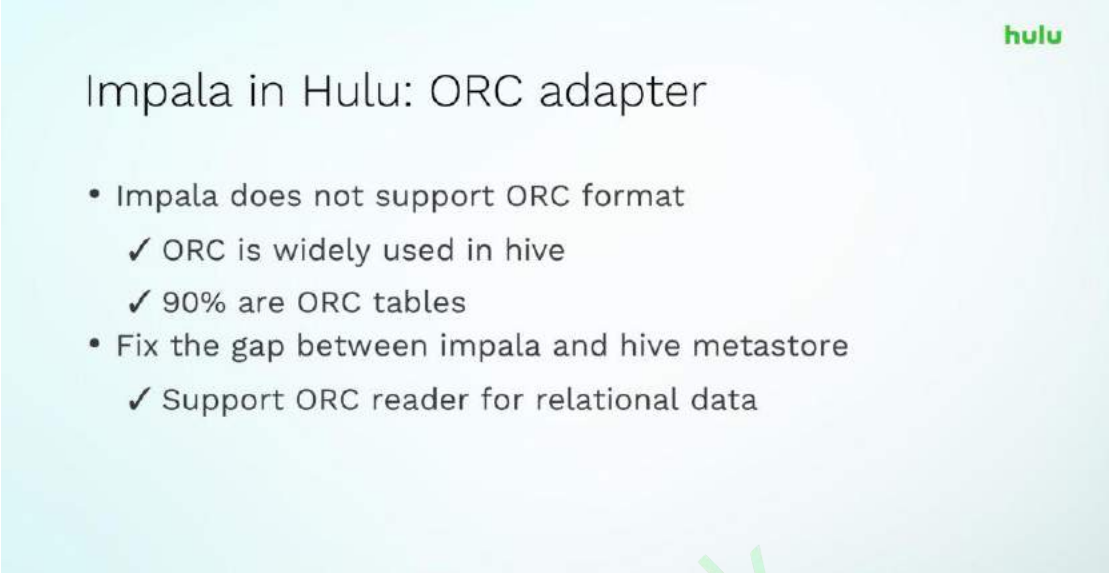
需求来选择应该计算哪些中间结构去加速查询，并自动缓存数据。再次，它比较灵活，它不仅可以直接命中缓存表，还可以将缓存表与其它的表进行关联，由于缓存表的规模一般比较小，这样的场景下，它的速度也可以有明显的提升。最后，它易于集成，由于分离了优化模块和 SQL 改写模块，未来可以集成更多查询引擎。

3. Impala 在 Hulu 的应用

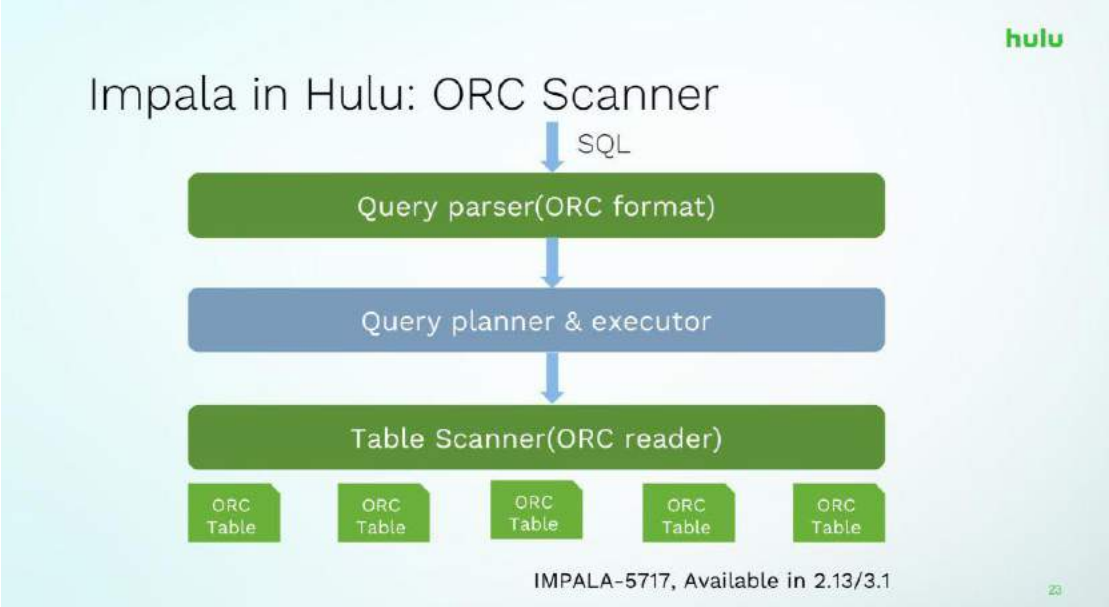
除了 Sophon 以外 ,Sophon 的底层非常依赖 Impala,下面我们介绍下 Impala 在 Hulu 的应用，首先介绍下 Impala 的选型背景，Impala 是一种 MPP 查询引擎，它的查询速度相对 Hive、Spark 要快很多。其次它对 Hive SQL 的兼容性非常好，数据集市的迁移会非常方便。再有它是 C++编写的，OLAP 在很多情况下 Worker 节点之间的 Shuffle 数据量会非常大，对于 java-based 的 OLAP 查询引擎的话，GC 垃圾回收的压力非常大，很容易导致 Full GC，所以 Impala 在内存管理这方面会友好一些。再有就是它向上提供的 JDBC 或者 ODBC 的操作，由于这些特性，我们选择了 Impala 作为我们的后端。

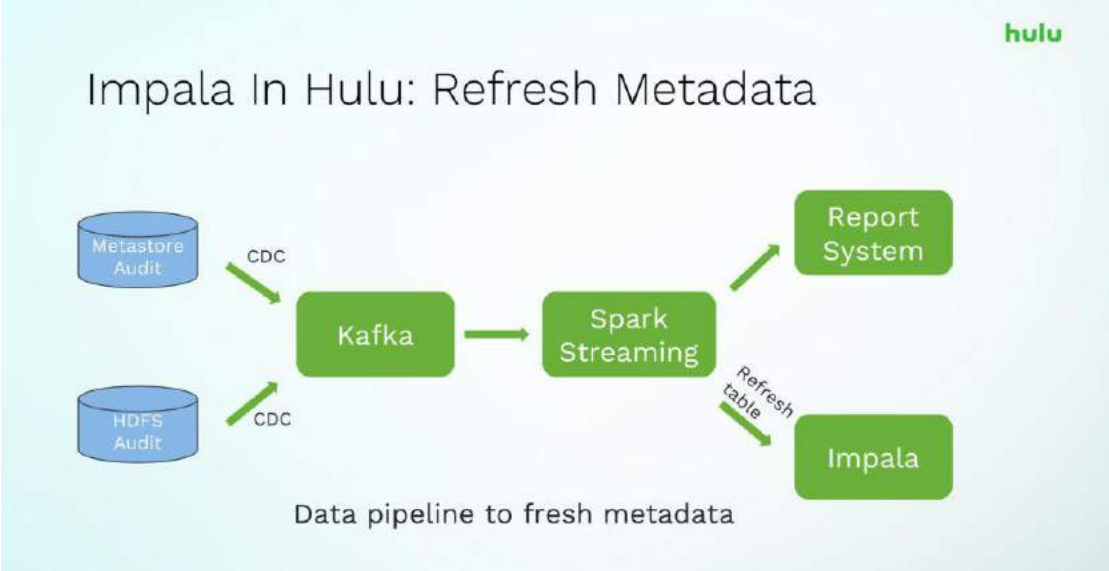


Impala 在 Hulu 有 70 多台机器节点，主要有两个集群，广告的是一个单独的集群，Hadoop 的机器节点有 1 千多台，存储了 Hive 中的 1 万多张表，支持的文件格式有 parquet、orc、text、json、sequence file。Impala 默认是不支持 ORC 格式的数据查询的，这也是很多时候用 Impala 查询 hive 表的痛点，而 Presto 支持的数据格式就会丰富一点，因此我们在这方面做了单独的支持，再有 Impala 有一个问题就是它的元数据是会被 Cache 缓存的，被 Cache 的话，它不能及时的知道 Hive 层的数据变动，而我们的数据导入 pipeline 入库的过程一般来说还是通过 Hive 或者 Spark 这些工具导入进去的，那么怎么在数据导入以后，让数据有及时的感知，这是个很重要的问题。



由于 ORC 在 hive 表中应用广泛，Hive 中 90%的表都是 ORC 格式的。因此如果要使用 Impala，我们有很强的需求使它能够查询 ORC 表，Hulu 在 17 年开发了这个 Feature，在 C++这一层实现了 Impala 的 scanner，目前它能够支持查询主要的数据类型，包括一些比较简单的数据格式的下推，这些工作由另外一个同事完成，目前已经反馈给社区，这个 Feature 会在 2.13 和 3.1 的版本中会被 Release 出来，现在正在开发复杂类型的支持和条件下推，还在和社区讨论过程中。





另外一个问题是关于元数据的，当你有新的数据进来，或者表的 Schema 发生了改变时，Impala 无法自动更新这些变化，查询在执行过程中出错。这种情况下，简单的办法是根据 Impala 提供的指令手动更新元数据。但是我们希望这个过程能够更加的自动化，来减少人工的参与。

对于这个问题，我们在 Impala 外部构建一个 Pipeline 直接拿到 HDFS 和 Hive 的 audit 日志形成一条事件流，通过 Kafka 灌入 Spark Streaming 中，来监控数据和表 schema 的变化，当变化发生时，会自动触发 Impala 的刷新，这是一个独立的外部组件，同时也有一个报表系统，会具体记录用户对集群做了哪些操作，如果这个时候有人误删了一些表或者一些有害性的操作，报表系统会检测出来，并触发报警。

4. 总结

今天我们主要分享了Sophon的设计方案和Impala在Hulu的应用。Sophon作为Hulu在 OLAP 缓存层的尝试，更加注重灵活性，更注重智能，来减少开发人员的工作，同时也希望它能够处理更复杂的数据模型。在设计这些特性的同时，我们也尽可能的使它成为一个轻量且易于集成的系统。

举重若轻的人人车移动端数据平台

嘉宾：吴水永　编辑：安小宏

关于人人车数据平台，水永老师主要分享了 4 部分内容：

第一部分，整体的架构；

第二部分，Web IDE 的实时计算平台；

第三部分，对于离线结果的 BI 报表平台；

第四部分，移动端的数据驱动实践。

整体架构：

1. 数据门户中涵盖几个平台：

BI 报表、元数据管理、实时计算平台、自助取数平台、数据工单平台、监控平台。

2. 数据平台的发展历程：

元数据管理平台、BI 报表平台、实时计算平台、监控平台、自助取数平台、数据工单平台、统一数据权限平台

3. 大数据平台的整体架构：

DataFunTalk

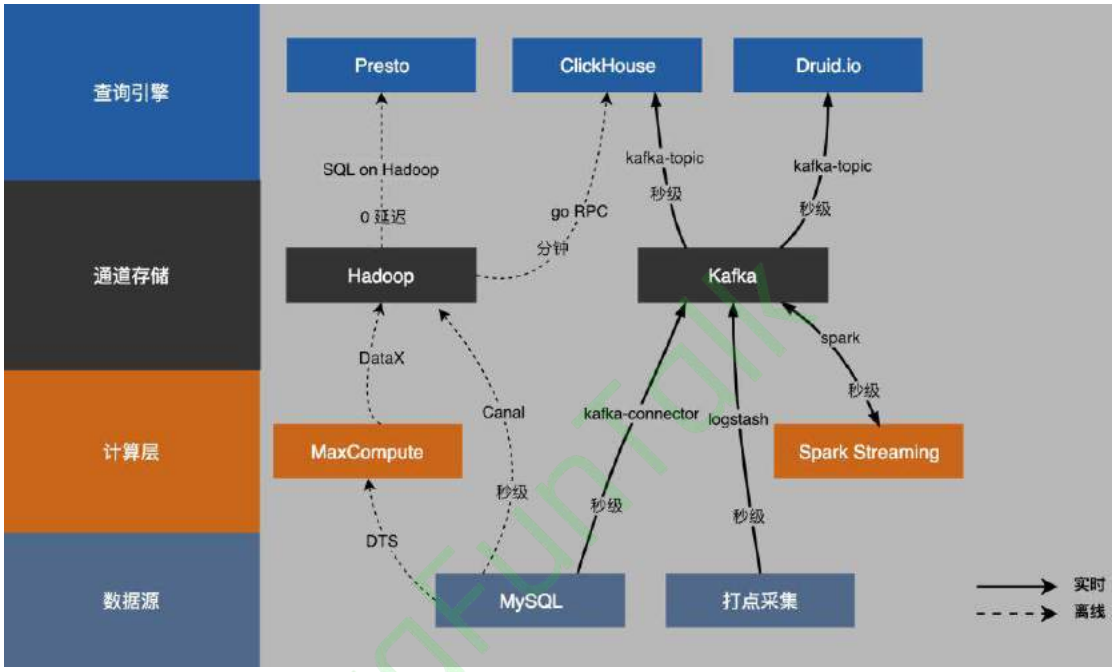
One data，如图，最左边是数据源，有日志、埋点、MySQL、SDK，最后统一落在 Hadoop 上。避免数据烟囱，降低计算、存储成本，保障数据指标口径一致，各种场景下看到的数据一致性，为上层建立了统一的数据服务层 One service 奠基。

One service，数据的下游使用方很多，不可能让每个使用的人直接查表，库。在这种状态下定义一个 Service 层，也就是 api 层，对外暴露查询服务。这些是 service 层以下的引擎，底层有很多的引擎，对上层应用来说是无感知的。比如时序数据库 Druid，包括 SQL on Hadoop 模式的 Presto，spark SQL。还有一些列式存储的数据库，比如 ClickHouse。数据的使用通过数据元数据平台申请，按表级别粒度管控。Service 层保证数据的一致性，不论是查 Druid，还是 ClickHouse，拿到的数据是一样的。如果是小量数据则是 restful api，大批量数据则是下载任务。

One meta，统一的元数据管理，包含数据地图，指标管理，权限管控。元数据可能是最后一块要攻克的高地了，元数据是数据的数据，即数据的骨架。承载底层数据生产的

血缘、上层应用报表/指标。使用数据第一步，申请权限，账号。在指标百科中可以看到自己使用的数据指标在哪个仪表盘中，在 BI 平台的仪表中申请权限，大大提高数据分发使用的效率。有句话说的好，元数据做得好，产品技术下班早。

4. 数据的流向



如图，左侧是它的分层，从数据源到计算层，到数据通道，到引擎。从下往上，左侧虚线是离线的数据，右侧实线，是实时的数据。

首先数据从数据源落到自己的 Hadoop 集群上。再往上是 SQL on Hadoop 模式，零延迟，可以快速查到数据。再往上，实时部分，使用 Spark Streaming 去做 Kafka 通道的转换，支撑订单预估，风险评估等。

Lambda 架构很理想，但在数据的一致性和准确性的挑战上，大大增加了复杂度。对离线数据和实时数据上的应用和数据流进行拆分，业务分离，对不同的领域用专业的方案去解决。

实时计算平台：

实时计算平台是基于 Apache Spark 构建的一站式、高性能实时大数据处理平台，提供了一整套 Web IDE 用于编写 SQL，语法检测、调试、发布、监控。对于只会写 SQL 的 BI 同学来说，提供 SQL 语义的流式数据分析能力，可大幅降低流数据分析门槛。BI 同学无需关注实现细节（将 SQL 转化为 Spark 流式任务），只需选取输入输出表，根据需求编写 SQL 即可实现一个实时计算任务。平台当前支持 MySQL、Kafka 两种数据源，并支持异构数据源 join,及多流 join（限定场景），支持用户自定义 UDF，同时为了应对不同的场景需要，支持了三种端到端的消息传输语义：At-least-once、At-most-once、Exactly-once。

1. 计算引擎选型

实时计算常用的 3 个选型：Storm，Spark Streaming，Flink。

通过如上表格可对三者进行比较，首先淘汰的是 Storm，因为它不支持使用 SQL。

那剩下的就是 Spark 和 Flink 了，两者都基于内存计算，而且都支持 SQL，尤其 Spark 在 SQL 方面做了深层次的优化。

虽然两者在流式计算方面做的都比较好。但两者还是有区别的，Spark 当前支持两种引擎 Streaming 和 Structured Streaming。Spark Streaming 是 Micro Batch 模式，Structured streaming 即支持 Micro Batch 模式，又支持 continuous 模式（以流方式处理）。Flink Streaming 以流的方式处理流数据。Spark 和 Flink 当前都可以满足我司在实时方面的需求，鉴于 Spark 社区更加活跃，更加稳定，而且在 SQL 优化方面做的比较好，最终选择 Spark 作为计算引擎。

2. 实时计算平台整体架构

实时计算数据流如上图所示自左至右。数据源从左侧打点开始，业务 RD 在前端和后端会由不同的语言打点。通过统一的 log 打点平台(log.renrenche.com)接受打点过来的数据。数据收集上来之后，开始数据平台接入。数据上来以后做一个解压和还原，还原之后进入到 Kafka 相应 Topic 中。实时计算平台将 Kafka topic 作为数据源，抽象为 View 进行使用。BI 同学编写 SQL 时可直接作为表查询使用，输出结果可根据实际需要选择输出到 Mysql 或者 Kafka 中。进入到 Mysql 的数据可直接用于业务需要进行展示，进入 Kafka 的数据可通过同步工具，同步到 ES 或者 druid 中进行报表展示或者实时预警。

3. 实时数据管理

数据采集端有很多 Client 进行打点，将数据收集上来，这些数据如何管理？我们强调一个数据工单的平台，其最大的好处是，在下拉单选页面申请一个 data job（业务线-

部门-功能-xx），并填写其用途，下游消费方。当下游拿到这个 data job 时，可以在平台上查到 Schema，这个 Schema 就是后续的一个基础。

4. 流与表的转换

数据处理有一个非常核心的概念就是流和表，在实时里就是一个流，在离线里面就是一个表。Kafka 完美的诠释了流和表，因为 Kafka 既能为数据定义 Schema，同时具有一个持久性以及可重放性，所以它能够完美被 Spark Streaming/ Structured 读取和解释。平台根据 Topic 对应的 Schema ,会将其生成一个 base view。拿到这个 base view 之后对于用户来说，将这个流变成了一个表。有了表之后，就可以让用户进行 SQL 实现一个实时任务，可以对表进行聚合等离线常规操作。也可以异构表 join，多流 join 等操作，底层流 join 由 Spark Structured 引擎支持。

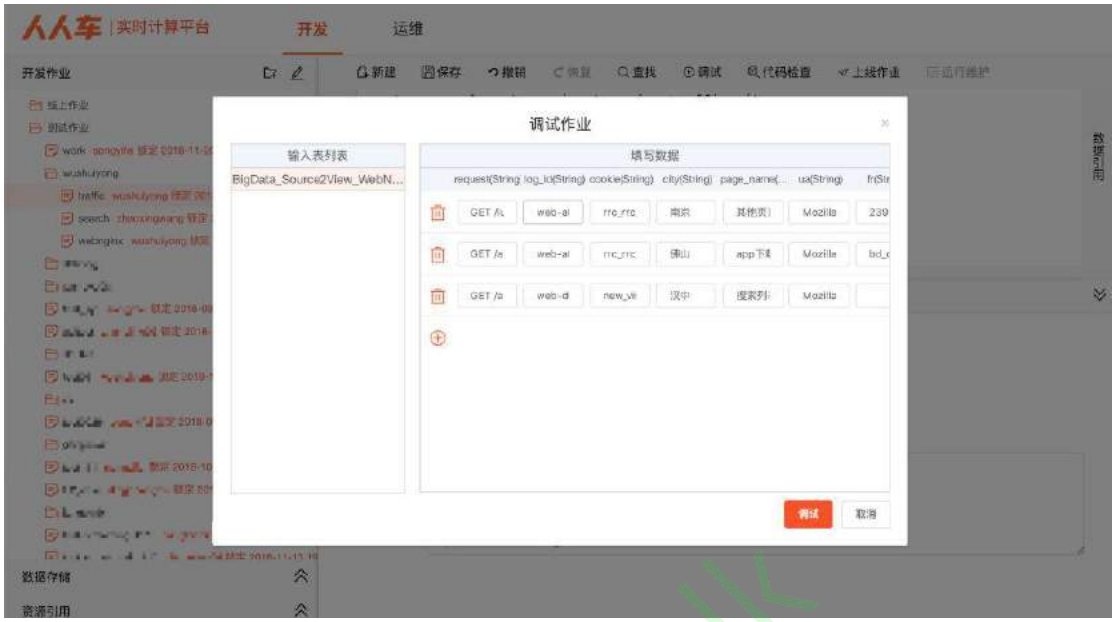
计算之后的结果要被下游使用 ,又回写入到 kafka 中去 ,谁让 kafka 是最好的数据通道。实时的特点就是时序性，数据应用一个引擎最好的方式选择一个时序数据库，我们选择了 Druid。

5. Web IDE

左侧上方的是目录树，方便进行项目管理。左中是数据源，包括 Mysql 和 Kafka，亦可以自行添加数据源。左下是管理的 UDF，引入进入项目即可使用，为了优化引入的 jar 包，亦可自行添加自定义 UDF。右侧是主界面，写完 SQL 即可进行调试、排错，以及构造数据验收，屏蔽掉实时环境、DataSet、DStream、RDD 等概念。IDE 中包含，调试，监控，查看血缘关系功能，对于非实时开发的同学而言，是极低的理解和使用成本，完全可以 WEB IDE 上自行完成一个实时任务开发。

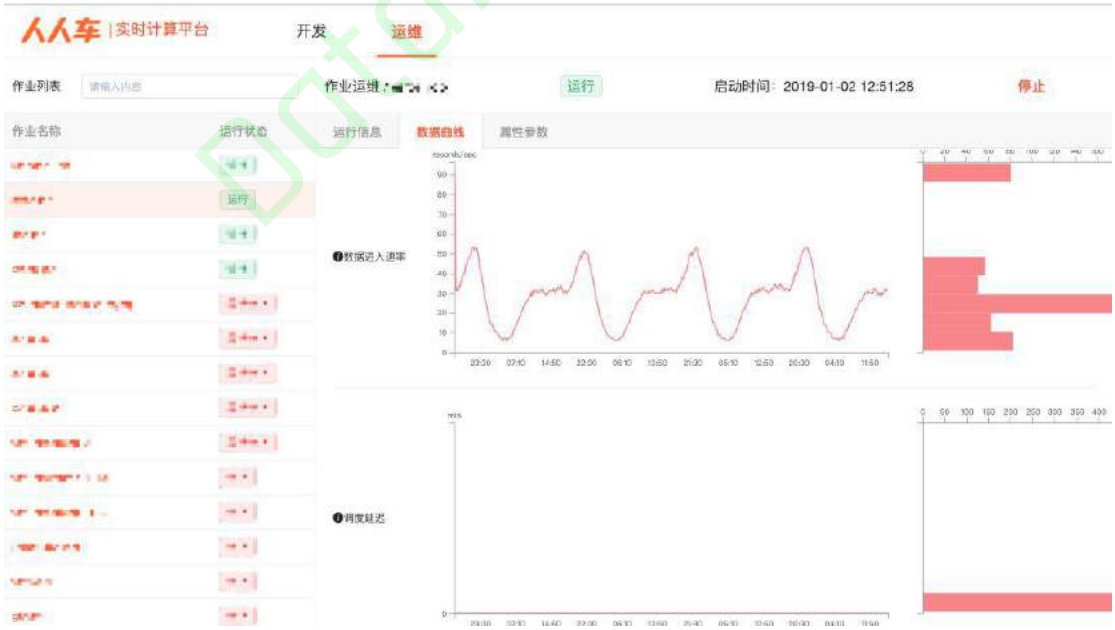
6. 调试

自动从数据源 Mysql/Kafka 中取前几条，填充到每一个字段中去，得到构造数据，亦可以手工自定义数据，即可调试和结果查看。整个过程是与生产环境隔离，在 docker 环境完成，任务与任务之间相互隔离。



7. 监控

根据 Spark 暴露出来的 API 实现监控流数据的状态，可时刻监控数据流入以及调试延迟信息。



8. 血缘关系

方便 BI 同学进行表间依赖关系的梳理，表有类型三种类型：真实表、匿名表、临时表，尤其排查复杂的 SQL 时，此处一目了然。

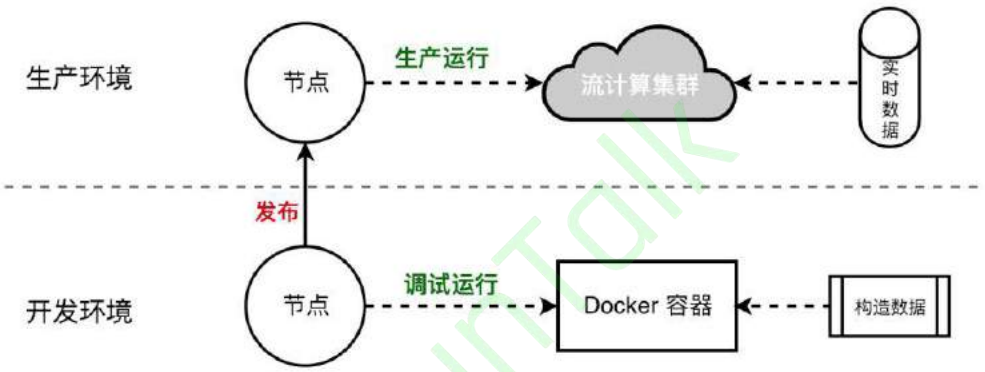
9. 参数配置

Web IDE 意味着线下 IDE 的功能都要有，包括可以配置参数，比如配置信息，优化参数等，UDF 的配置。对于各个参数要有默认参数。

10. 开发-调试-发布

手动提交实时任务时都是使用 spark-submit ,但这种方式并不适合进行自动化提交(依赖客户端环境、上线速度慢、不便查看任务状态），最终确定基于 YARN-API 进行提交。所有的控制都在一个 API 中，无需上传依赖 jar，上线速度快，也方便查看任务当前状态，同时 YARN-API 是更友好的 API 接口。这样就可以很轻易的去封装每个参数。

封装 YARN-API 的过程中包括了调试和发布。在 Docker 容器中编译调试，验收通过后发布上线，接入生产环境 MySQL 和 Kafka 数据。



11. 最后总结下实时计算平台的特点：

BI 报表平台：

我们的目标是，构建一个小白级拖拽，所见即所得，包含移动端的数据平台。而在技术上需要考虑几个大难点：

低延时。包括数据更新低延时、数据查询低延时，在灵活的报表平台上，数据预构建类的引擎几乎不用考虑，需要精心管理索引的成本亦高，能在千万记录级别百余字段的数据基础上，亚秒或秒级别。

高并发。低延时的一个克星。没有大内存，MPP 架构就难以施展能力。即使有大内存，复杂度和高并发下，同样也难以通过 benchmark。

同步中全量增量的数据一致性。全量和增量同步还需要考虑主键去重、分区去重，一般的 OLAP 引擎数据写入都是 append 模式，几乎不支持 update 和 delete。

1. 技术选型

在这些眼花撩乱的数据引擎中，各自有着自己特性，在不同的业务体系中，总有他们发挥的场景。我们需要一些标准评估哪个引擎更适合自身的业务场景，POC 产品原型验证以及 TPC benchmark 压测。

自古磨刀不误砍柴工，梳理清楚业务场景才能在对每一个引擎做 POC 时高效靠谱，不至于贸然上船而到后期骑虎难下。若能转化成 SQL 场景，则在 TPC-DS/TPC-H 压测下更能科学判断一款引擎的场景能力。**前者 POC 是定性分析，决定引擎能不能；后者 TPC 是定量分析，决定引擎到底有多强。**

2. 一见钟情 ClickHouse

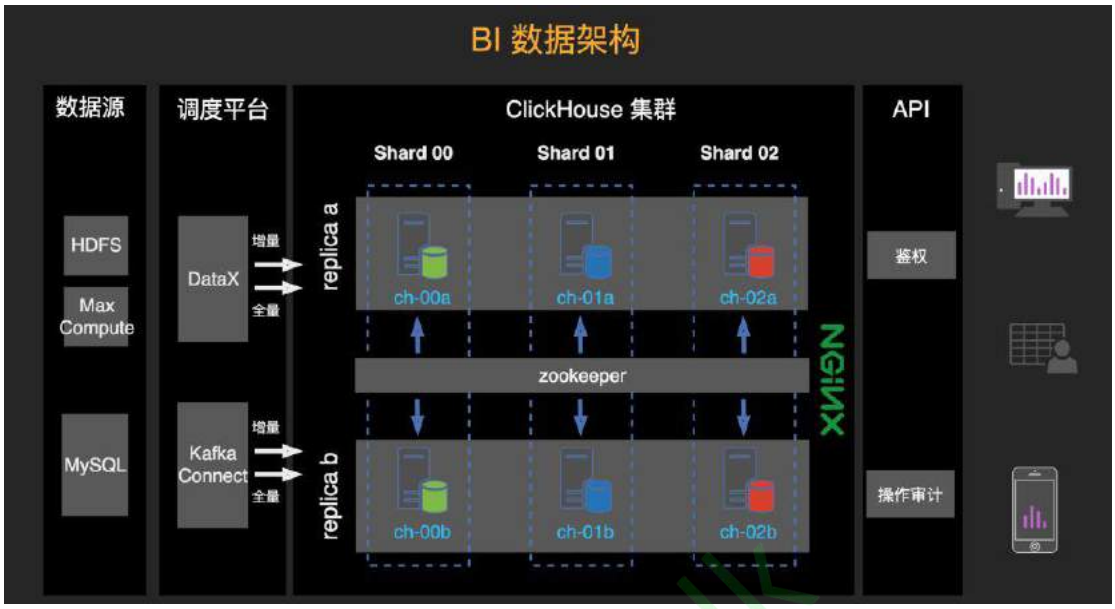
人人车的 BI 平台选择了 ClickHouse，支持较高的并发，支持亿级别量级查询，支持增量、全量同步，支持幂等性去重，支持更新 update 和删除 delete，亚秒级延时，支持 SQL。用自身业务的 sql 跑压测，4 台测试机就跑出了 10qps/30 并发，已经是一见钟情了。

DataFunTalk

列式数据引擎 ClickHouse，有很多特别适合 BI 场景的特性。在我们使用 SQL 的时候往往查询的列是很少的，列式数据库可以让我们在做聚合等操作时，只需要取出少量的列，可以大大减少内存与外存之间的数据交换。

ClickHouse 性能确实太强悍了，在一台 4 核 8G 的机器上，对一亿七千条数据 count，跑了 3 秒多。在第一次没有任何缓存的情况下，多维度 group by+order by 跑了 9 秒多。在我们看来 ClickHouse 如同 AK47，简单可靠，性能强悍，手动性强。换言之大部分需要自己配置，如分布式和副本集，数据去重幂等性。Clickhouse 已经支持 update/delete ,方便做数据的更新 ,但在大数据的性能要求下 ,ReplacingMergeTree 在实践中仍然是更好的选择。

3. BI 数据架构



DataFunTalk

ClickHouse 借用 Zookeeper 实现集群管理，高效管理副本和分片。副本可以提高集群的稳定性，分片用于做性能的扩展。ClickHouse 的副本，不仅可以提高稳定性，还可以提高性能。发来的请求可以均匀的落在副本上，压测过程在几乎性能随之线性增长。

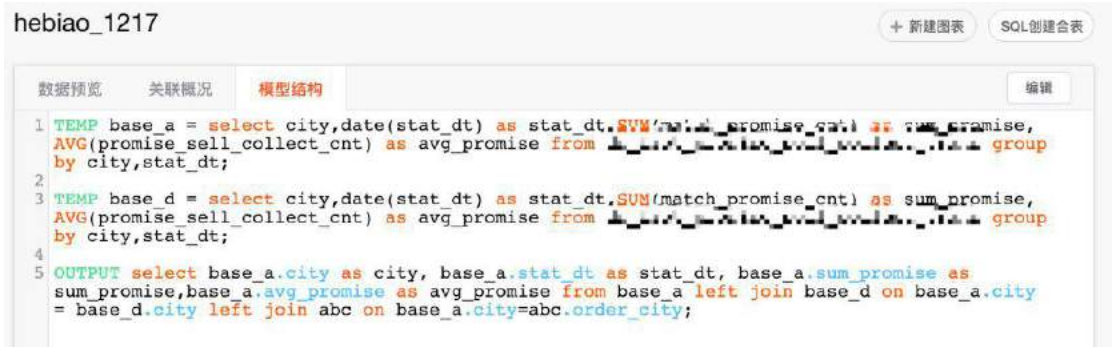
4. BI 报表平台

单表查询，选择一个表，左侧列出所有字段，亦可自定义配置字段名。拖拽字段到维度和数值上，右侧即会高亮显示可用图表类型。亦可增加对比轴，数据源过滤，图内筛选器，图表生成后前端方便用户自行筛选城市，时间等。

图表编辑生成后，三端（PC、Android、IOS）同步原样展示。

5. 高级功能

生成合表。除了单表查询外，还可以自己写 SQL join/union 生成新的表，仪表盘中的图表可以在合表的基础上输出。适合一些有能力的分析师对数仓的主题表做深度关联分析。



合表的血缘关系，上游的表是否生成，是否同步完成，以及被哪些合表作为源表使用，被哪些图表使用。另外高级用户可以根据自己的权限从数据仓库中查询数据，已经在全家桶中 AD-HOC 打通。

6. 最后总结 BI 报表平台特点：

数据驱动：

我们线下有两万多人，包括销售，评估师等，线上有上千人，包括运营，产品等。线上做数据分析和线下执行完全不是一个思路，线上的总部可以在 web 上在几十个仪表盘关注几百个指标，但是线下人员一个手机。

其次，要分清线下管理的难点和本质。线下评估师销售面对的是各色各样的人，还要跟黄牛斗争，直面金钱的诱惑（飞单），让员工每天在一个开放的空间里做人性的考验，这种商业模式是不明智的。不要尝试与线下斗智斗勇，犹如秀才带兵打仗，如果没有一个有力的管理抓手，将会在无数次学习中成长。

如果管理抓手刚好又能跟数据结合，实现对线下的业务管理和数据管理，这就是数据驱动。

数据驱动一直是经常听说，但一直没有亲眼看到的东西，尤其在这种复杂的业务场景中。线上总部数据分析在 PC 端，做更全面深度分析，制定出与战略相匹配的业务指标，且与业绩直接挂钩。并且特别为线下提供掌上数据移动端（Android，iOS），使得线下随时随地只关注和自己业绩相关指标。考核什么就会得到什么，使线下人员的注意力聚焦自己每天业绩，各层级管理人员数据汇总，无论是在数据流通和人员调动上，管理抓手更为高效。

DataFunTalk

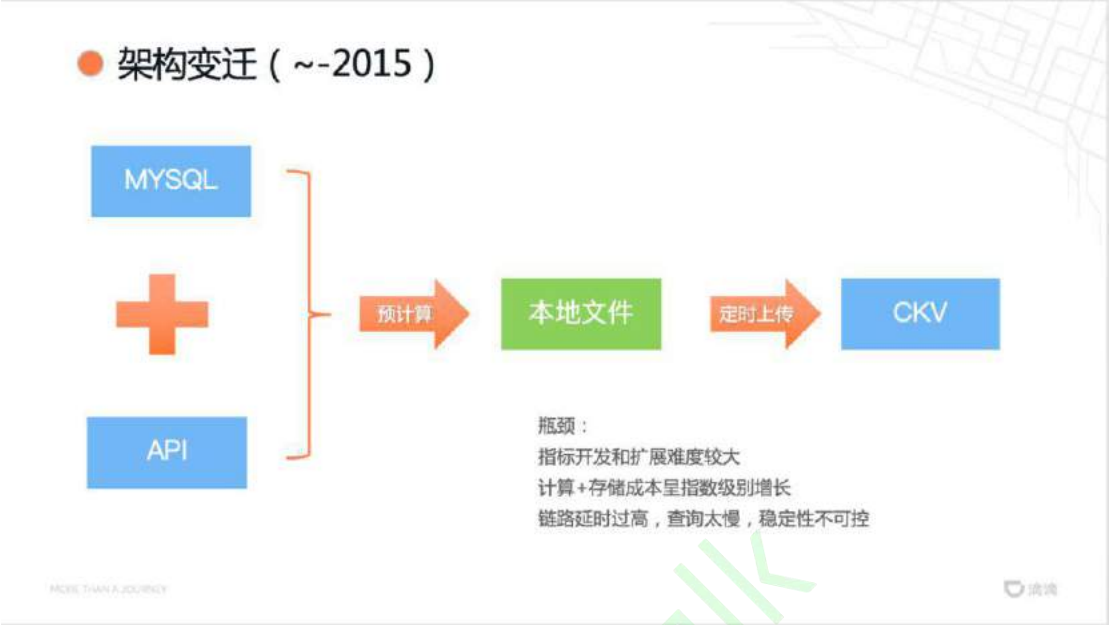
时序数据在滴滴实时数据开发平台中的处理和应用

嘉宾：张婷婷　编辑：孙博文

本次分享内容：

- 时序数据如何与滴滴实时计算平台产生联系；
- 当前使用的技术方案；
- 基于当前技术方案能够达到的效果。

首先了解到该平台的历史使命：作为业务的双眼，能够及时的发现数据的变化，同时拥有配套的报警能力，一旦发现数据的异常能够及时的通知到相关人员。



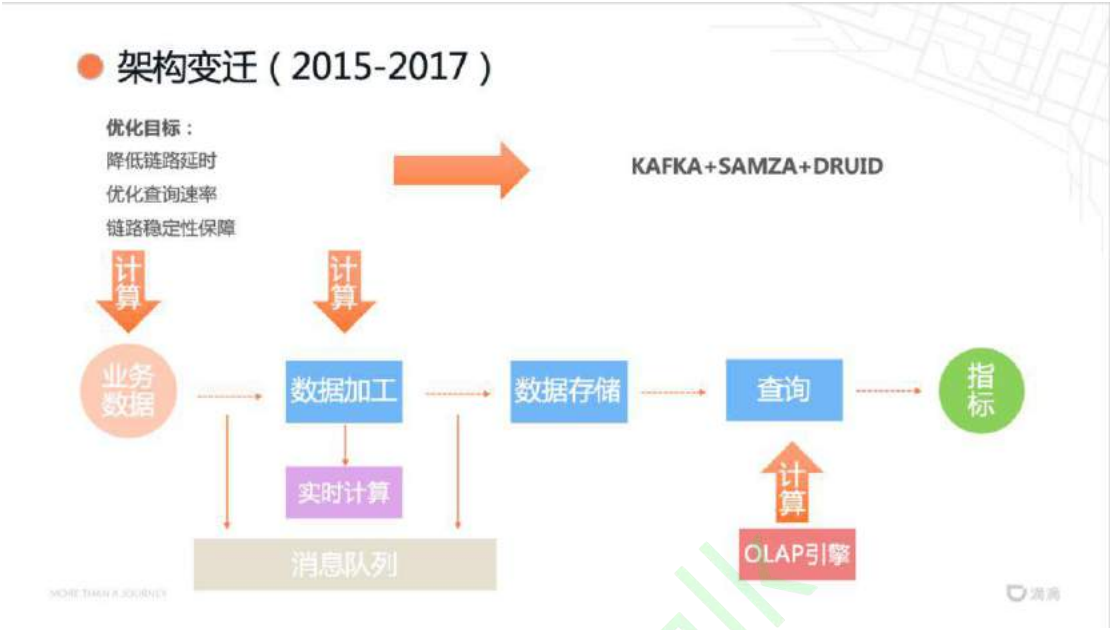
然后看一下该数据平台的技术架构演进历史，初始的架构采用扫描 MySQL 从表加拉取 API 的数据的方式，对数据进行预计算，按照一定规则存储到本地文件（例如：对城市级别进行订单量统计，一级目录为城市 ID，二级目录为日期，保存文件）。将本地文件定期上传到 CKV 数据库，对指标进行查询。

该架构的瓶颈很显然，随着数据增长：

- 指标开发和扩展难度大，新增和功能扩展指标需要修改的部分涉及很多；
- 计算和存储成本呈指数增长；
- 链路延时高，查询慢，稳定性不可控。

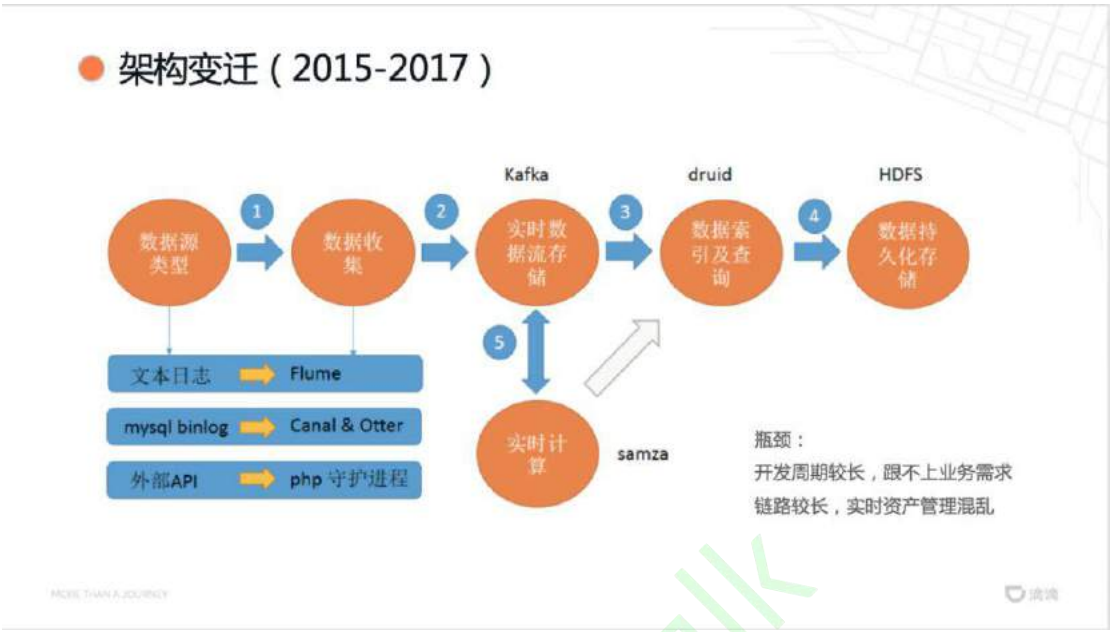
对于实时监控系统需要保证及时性和稳定性，基于这两点对系统进行优化。

首先理清由输入的业务数据到输出的业务指标，中间主要经过了数据加工、数据存储、指标查询的过程。原始的架构，各个部分的连接通过 HTTP 请求的方式，不能保证吞吐量，并且数据可能遗漏。第一步将数据的连接替换为消息队列的方式，提高了吞吐量的同时，利用 MQ 的 ACK 机制保证数据被消费，降低了链路的时延，提高了稳定性。然后，为了使数据加工的能力匹配消息队列的吞吐量，将离线批式的数据加工替换为实时流计算。



接下来就是考虑到指标的计算，计算可以在业务方进行计算、在数据加工阶段进行预聚合、在查询的阶段得到计算结果。采用了第三种方案（查询时计算）的原因：计算的阶段越靠后，计算前的部分的复用性和性能可以大大提高。例如，实时订单呼叫量、实时订单应答量的统计，如果在数据加工时通过程序处理，一条数据到达需要经过两个实时任务的计算，而且会持续的消耗计算资源。如此，指标的增加，计算资源膨胀严重。而如果实时计算只做简单 ETL 处理后得到订单表，基于此表可以查询多种指标，可以明显提高计算性能，和该表的复用性。

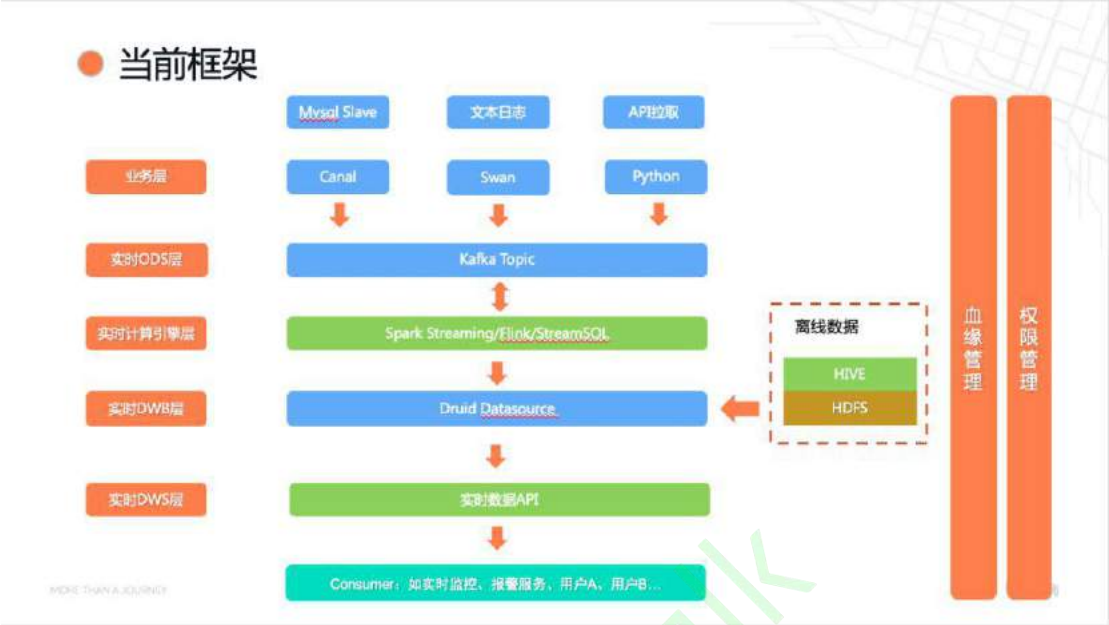
因此对数据库查询的能力有比较高的要求，我们引入了 OLAP 引擎解决。由于业务数据天然的具有时序的特性，采用时序数据库，选取的技术方案为：Kafka+Samaz+Druid。



第二阶段的平台架构如上图所示，但是存在以下开发瓶颈：

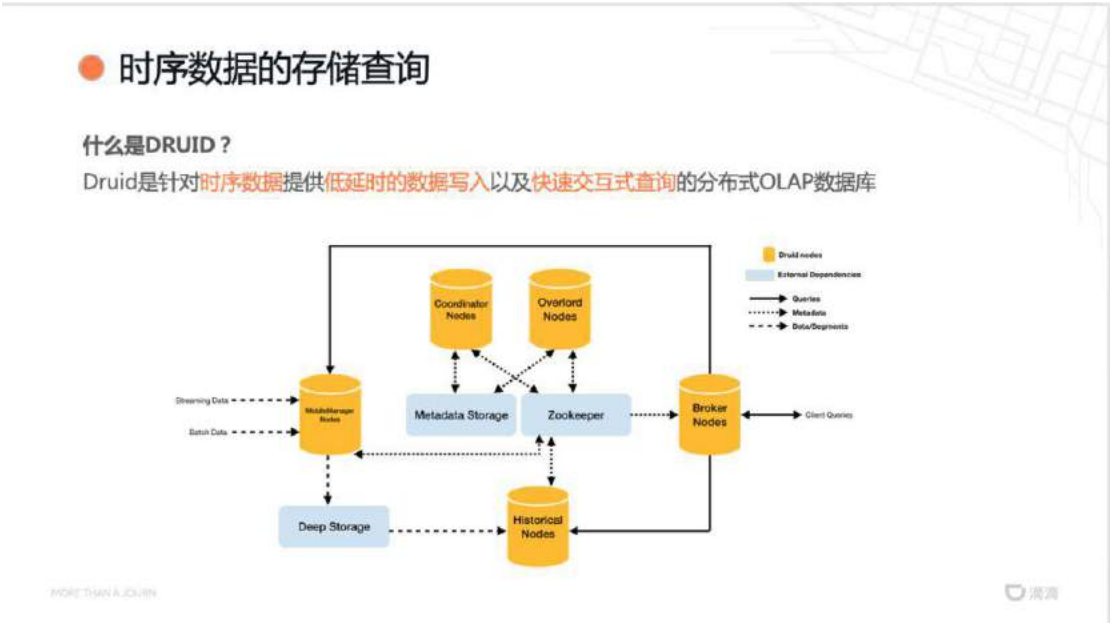
- 开发周期比较长，每个需求的实现都需要投入相应的人力；
- 链路比较长，不同的模块有不同人员负责，管理比较混乱。
- 因此考虑到将平台的能力开放出来，让所有人都可以参与到实时数据的加工和报表的产出。

考虑到 Druid、实时计算引擎、开发语言等不是所有人都了解，需要一定的学习成本。而且大部分人参与进来，对资产管理提出了新的挑战。从而提出了滴滴实时计算开发平台（Woater）。主要目标：降低开发难度和资产管理模式的优化，得到当前的技术方案如下：

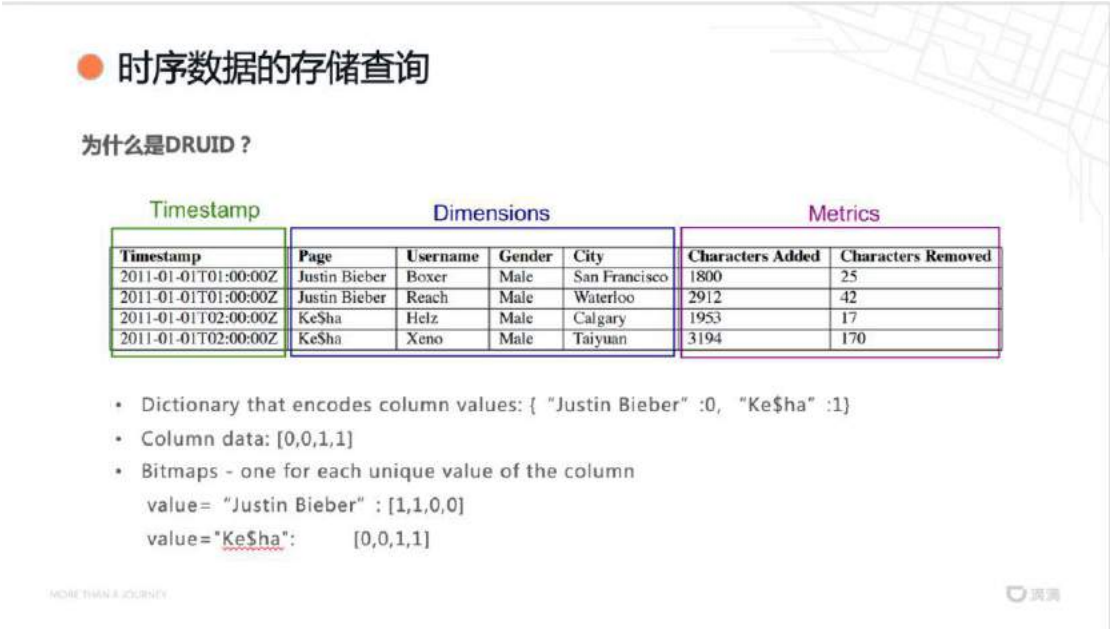


数据通过多种方式接入到 Kafka，通过 Spark streaming 或 Flink 处理后写入 Druid。考虑到 Kafka 对消息保存的生命周期，添加了对 Druid 数据进行一些离线的处理（Hive+Druid），保留历史数据的信息。数据落地存储后，通过统一的实时数据 API 对第三方开放（开通权限后可用）。

在主线功能之外，提供了两个支线功能：权限管理和血缘管理。权限可以管理到模块粒度上的开放和使用，通过血缘可以清晰的看到数据的流动路径，查找数据的上下游信息。

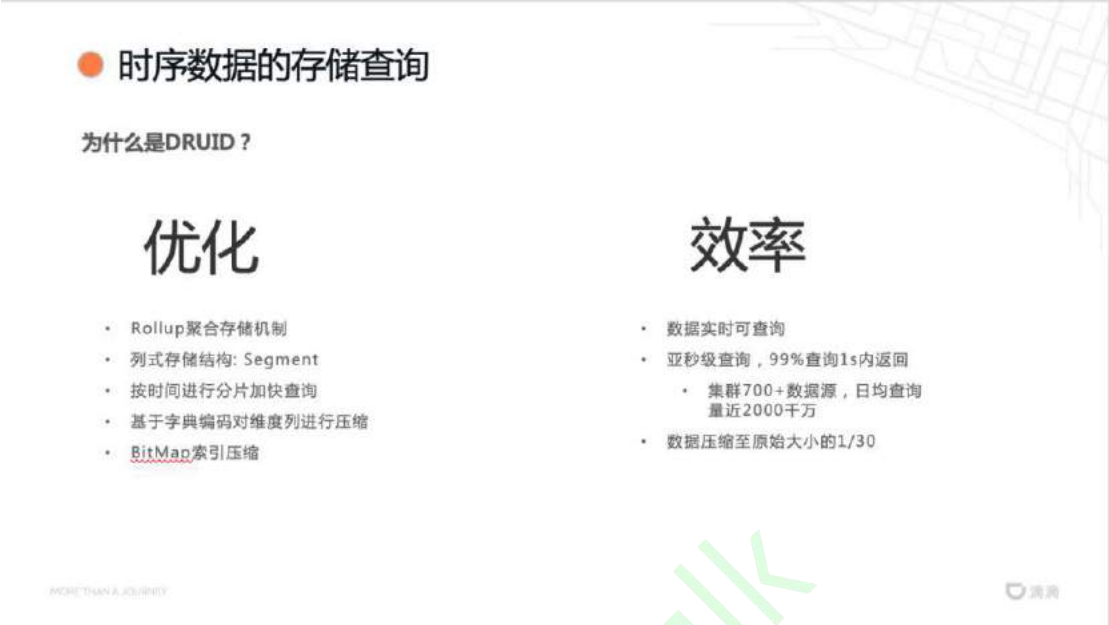


什么是 Druid？Druid 是针对时序数据提供低延迟的数据写入以及快速交互式查询的分布式 OLAP 数据库。核心思想是：Druid 的数据会按时间写入不同的分片，一次查询会拆分为多个子查询，并汇总结果后返回。

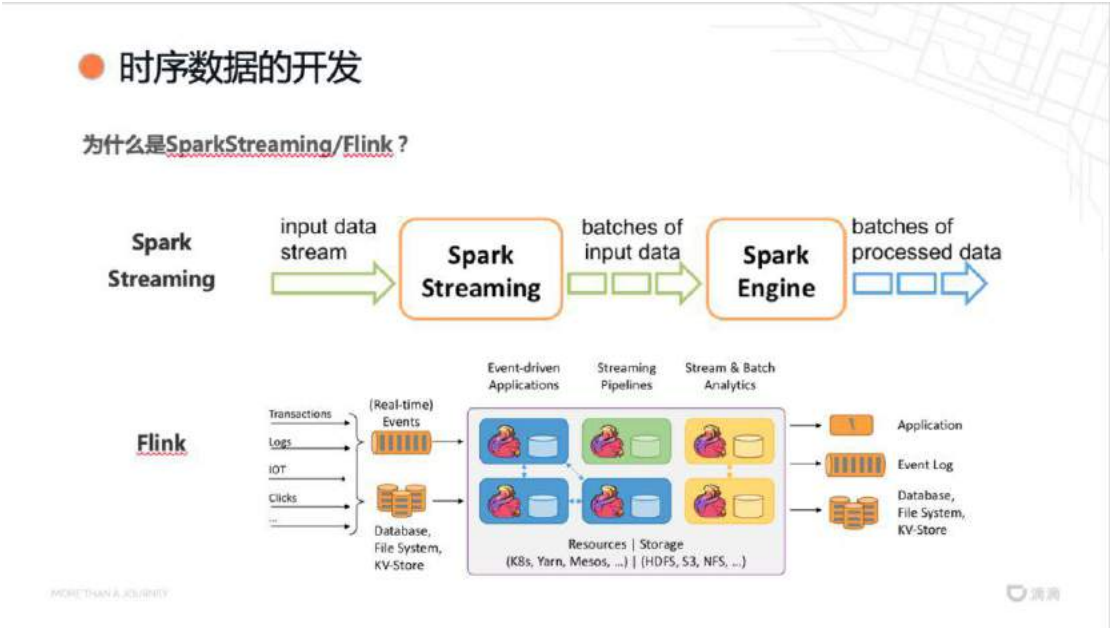


为什么选择 Druid？Druid 对时序数据做了一些优化，所有的数据包括三个部分：时间戳（Timestamp）、维度（Dimensions）、指标（Metrics），使用列存储，针对每一列的数据能够提供很高的压缩比。

以上图的维度列 Page 为例，实际存储的不是真实字符串，会先解析列值生成映射表：{“Justin bieber”:0,“Ke$ha”:1}，实际存储 [0,0,1,1]，代表了 [“Justin bieber”，“Justin bieber”，“Ke$ha”，“Ke$ha”]，节省存储空间，提高压缩比。同时会维护一个倒排索引，如 “Justin bieber” =[1,1,0,0]，代表了第一、二行是该值，第三、四行不是该值。当多个列组合查询时，可以通过位的与、或运算，得到结果行的集合。



Druid 其他的优化点如上图所示，除了前面提到的列式存储的优势，还有 Rollup 预聚合存储机制和 Bitmap 索引压缩优化，按照时间对数据进行分片，能够加速查询，达到数据实时可查询的效果，亚秒级查询（集群规模：700+数据源，日查询 2000 千万），数据压缩至原始的 1/30。



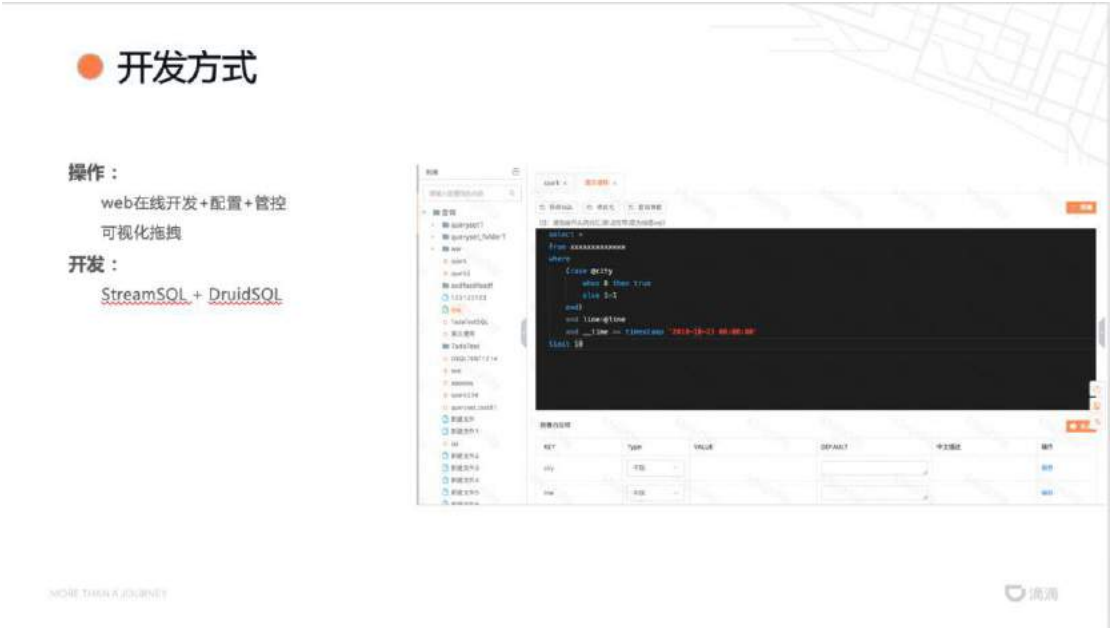
Druid 在查询方面表现很好，但是计算能力偏弱，例如两表 Join 查询的场景支持不够。因此计算的能力由 Spark streaming 和 Flink 来提供。

Spark streaming 基于 Spark 发展而来，将实时流数据拆分为微批数据进行处理。适用于有 Spark 经验的开发人员，对实时要求不高的场景。

Flink 是纯流式处理，每接收到一条数据，就立即计算。同时提供了 Exactly-once 的语义。可以使用于延时要求高的场景。

为什么同时用？适用于不同场景和不同的开发人员，所以两种都支持。

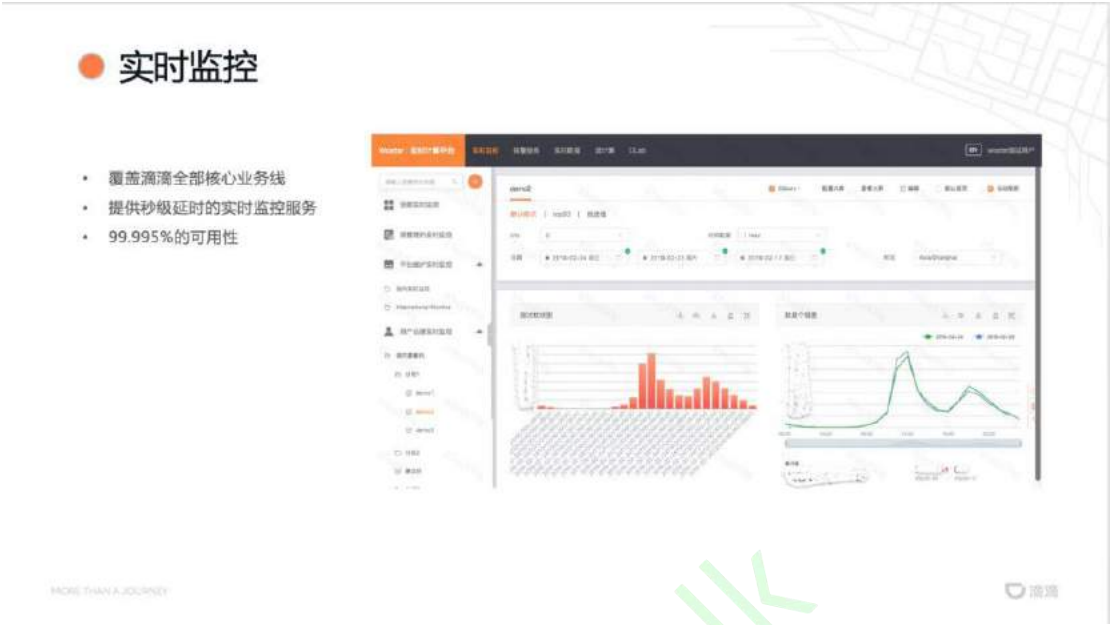
DataFunTalk



总结该平台开发的开发方式提供了：

- Web 在线开发，适用于有一定开发经验的人员；
- StreamSQL+DruidSQL，适用于有一定 SQL 能力的开发人员；
- 可视化拖拽适用于初级用户。

同时，还提供了配置和资源管控的能力。



目前该平台的现状：

- 平台覆盖了滴滴全部核心业务线，提供秒级延时的实时监控服务，达到 99.995% 的可用性。
- SLA 以业务是否发生了真正的问题来评价报警质量。报警服务在准确性上达到了一个月不超过 3 次误报和 1 次漏报，在 1 分钟内能够及时的响应。
- 其他功能如：大中小屏一体化解决方案，更丰富的图表和配置功能。

阿里巴巴双十一千万级实时监控系统技术揭秘

嘉宾：柴武　编辑：杨迪

本次分享主要介绍以下四方面：

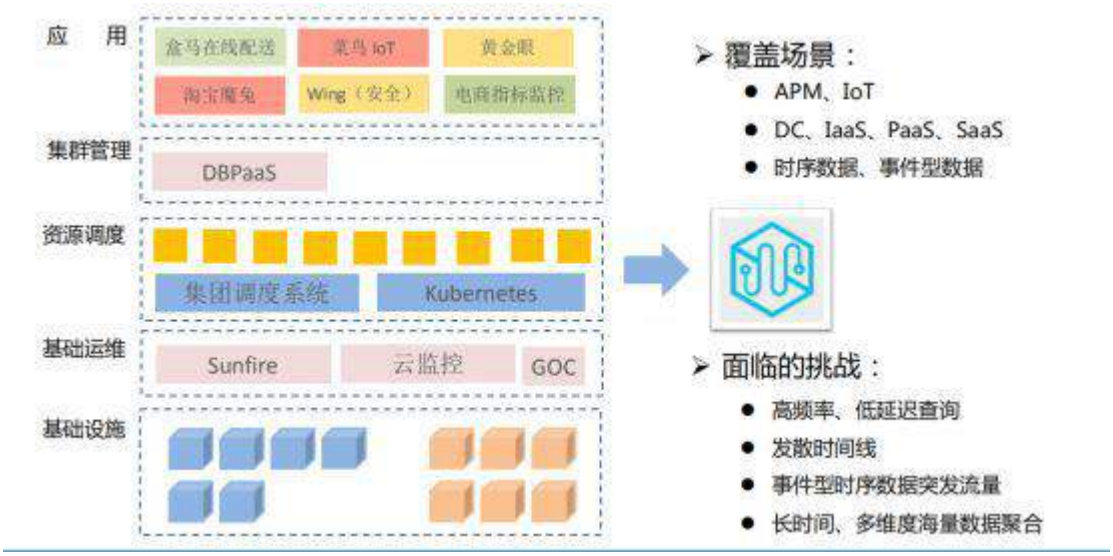
- 时序业务全景
- TSDB 介绍
- 核心技术
- 总结展望

▪ 时序业务全景

从底层的机器监控到直面用户的应用，都离不开时序性的业务场景，而时序性的数据一般都由专业的时序数据库来存储分析，下面主要介绍 TSDB 覆盖的业务场景以及面临的挑战。

1.1 时序数据库覆盖场景

DataFunTalk



基础设施层：机柜、物理机、操作系统监控日志

基础运维：sunfire(集团统一监控、采集、报警系统)、阿里云监控、GOC(阿里全球应急调度指挥系统)

资源调度：集团内部调度系统、Kubernetes

集群管理：DBPaas(阿里所有数据库实例的监控和调度)

应用层：APM 场景下的各种应用

1.2 时序数据库面临的挑战

由于面临各个层级的不用场景，所以时序数据库也面临不同挑战：

- 应用层挑战：由于直面客户所以需要提供高频率、低延迟的查询
- olap 数据库本身特性：海量数据的聚合

- 时序数据库特有的：发散时间线

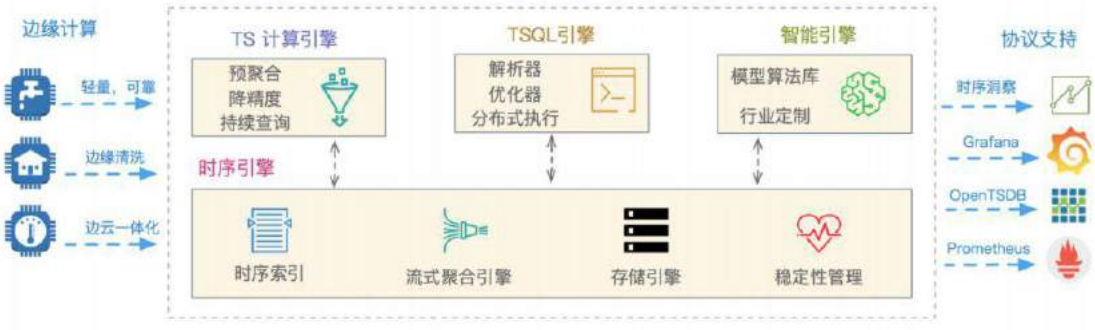
- 双十一大促：突然流量十倍以上增长

▪ TSDB 介绍

2.1 TSDB 的发展及性能

TSDB 于 2016 年开始服役,到目前为止参与了三次双十一大促，相比于 2017 年读写吞吐翻倍增长,写入 TPS4000w ,查询 2wQPS 覆盖集团 130+业务线及存储百亿的时间线。

2.2 TSDB 架构介绍



如上图，从左到右为数据的采集到展现的过程：

边缘计算：轻量可靠的计算方案，主要负责数据的采集，与云端的 TSDB 打通，在 OLAP 场景或者资源不稳定的场景下实现数据的稳定采集、清洗等。

时序引擎：

时序索引：时间线的查询；

储存引擎：时序数据、海量数据存储的解决方案；

流式数据聚合：在时序数据库的海量数据里做高效的的聚合分析；

稳定性管理：在云上稳定安全的运行；

计算引擎、sql 引擎、智能引擎：主要与时序引擎交互实现数据计算、sql 解析、模型算法等功能，可以扩展时序引擎的能力，降低使用的门槛；

协议支持：主要面向用户，为用户提供一些可视化的查询和分析支持。

▪ 核心技术

3.1 海量时序数据存储

3.1.1 数据压缩

说起存储就离不开压缩,数据的压缩方法和压缩算法的选择很大程度上支持了海量数据的储存。

如上图代表时间窗口为一小时的数据，0-3600 代表过去一个小时内的数据，采用 key-value 存储格式,以秒值作为 key，每秒的数据作为 value 存储。

这里参考 facebook grada 思想引入了时序压缩算法，通过列合并的方式把所有的时间戳和对应的 value 聚合长两个的大数据块，然后对这个两个大块进行时序压缩算法，然后再用通用的块压缩算法进行压缩。

另外不同的数据类型采用不同的数据压缩格式，如：

时间戳：dalta- dalta

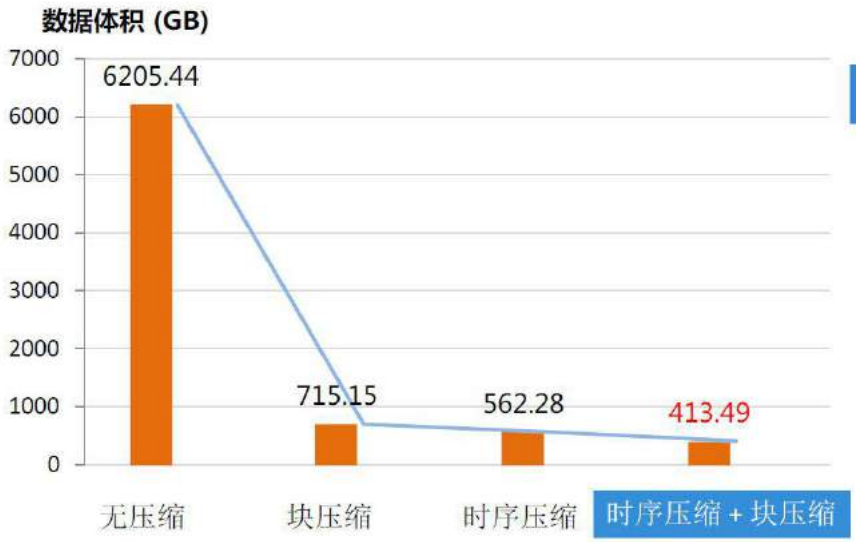
浮点型：XOR 编码

DataFunTalk

整型：variable length encoding

字符串：LZ4，实现了存储层乱序数据压缩，保证压缩数据的准确性,整理的压缩率在15:1。

3.1.2 数据压缩效果



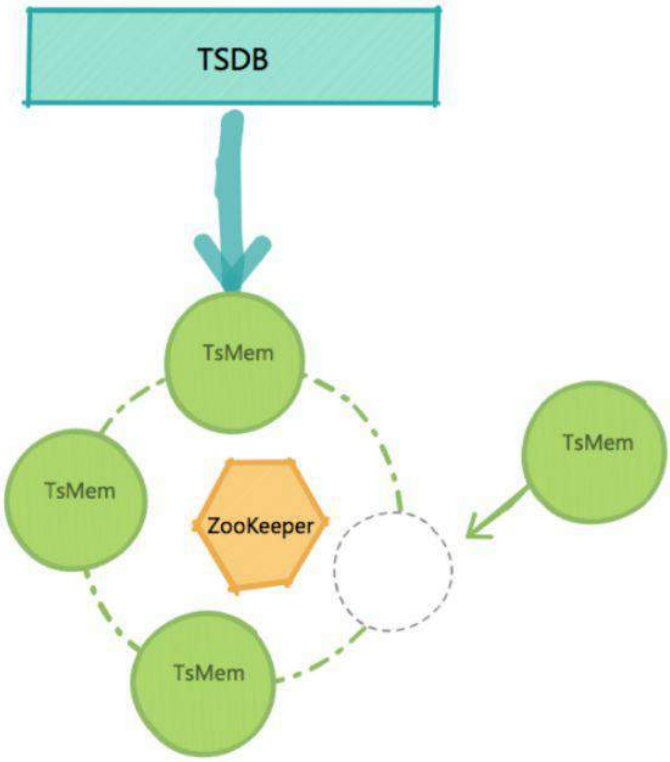
为什么要采用时序压缩+块压缩，我们可以看一下这个图。

首先时序压缩针对不同类型的数据采用了不同的压缩格式，所以整体的效果优于块压缩算法，而在时序压缩算法对数据压缩后在采用通用的块压缩，不会影响到块压缩的压缩效率，用时序压缩+块压缩相比单独的块压缩能有 40%的压缩率提升，这为海量数据的存储提供了有力的帮助。

3.2 高频、低延迟查询

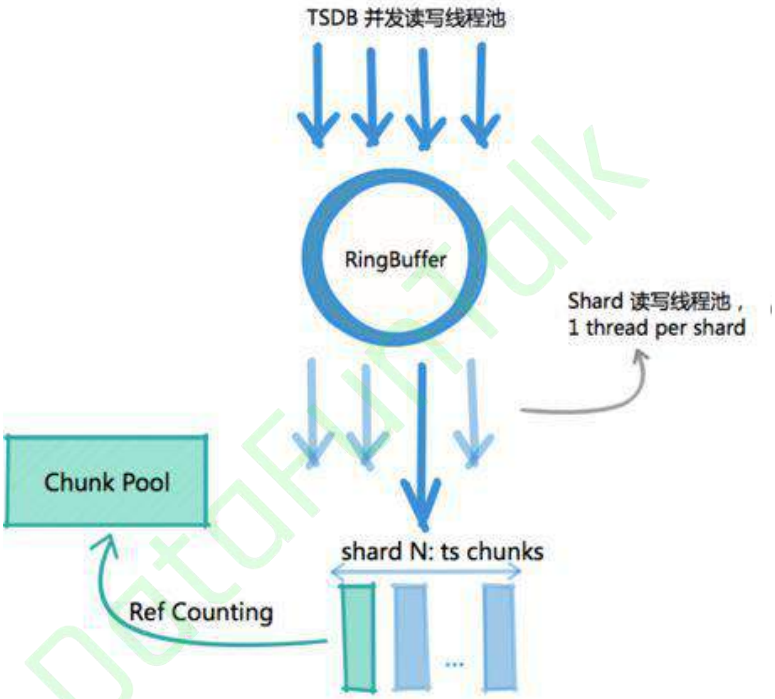
淘宝魔兔是阿里一款应用无线端数据分析和监控的产品，支持集团内部 500+的应用，在双十一大促是查询峰值可达 4000QPS，相较于平常查询量有 10 倍的提升，,99%读写 rt 都要求在 20ms 以内，那么 TSDB 是如何实现用户端高频、低延迟的查询呢?

3.2.1 分布式缓存存储适配



参考 Facebook Gorilla 论文，基于 java 做了一套分布式的内存缓存存储，基于 zookeeper 实现分片及容量的调整，可以实现动态的扩容和缩容，在整个双 11 过程中支持 1000wTPS 的写入和 4000 的 QPS 的查询。

3.2.2 TsMem 设计



如图所示，TsMem 基于 Disruptor 做一个 RingBuffer，把用户读写的请求都暂存在 RingBuffer 中，采用多个生产者和一个消费者模式，一个消费者的请求会打到多个 worker 线程中，每一个 worker 线程又是一个分片，所以其实就是基于 RingBuffer 做了一个内存的分片，这样一来就是一个线程对应一个分片，这样就不会产生共享资源，也就无需考虑锁的实现。

把写和读都分配到一个链路上，一个 worker 同时处理读和写，提高读写性能

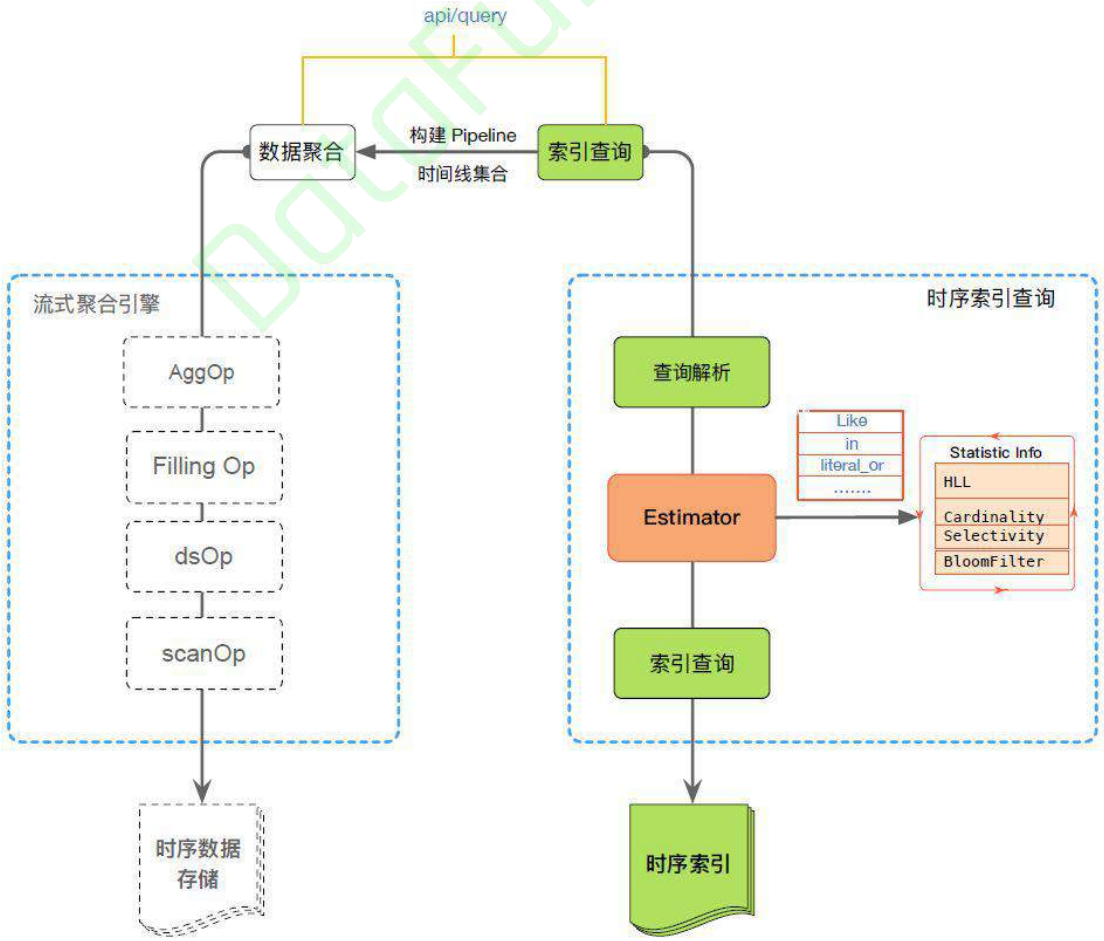
同时还利用了 RingBuffer 的 batching 特性，将用户的读写请求都暂存在一个 butching 中，然后当达到一定阈值或时间 worker 将直接提交一个 batching，这样虽然会是请求有一定的延迟，但是大大提高了 worker 的吞吐量。

那么如何保证高效的内存管理和极低毛刺的延迟呢？

对于数据块基于引用计数的 chunk 池化管理，把所有的时序数据块在内存中做了池化，这样就能减少读取数据时临时对象的创建，而且还能避免大块时产生的抖动和延迟。

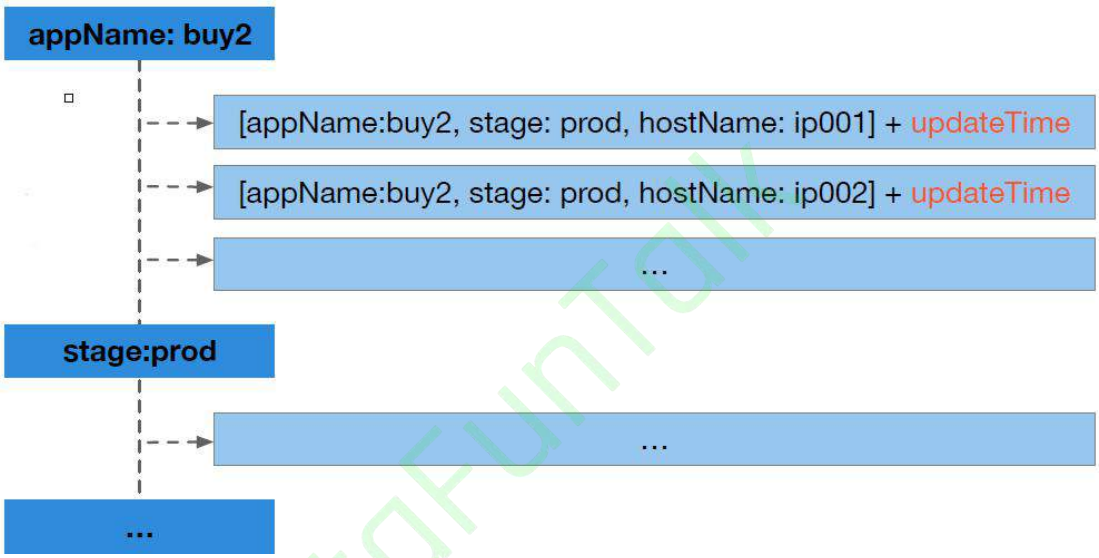
3.3 高纬聚合分析

3.3.1 TSDB 引擎核心模块



TSDB 和核心模块主要包含两部分，分别是时序索引和流式聚合引擎，当用户的一个查询请求过来时将会由时序查询引擎返回对应的时序索引构建 Pipeline 时间线集合，然后再由流式聚合引擎计算出相应的聚合结果。

3.3.2 时序索引



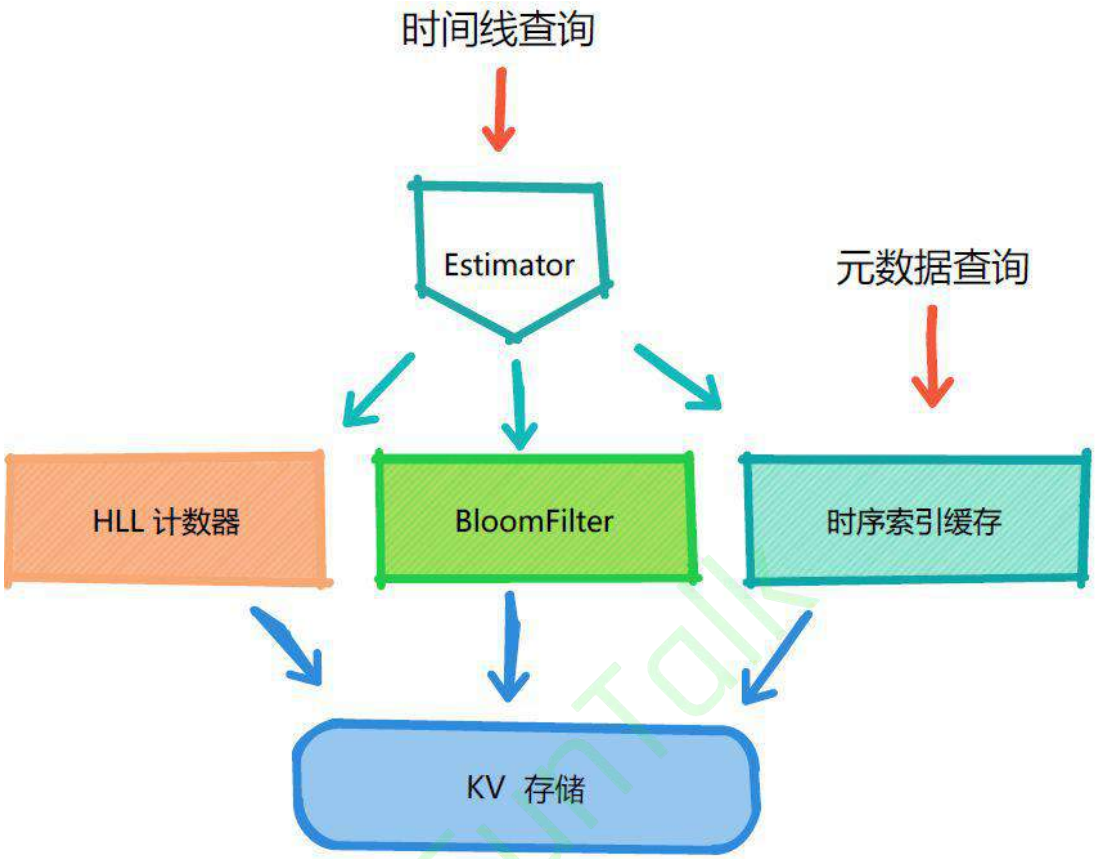
时序索引是什么？

本质是一个带时间戳的倒排索引，对数据进行倒排索引，然后在加上时间戳，这样当用户的请求过来时就可以根据用户的筛选条件获取数据的位置，然后再用时间线和时间戳进行二次过滤，这样提高了索引的命中率，同时也支持对时间线的 TTL。

如何存储？

时序索引是基于 kv 进行存储，是一个无状态的节点，可以支持水平扩展。

3.3.3 时序索引优化器



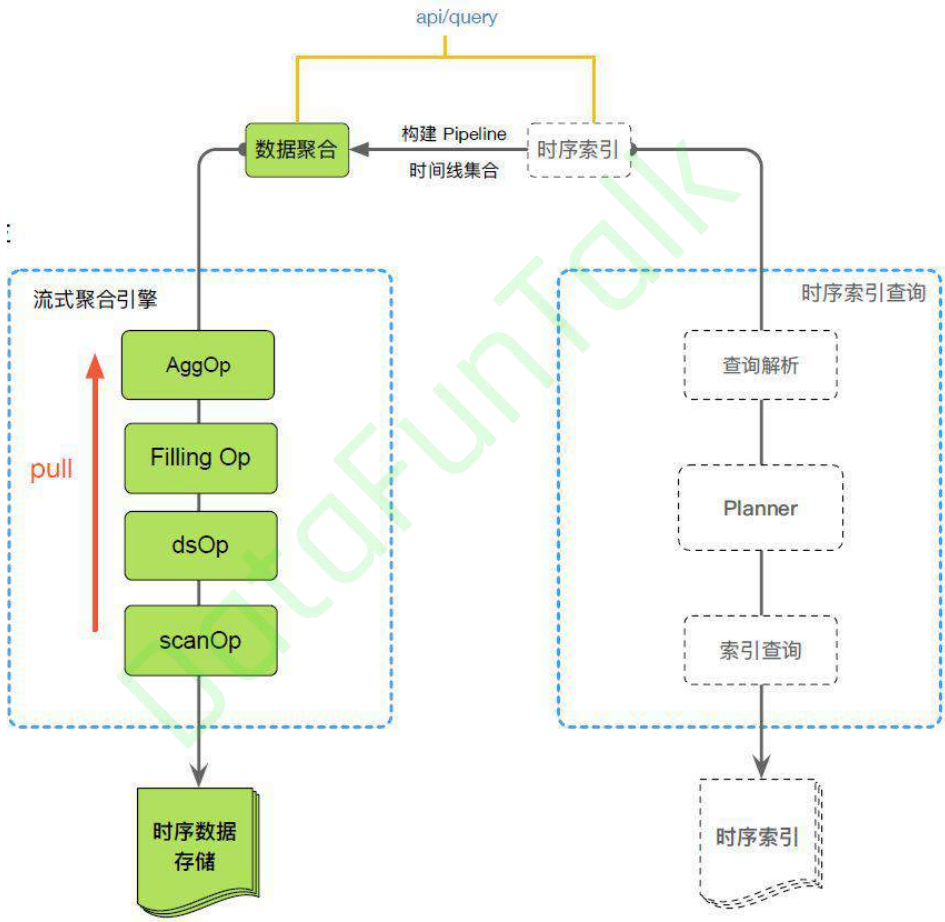
时序索引优化器主要是为了帮助用户在查询倒排索引的时候提高效率，这里优化器会对于用户的请求有三个处理：

- HLL 计数器：根据用户的查询规则匹配历史的时间线数量,或者某个 tag 下的时间线数量；
- BloomFilter：根据布隆过滤器判断某个时间线是否存在；
- 时序索引缓存：直接查询缓存中是否已有相同的时间线内容。

优化器对于用户查询如何评估优化？

首先会根据用户的查询条件选取最小的集合进行计算，然后会判断查询的时间线是否存在，如果不存在直接返回，一些明确限定的条件优先于模糊的条件，例如等于肯定优先于包含。

3.3.4 流式聚合引擎



流式聚合引擎采用轻量单进程实现，在用户查询的时候将用户的复杂查询转换为聚合算子组合，流式聚合引擎包含了 10+核心的聚合算子、20+填充策略、10+插值算法。

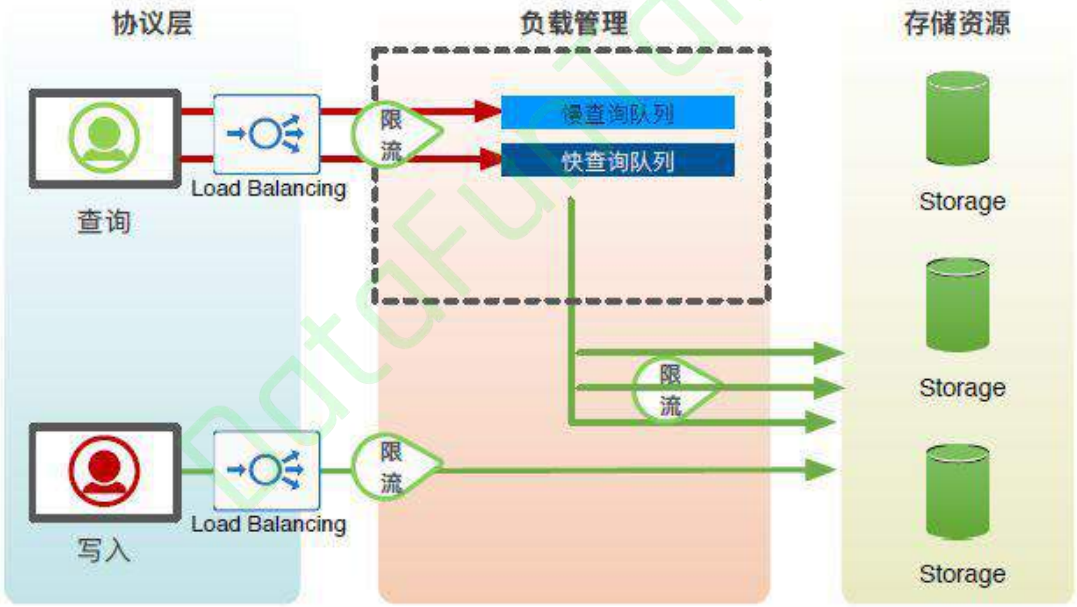
由于是将复杂的查询转换为算子组合运算，所以在实现上是一个松耦合的结构，扩展性很强而且每个算子的执行都非常高效,快速，减少了对内存的开销以及底层存储的压力，

另外聚合出和数据还可以与外部预聚合、降精度聚合的数据实现无缝衔接，提高了查询结果的复用率。

3.4 稳定性保障

3.4.1 TSBD 稳定保障机制

TSDB 不仅服务于专有云，更服务于公有云上的客户，因此稳定性是一个可以和内核提到同等层次的问题，TSDB 从三个方面实现了稳定性的保障。



资源隔离

读写线程分离：保障了查询有故障时不会影响写入，写入有故障时不会影响查询；

慢查询、大查询的隔离：根据用户的查询条件生成一个指纹，在根据历史记录判断这个指纹是不是一个慢查询或者大查询,如果是的话会将这个查询放到一个单独的队列 ,这个

队列的资源是受限的，而正常的查询会进入到正常的队列，这样一定程度上加速了整体查询的速度；

查询状态管理和调度：查询任务的状态管理和监控与任务的调度分离，这样降低了任务管理和调度的耦合性，提高了任务运行的稳定性。

基于时间线、时序数据的细粒度流控

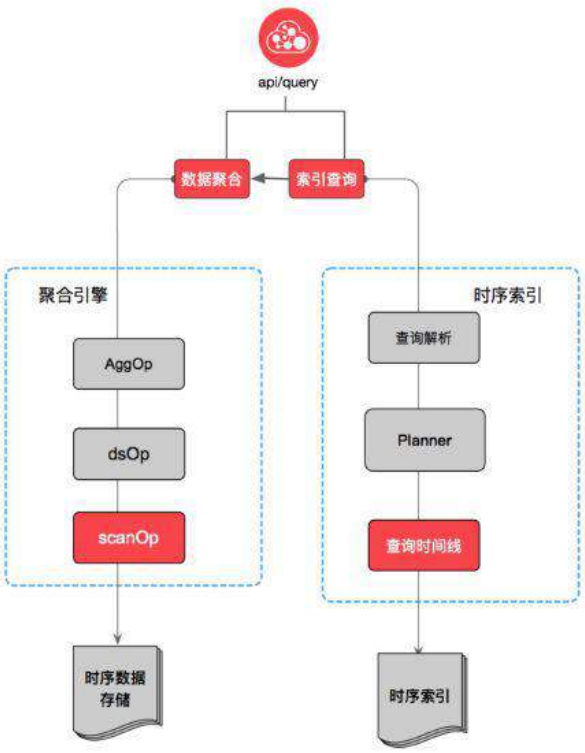
每一份数据都通过倒排索引+时间戳的索引方式定位，保证用户的查询条件能更粒度的命中数据，减少不必须的资源消耗。

全面的监控指标

TSDB 对整体的吞吐量、查询响应时间、IO 层关键指标、以及各个核心模块都有详细而全面的监控,保证能实时的了解到 TSDB 内部发生了什么问题，继而快速的定位解决问题。

3.4.2 TSDB 工作负载管理

DataFunTalk



端到端的流控：在用户的写入接口会做资源的 IO 控制，两个核心模块时序索引、聚合引擎的入口层也会做一个 IO 流控，保证党用户的写入量或多或者查询量过大时，两个核心模块不收影响；

多维度的控制：从不同维度对 TSDB 的读写进行一些限制，例如查询的时间线过长、访问的数据点过多，读取的字节数过大、整个查询消耗的时间过长等等。

▪ 总结展望

前面已经介绍了 TSDB 的发展历程和解决的问题，下面将要介绍 TSDB 未来的发展方向及特性：

1. 冷热数据异构存储

DataFunTalk

随着时间的推移数据量会越来越庞大，为了降低用户成本一些不必要或者低热度的数据不需要按照现有的数据存储方式来保存，可以换成一种更低成本的存储方式。

2. 提高 Serverless 读写能力

- 提供能高频率、低延迟的查询
- OLAP 系统长时间高纬度的分析
- 对于历史数据分析或者冷数据的分析
- 降低计算和查询的成本

3. 拥抱时序生态

将已有的时序引擎和计算引擎与业界很多成熟的时序生态，比如 prometheus、kubernetes、openTSDB 等结合，为用户体用更好的解决方案。

4. 时序智能分析

为用户提供更多稳定、可靠的智能分析模型，深入行业内部了解一些用户的痛点，解决一些亟待解决的问题。

DataFunTalk

深入了解 TiDB SQL 优化器

嘉宾：张建 编辑：HohXil

导读：本次报告张老师主要从原理上带大家深入了解 TiDB SQL 优化器中的关键模块，比如应用一堆逻辑优化规则的逻辑优化部分，基于代价的物理优化部分，还有和代价估算密切相关的统计信息等。

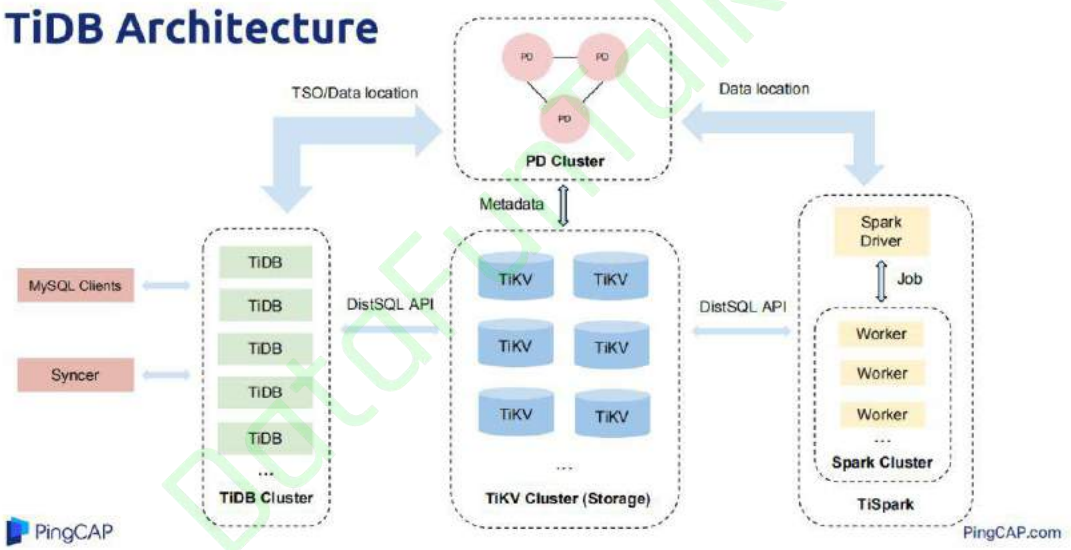
本文将从以下几个方面介绍：首先讲一下 TiDB 的整体架构，接下来就是优化器的两个比较重要的模块，一个是 SQL 优化，做执行计划生成；另一个模块就是统计信息模块，其作用是辅助执行计划生成，为每一个执行计划计算 cost 提供帮助。最后介绍下优化器还有哪些后续工作需要完成。

DataFunTalk

PingCAP PingCAP.com

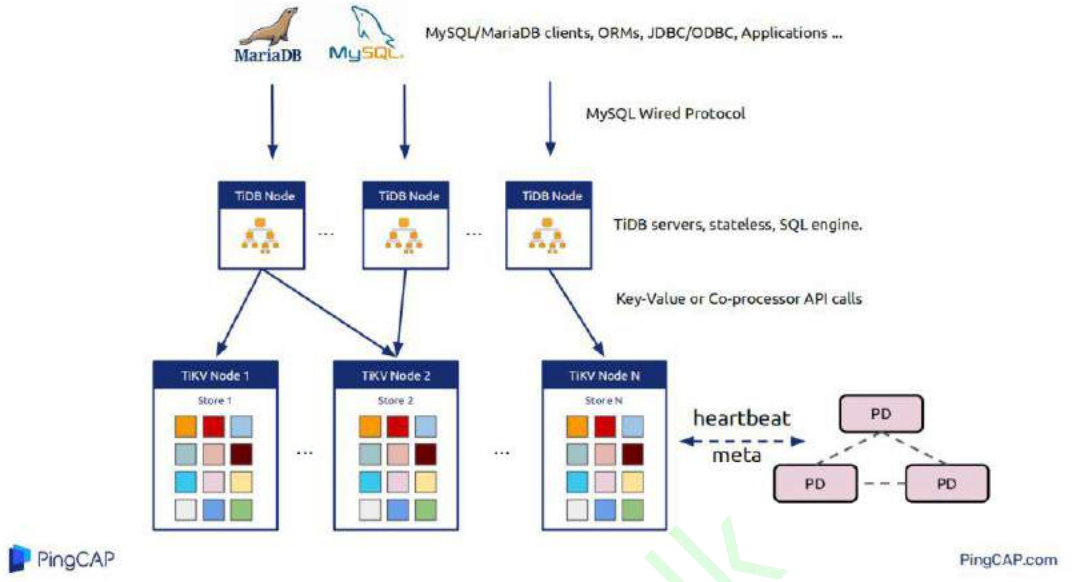
1. TiDB 的整体架构

TiDB 架构主要分为四个模块：TiDB、TiKV、TiSpark 和 PD，TiKV 是用来做数据存储，是一个带事务的、分布式的 key-value 存储，PD 集群是对原始数据里用来存储 key-value 里每一个范围的的 k-v 存储在每一个具体的 k-v 元数据信息，也会负责做一些热点调度；如热点 region 调度。在 Tikv 中做数据复制和分布式调度都是 rastgroup 做的，每一个读写请求都下放到 Tikv 的 leader 上去，可能会存在某些 Tikv 的 server 或者机器的 region leader 特别多，这个时候 PD 集群就会发挥热点调度功能，将一些热点 leader 调度到其他机器上去。TiDB 是所有场景中对接用户客户端的一层，也负责做 SQL 的优化，也支持所有 SQL 算子实现。Spark 集群是用来做重型 IP 的 SQL 或者作业查询，做一些分布式计算。

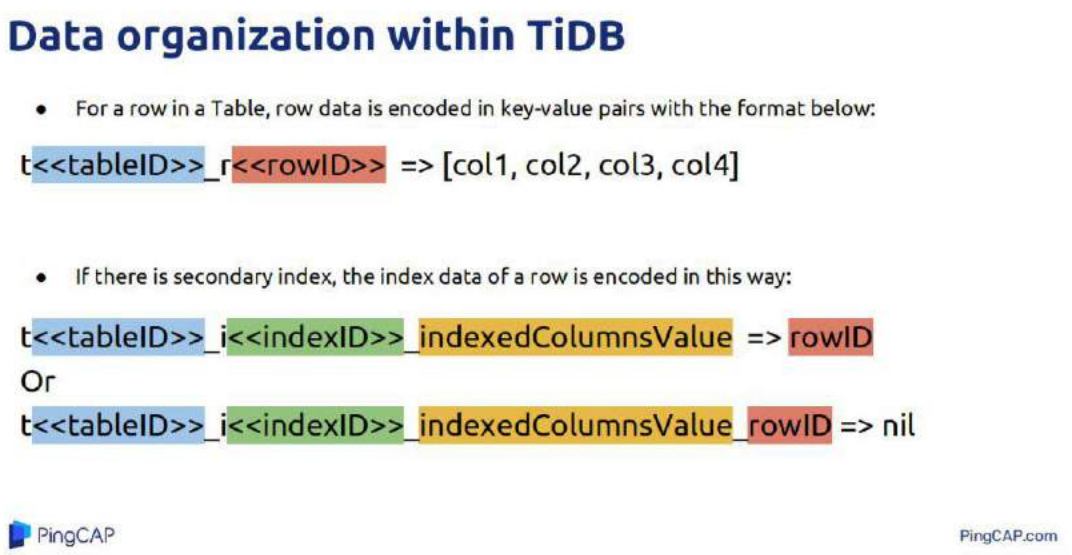


刨除 Spark，TiDB 集群主要有三个核心部分。最上层 TiDB 对接用户的各种 My SQL/Maria DB clients，ORMs，JDBC/ODBC，TiDB 的节点与节点之间本身是不做任何数据交互，是无状态的，其节点就是解析用户的 query，query 的执行计划生成。把一些执行计划下推到

一些 Tikv 节点，将一些数据从 Tikv 节点拿上来，然后在 PD 中做计算，这就是整个 TiDB 的概览。

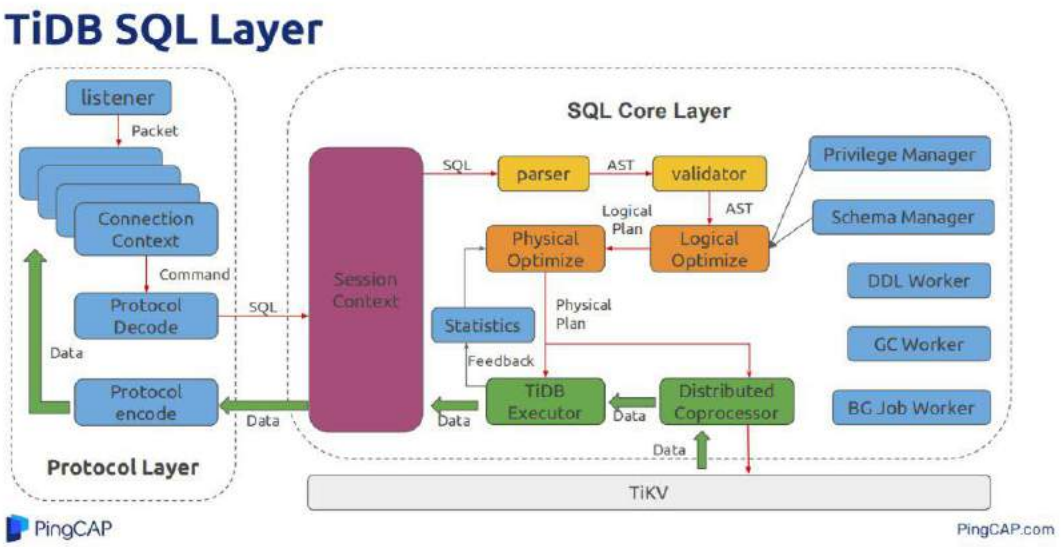


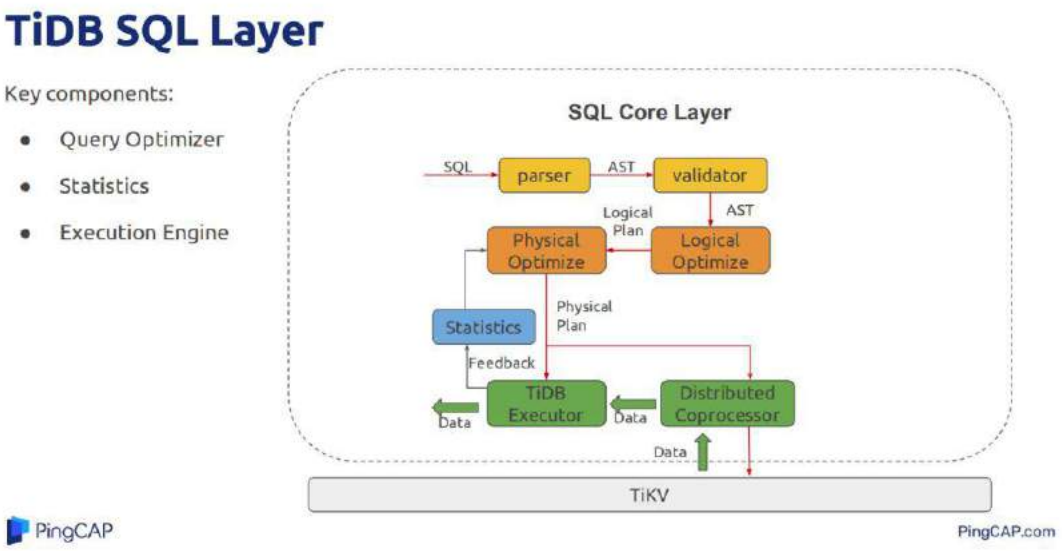
讲优化器之前需要讲一下 TiDB 中结构化的数据是如何映射到 K-V 数据的。在 TiDB 中只有两种数据，一种是表数据，一种是为表数据创建的 index 数据。表数据就是 tableID 加 RowID 的形式将其映射为 Key-Value 中的 key，表数据中具体每一行的数据一个 col 的映射为其 value，以 Key-Value 的形式存储到 Tikv 中。索引数据分为两种一种是唯一索引和非唯一索引，唯一索引就是 tableID+IndexID+索引的值构成 Key-Value 中的 key，唯一索引对应的那一行的 RowID，非唯一索引就是将 rowID encode 到 Key 中。



下面是 TiDB SQL 层的应用组件，左边是协议层，主要负责用户的 connect 连接，和 JDBC/ODBC 做一些数据协议，解析用户的 SQL，将处理好的结果数据以 MySQL 的形式 encode 成符合 MySQL 规范的格式化数据返回给客户端。中间的 Session Context 主要负责一个 session 里面需要处理的一些用户设置的各种变量，最右边就是各种权限管理的 manager、源信息管理、DDL Worker，还有 GC Worker 也是在 TiDB 层。

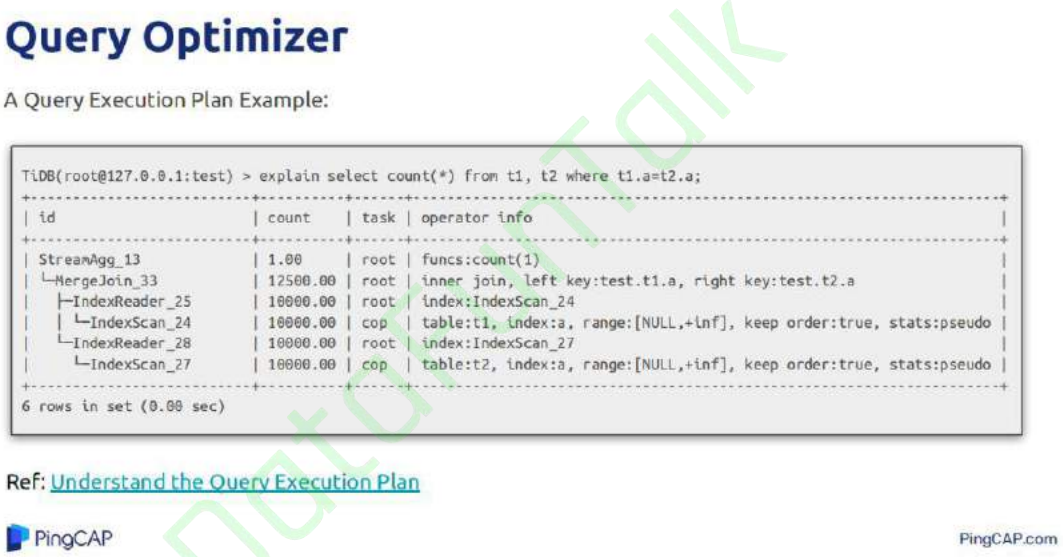
DataFunTalk





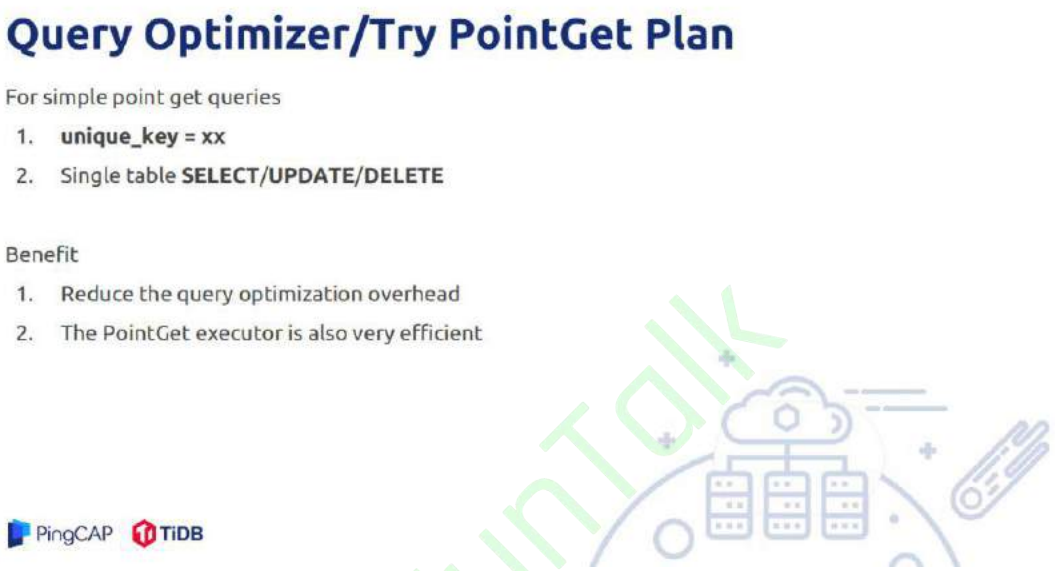
今天主要介绍 SQL 经过 parser 再经过 AST ,然后 Optimize ,经过 TiDB 的 SQL 执行引擎，还有经过 Tikv 提供的 Coprocessor，Coprocessor 支持简单的表达式计算、data scan、聚合等。Tikv 能让 TiDB 将一些大量操作都下推到 Tikv 上，减少 Tikv 与 TiDB 的数据交互带来的网络开销，也能让一部分计算在 Tikv 上分布式并行执行。

2. Query Optimizer



上图中的执行计划比较简单，就是两个表做 join，然后对 join 的结果做 count（*），join 方式是 merge join。

查询优化器解决的工作很复杂，比如需要考虑算子的下推，比如 filter 的下推，尽量下推到数据源，这样能减少所有执行数据的计算量；还有索引的选择，join Order 和 join 算法的选择，join Order 指的是当有多个表做 join 时以什么样的顺序去执行这些 join，不同的 join Order 意味着有不同大小的中间结果，而且 join Oder 也会去影响某一些 join 节点算法的选择；还有子查询的优化，如硬子查询是将其优化成 inner join 还是嵌套的方式去执行硬子查询而不去 join，这些在各种场景中因为数据源的分布不同，每一种策略都会在一种场景中有它自身的优势，需要考虑的方面很多，实现起来也比较困难。



优化器进行优化逻辑复杂，进行优化需要进行一些比较重的计算，为了降低一些不必要的计算。比如对一些简单的场景点查，根据一些组件查一条数据，这种就不需要经过特别复杂的计算，这种需要提前标记出来，直接将索引的唯一值 ID 解析出来，变成一次 k-v scan，这种就不需要做复杂的优化，不用去做执行树的迭代。目前 TiDB 中的 update、delete 、scan 都支持 k-v scan，还有 PointGet Plan 也支持这种优化。

TiDB 的 SQL 优化器分为物理优化阶段和逻辑优化阶段，逻辑优化阶段的输入是一个逻辑优化执行计划，有了初始逻辑优化执行计划后，TiDB 的逻辑优化过程需要把这个逻辑执行计划去应用一些 rule，每一个 rule 必须具备的特点是输出的逻辑执行计划与输出的逻辑执行计划在逻辑上是等价的。逻辑优化与物理优化的区别是逻辑优化区别数据的形态是什么，是先 join 再聚合还是先聚合再 join，它并不会去聚合算子是 stream 聚合还好 hash 聚合，也不会去关注 join 算子是哪一种物理算子。同时也要求 rule 将产生的每一个新的逻辑执行计

划一定要比原来输入的逻辑执行计划要更优 ,如将一些算子下推到数据源比不下推下去要更优，下推后上层处理的数据量变少，整体计算量就比原来少。

接下来就讲一下 TiDB 中已经实现的一些逻辑优化规则 ,如 Column Pruning 就是裁减掉一些不需要的列 ,Partition Pruning 针对的是分区表 ,可以依据一些谓词扫描去掉 ,Group By Elimination 指的是聚合时 Group By 的列是表的唯一索引时可以不用聚合。Project Emination 是消除一些中间的没用的一些投影操作，产生的原因是在一些优化规则以自己实现简单会加一些 Project ，还有就是从 AST 构造到最初逻辑执行计划时也会为了实现上的简单会去添加一些中间节点的投影操作，Outer Join Simplification 主要针对 null objective，如 A>10，而 A 有又是 null 而且又是 inner 表中的列时，Outer Join 就可以转化为 inner join。

Query Optimizer/Max Min Elimination

Convert a single Max to **descending ordered scan**

```
TiDB(root@127.0.0.1:test) > desc select max(a) from t;
+--------------------------+-------+------+-----------------------------------------------------------------------------+
| id                       | count | task | operator info                                                               |
+--------------------------+-------+------+-----------------------------------------------------------------------------+
| StreamAgg_13             | 1.00  | root | funcs:max(test.t.a)                                                         |
| └─Limit_17               | 1.00  | root | offset:0, count:1                                                           |
|   └─IndexReader_34       | 1.00  | root | index:Limit_33                                                              |
|     └─Limit_33           | 1.00  | cop  | offset:0, count:1                                                           |
|       └─IndexScan_32     | 1.00  | cop  | table:t, index:a, range:[-inf,+inf], keep order:true, desc, stats:pseudo    |
+--------------------------+-------+------+-----------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

PingCAP TiDB

Query Optimizer/Outer Join Elimination

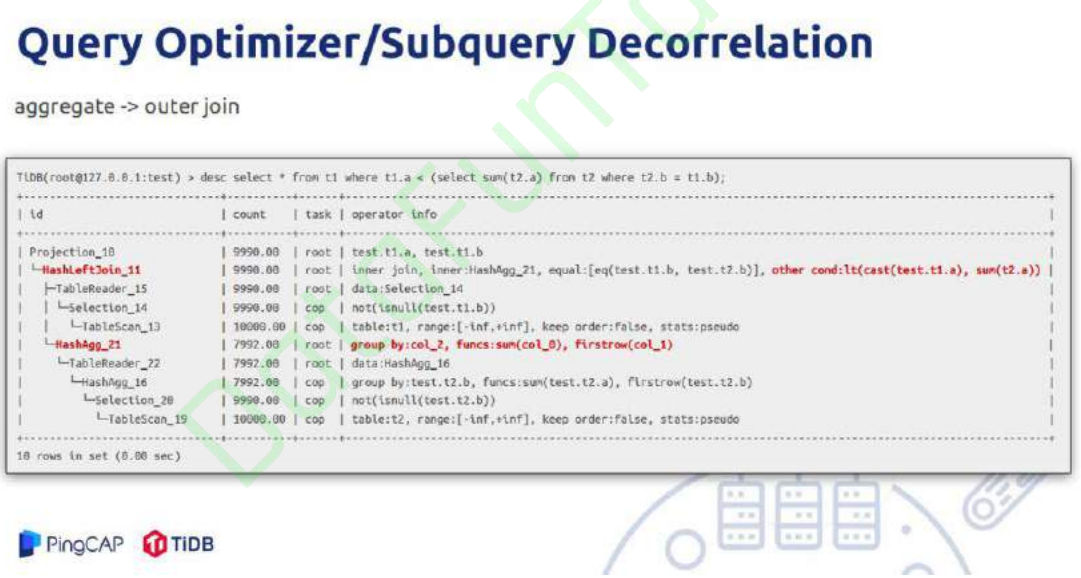
- Parent operator only needs outer columns
- Join Key on inner side is unique

```
select t1.* from t1 left join t2 on t1.a=t2.unique_key

select * from t1;
```

PingCAP TiDB

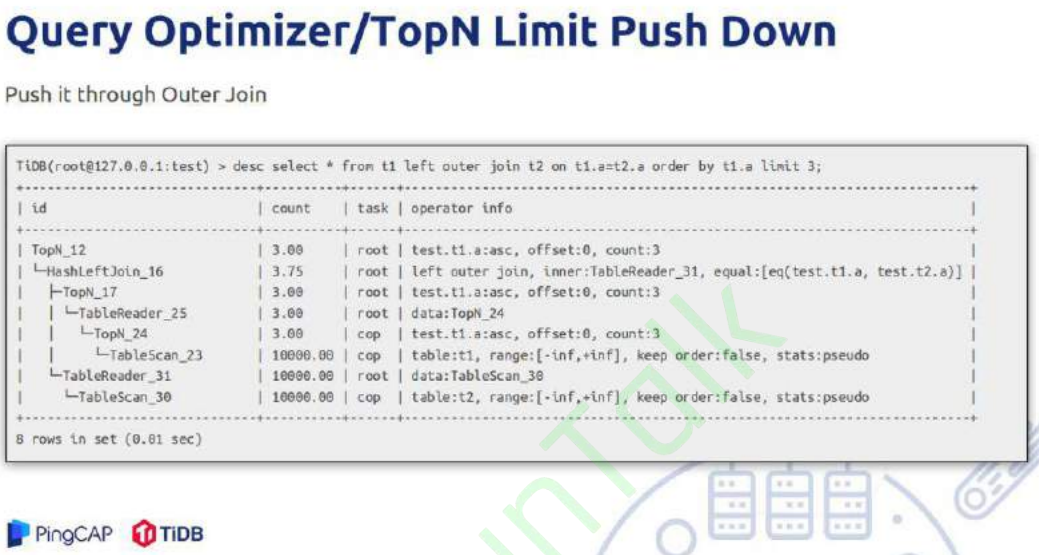
Max/Min Eliminatation 在有索引的时候非常有用，如 Max A 是一个索引列，直接在 A 上做一个逆序扫第一行数据就可以对外返回结果，顶层还有一个 Max A，这个是为了处理 join 异常情况，如 Max 和 count 对空输入结果值行为结果是不一样的，需要有一个顶层的聚合函数来处理异常情况，这样就不需要对所有数据做 max，这样做的好处就是不用做全表扫描。

Outer Join Elimination 可以将其转化为只扫描 Outer 表，比如当用户只需要使用 Outer Join 的 Outer 表，如例子中只需要 t1 表中的数据，如何 inner 表上的 key 刚好是 inner 表上的索引，那么这个 inner 表就可以扔掉，因为对于 outer 表中的每一条数据如果能 join 上，只会和 inner 表的一行数据 join 上，因为 inner 表上的 key 是唯一值，如果对应不上就是 null，而返回的数据只需要 outer 表，inner 表上的数据不需要。还有一种情况是父节点只需要 outer 表的唯一值，再做 outer join 如果对应上会膨胀很多值，而上层只需要不同值这样就不需要膨胀，这样就可以消除在 outer 表做一个 select 的 distinct 操作。



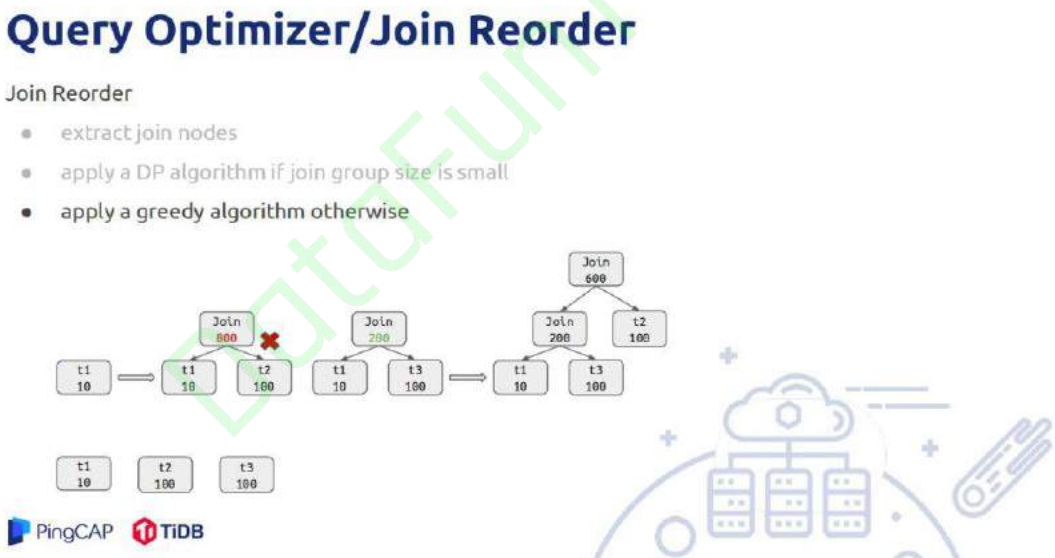
Subquery Decorrelation 是一个多年研究的问题，上图例子是先从 t1 表中扫一行数据，去构造 t2 表的 filter，然后去扫描 t2 表中满足这样的数据，对 t2 表的 A 做一个聚合，最终是 t1 表的 A 类数据小于求的和，才把 t1 表的这行数据输出。如果执行计划按照上述逻辑执行，那么每一行 t1 的值都会对 t2 进行全表扫描，这样就会对集群产生非常大的负担，也会做很多无用的计算。因此可以将优化成先聚合再 join，就是先把 t2 表先按过滤的条件的

列做一个 group by，每一个 group 求 t2 表 A 的和，将其求得的和再去和 t1 表做 join。上层的 arcconditon，这样就不会对 inner 表频繁的做 inner 操作，从整体上看不用做全表扫描，每一行 outer 都会对 t2 表做扫描。



聚合下推不一定要优，但在某些场景很有用。两个表做 join，以上面一个表为例，join 的结果以 t1 的 a 做一个 group by。如果 t1 表的 t1.a 列重复的值很多，先去做 join 就会导致重复的值和 t2 表能够匹配的值重复很厉害，再去做聚合计算量也非常大，有一种策略是将聚合下推到 t1 表上。将 t1 表上 a 做一个聚合，很多重复的 t1.a 再 join 之间就压缩成一条，join 操作的计算量非常轻，在更上层的聚合相应减轻不少负担。但是不一定每种情况都有用，如果 t1.a 中的数据重复值不多，那么下推下去的聚合将数据过滤一遍又没有起到聚合的效果。Top N Limit Push Down 只需要将其 outer join push 到 outer 端，这是因为 outer 表的数据要输出，只需要拿三条数据和 inner 表做 join，如果有膨胀，再放一个

top/limit 将数据只限制在三条。相反如果将 topN 不 push 下去，那么从 table3 读取的数据会很大。



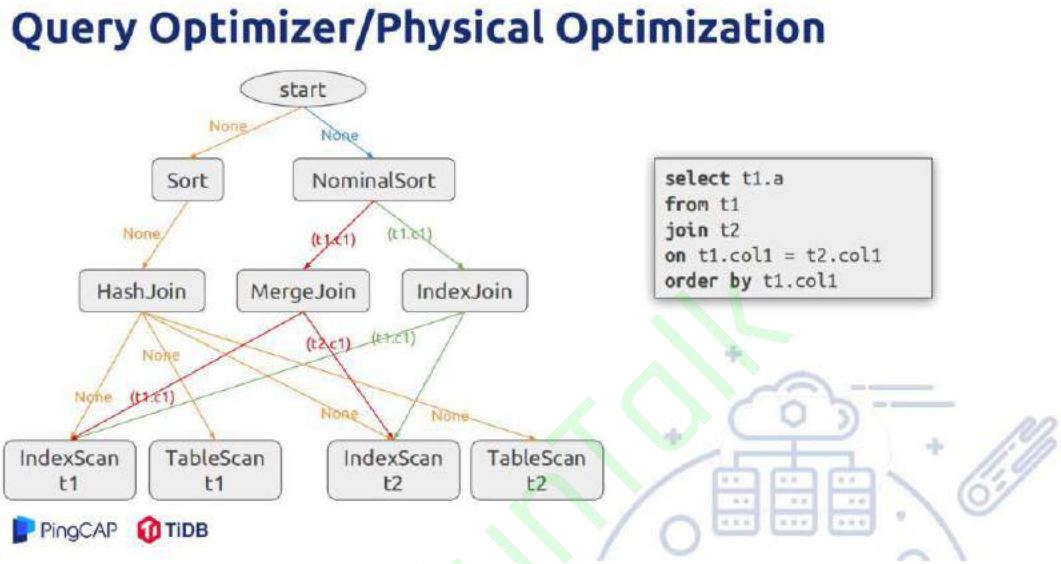
还有一个难题是 Join Reorder，目前 Join Reorder 的算法有很多。统计信息精准度一定的情况下，选出一个最好的 Join Reorder 算法最好的方式是用 DP 算法。如果两者信息精确，

利用动态规划得出的算法一定是最优的，但是现实中统计信息不一定优，如两张表信息是优但是 join 后的结果不一定符合数据真实分布，可能有推导误差。A、B 统计节点是推导出来的，再去推导节点的统计信息，误差就被放大，因此 DP 的 join order 在使用真实的统计信息做 join order 再去推导统计节点的统计信息所做出来的 order 也不一定是好的。

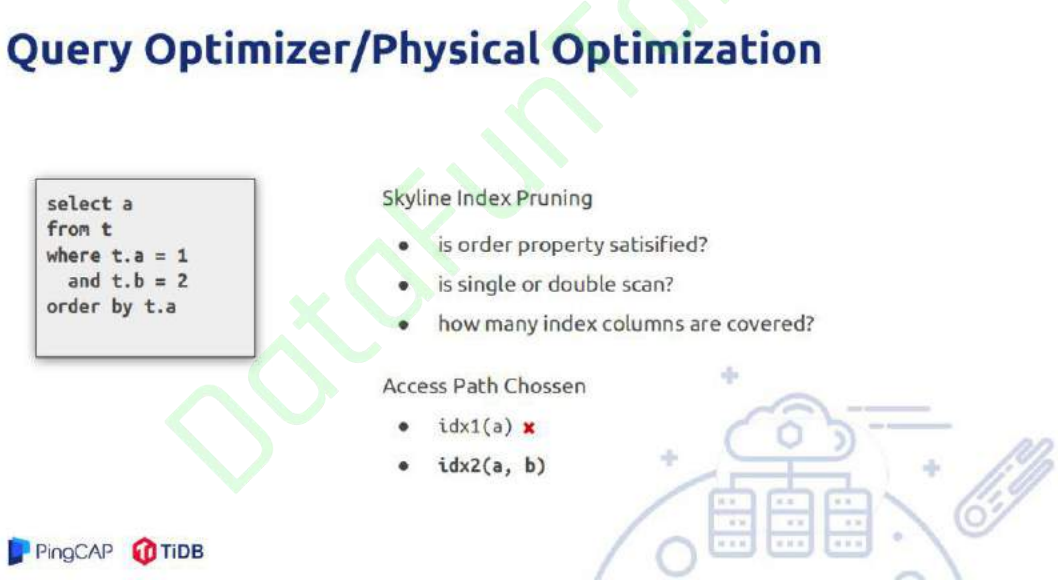
在 TiDB 中使用的 join order 是一个子树，使用状态压缩的方式做的，就是 6 的整数用二进制的形式表示 110， 0 表示节点不存在，1 表示节点存在，第 1、2 节点存在，第 0 号节点不存在。就决定了最优的 join 顺序是什么，这样 DP 算法推导就比较简单，不断的枚举其子集合，6 可以分为 110 和 10，分别 join 两个子集合，选择所有情况中最小的一个；这种方式时间复杂度很高，如果节点过多，做 join reorder 的时间会很长。还有 DP 算法是用整数代替 join 节点，如果 10 个节点就是 2^{10}，20 个节点就是 1M 内存。因此当节点比较大的时候采用贪心策略做 join reorder，实现原理是先将所有的 join recount 估算，从小打到大排序，一次选择按边相连的节点去做 join，如图一开始初始是 t1 和 t2 做 join 结果估算有 800，由于 t3 的 count 也是 100，也需要考虑 t1 和 t3 做 join，join 出来是 200，则 t1 和 t3 优先做 join，然后再遍历节点数后最小的节点与当前 join 数做 join，当为 join 节点集合为空时整个 join 树就生成了。但是局部最优不一定全局最优，并不能把所有情况都考虑最好的 join 顺序。

接下来是物理优化阶段，逻辑优化并不决定以什么算法去执行，只介绍了 join 顺序，并没有说要用那种 join 方式。物理优化需要考虑不同的节点，不同的算法对输入输出有不同的要求，如 hash 和 merge join 实现的时间复杂度本身不一样。要理解物理优化的过程要理解什么是物理属性。物理属性是一个物理算法所具备的属性，在 TiDB 就有 task type 属性，就是这个算法是应该在 TiDB 中执行还是在 Tikv 中执行；data order 说的是算法所产生的数据应该以什么样的顺序属性，如 merge join 是按 outer join 的 key 有序的。Stream 聚合也是按照 group by 的 column 有序。但是有些算法无法提供 join 顺序，如 hash join，还会破坏数据的顺序，hash join 无法对外提供任何顺序上的保证。在分布式场景中做执行计划时需要考虑分布的属性，如 hash join 在一个分式的节点上执行，考虑的是选表多下搜的方式，如果想正确出结果最好的方式是将小表和大表的数据都按照 join 的 key 下放到不同的机器上，那么分布式的 hash join 特点就是 join 的 key 分布在同一台机器上。在 TiDB 没有考虑数据分布的特性，动态规划的状态就是输入的逻辑状态是什么，实现的逻辑执行计划的物理执行计划需要满足什么样的物理属性，最后推导出一个最佳的物理执行计划。这样

同一个逻辑节点可能会多次被父节点以不同路劲访问它，因此需要缓存中间节点，下次父节点以同样的动态规划状态访问直接将之前最佳的结果返回就行。



上图的实例是对两个表做 join，join 后数据按照 join key 排序，假设 t1 和 t2 表都在各自的 join key 上有索引，对于 t1 和 t2 表扫描有两种方式，一种是 index scan 能够满足返回的数据以 index 有序，或者 table scan 不能满足 index scan 有序，nominalsort 是 TiDB 内部优化算子，既不会出现在逻辑执行计划里面也不会出现物理执行计划里面，只是在做物理执行计划辅助作用 ,从一开始调用动态规划过程 ,输入逻辑计划要求满足的物理属性是空，接下来可以用物理 sort 算子和 nominalsort 算子，其本身不 排数据，而是将排数据的功能传递给子节点。



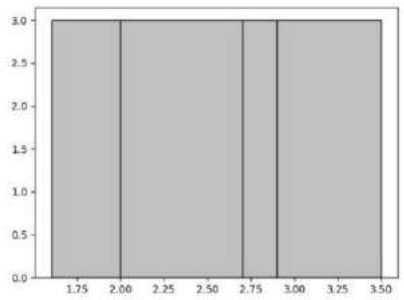
在物理优化中比较重要的一点是如何选择索引，没有索引一个慢查询会导致所有集群都慢。最后引入 Skyline index Pruning，当要选择那个选项最优时有多个维度可以考量，访问一个表的方式有多种方式选择，其要求就是父节点要求子节点返回的数据是否有序，还有就是索引能够覆盖多少列，这是因为用户建索引并不是一定按照最优解来建。从优化过程来说，算法并不是最优的，应用完一个 rule 不会再次去应用，但是实际是会多次使用的。解决有 Memo 优化，就是将所有表达式存储，将等价表达式存储于一个 group 里面，将所有 rule 用最小化、原子化做 group expression。

3. Statistics

Statistics

What kinds of statistics does TiDB need?

- Equi-depth Histogram

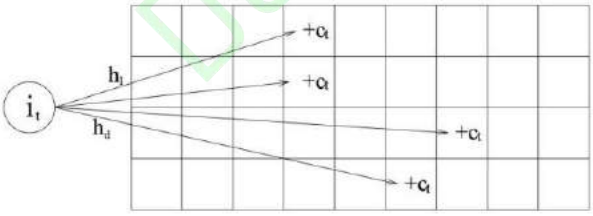


PingCAP TiDB

Statistics

What kinds of statistics does TiDB need?

- Equi-depth Histogram
- Count-Min Sketch



PingCAP TiDB

统计信息是用来估算 row count，需要估算的 row count 有 filter、join、聚合。TIDB 中存储的统计信息有直方图，主要用于估算范围查询的统计信息，被覆盖的其 count 直接加上去，部分覆盖的桶使用连续均匀分布的假设，被覆盖的部分乘以桶的 rowcount 加上去；另一个是估算点查询的 rowcount，可以理解 Min-Sketch，只是估算的值不再是 0 和 1，数据代表是这个位置被 hash 到了多少次，如一个数据有 D 个 hash 函数，将其 hash 到 D

的某个位置 ,对具体位置加上 1 ,查询也做同样的操作 ,最后取这 D 位置最小的值作为 count 估计，这个估计在实际中精度较高。

TiDB 收集统计信息的方式有很多，首先手动执行 analyze 语句做统计信息的搜集；也可以配置自动 analyze ,就是表的更新超过某些行数会自动做 analyze ;还有 Query Feedback，就是在查询请求 ,如果查的数据分布和以前统计的数据分布信息不太匹配回去纠正已有的统计信息。

4. Future Work

- Robust Query Processing
- The Cascades Planner
- Fast Analyze
- Multi-Column Statistics

接下来一些工作就是查询计划的稳定性，重要的是索引的准确，还有就是有些算法的选择也会影响查询计划的稳定性；The Cascades Planner 就是要解决搜索空间的搜索算法的效率问题，搜索空间导致执行计划不够优的问题。还有快孙 analyze，目前表以亿起步，如果现场采样，会比较慢因此会采取一些手段加速 analyze 过程。Multi-Column Statistics 主要生死用来解决多列之间的相关性，以前做 row count 估算都是基于 column 与 column 间的不相关假设做 row count，这样估计的值比实际值偏大，有多列相关估算准确度会提高很多。

Druid SQL 和 Security 在美团点评的实践

嘉宾：高大月　编辑：HohXil

导读：长久以来，对 SQL 和权限的支持一直是 Druid 的软肋。虽然社区早在 0.9 和 0.12 版本就分别添加了对 SQL 和 Security 的支持，但根据我们了解，考虑到功能的成熟度和稳定性，真正把 SQL 和 Security 用起来的用户是比较少的。本次分享将介绍社区 SQL 和 Security 方案的原理，以及美团点评在落地这两个功能的过程中所遇到的问题、做出的改进、和最终取得的效果。下面开始今天的分享：

我今天的分享内容包括四部分。首先，和大家介绍一下美团对 Druid 的使用现状，以及我们在构建 Druid 平台的过程中遇到的挑战。第二部分，介绍 Druid SQL 的基本原理和使用方式，以及我们在使用 Druid SQL 的过程中遇到的问题和做的一些改进。第三部分，介绍 Druid 在数据安全上提供的支持，以及我们结合自身业务需求在 Druid Security 上的实践经验。最后，对今天的分享内容做一个总结。

一、Druid 在美团的现状和挑战

1.1 Druid 应用现状

美团从 16 年开始使用 Druid，集群版本从 0.8 发展到现在的 0.12 版本。线上有两个 Druid 集群， 总共大概有 70 多个数据节点。

数据规模上，目前有 500 多张表，100TB 的存储，最大的表每天从 Kafka 摄入的消息量在百亿级别。查询方面，每天的查询量有 1700 多万次，这里包括了一些程序定时发起的查询，比如风控场景中定时触发的多维查询。性能方面，不同的应用场景会有不同的要求，但整体上 TP99 响应时间在一秒内的表占了 80%，这和我们对 Druid 的定位——秒级实时 OLAP 引擎是一致的。

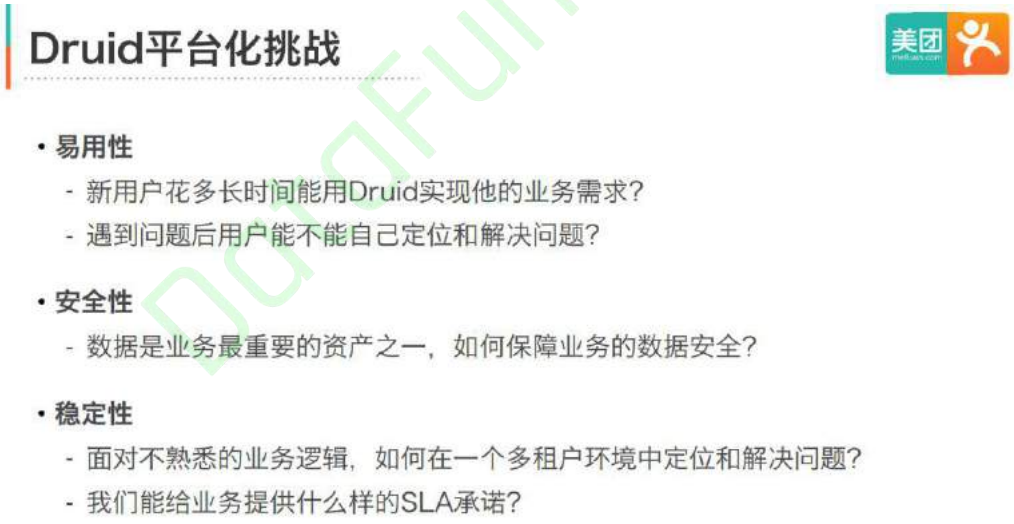
1.2 Druid 平台化挑战

把 Druid 作为一个服务提供给业务使用的过程中，我们主要遇到了易用性、安全性、稳定性三方面的挑战。

易用性：业务会关心 Druid 的学习和使用成本有多高，是否能很快接入。大家知道，Druid 本身对数据写入和查询只提供了基于 JSON 的 API 接口，你需要去学习接口的使用方法，了解各种字段的含义，使用成本是很高的。这是我们希望通过平台化去解决的问题。

安全性：数据是很多业务的核心资产之一，业务非常关心 Druid 服务能否保障他们的数据安全。Druid 较早的版本对安全的支持较弱，因此这一块也是我们去年重点建设的部分。

稳定性：一方面需要解决开源系统落地过程中出现的各种稳定性问题，另一方面，如何在查询逻辑不可控的情况下，在一个多租户的环境中定位和解决问题，也是很大的挑战。



二、Druid SQL 的应用和改进

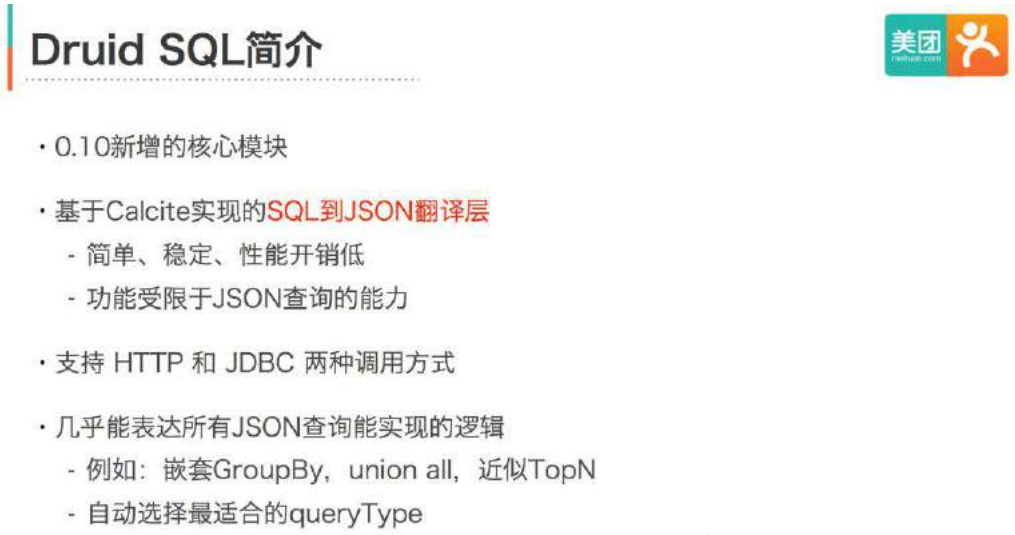
在 Druid SQL 出现之前，Druid 查询通过基于 JSON 的 DSL 来表达（下图）。这种查询语言首先学习成本很高，用户需要知道 Druid 提供了哪些 queryType，每种 queryType 需要传哪些参数，如何选择合适的 queryType 等。其次是使用成本高，应用需要实现 JSON 请求的生成逻辑和响应 JSON 的解析逻辑。

通过 Druid SQL，你可以将上面的复杂 JSON 写成下面的标准 SQL。SQL 带来的便利是显而易见的，一方面对于程序员和数据分析师没有额外的学习成本，另一方面可以使用类似 JDBC 的标准接口，大大降低了门槛。

```
select floor(__time to MINUTE),
       sum(added) + sum(deleted)
  from wikiticker
 where __time >= '2015-09-12 00:00:00' and __time < '2015-09-13 00:00:00'
   and countryName='China'
   and channel='#en.wikipedia'
 group by floor(__time to MINUTE)
```

2.1 Druid SQL 简介

下面我简单地介绍一下 Druid SQL。



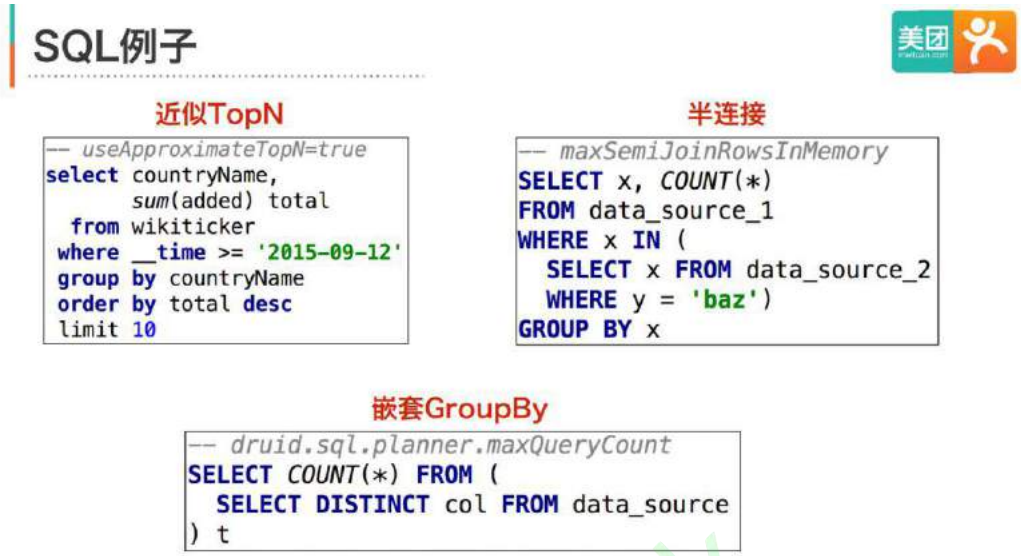
首先，Druid SQL 是 0.10 版本新增的一个核心模块，由 Druid 社区提供持续的支持和优化，因此不管是稳定性还是完善性，都会比其他给 Druid 添加 SQL 方言的项目更好。

DataFunTalk

从原理上看 ,Druid SQL 主要实现了从 SQL 到原生 JSON 查询语言的翻译层。由于只是做了一层语言的翻译，好处是 Druid SQL 对集群的稳定性和性能不会有很大影响，缺点是受限于原生 JSON 查询的能力，Druid SQL 只实现了 SQL 功能的一个子集。

调用方式上，Druid SQL 提供了 HTTP 和 JDBC 两种方式来满足不同应用的需求。最后表达力上，Druid SQL 几乎能表达所有 JSON 查询能实现的逻辑，并且它能自动帮你选择最合适的 queryType。

下面是三个 Druid SQL 的例子。



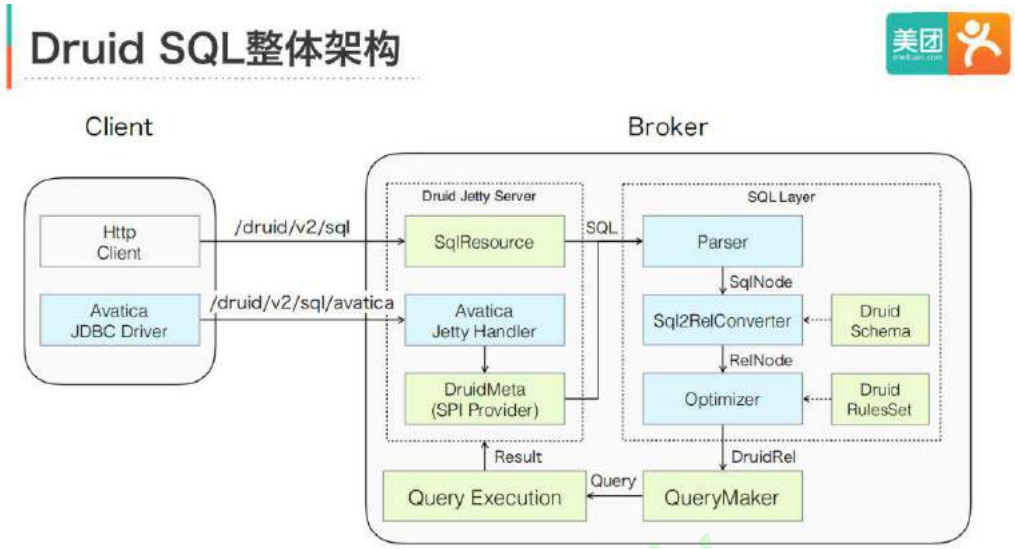
第一个例子是近似 TopN 查询。对于根据某个指标分析单个维度 TopN 值的需求，原生的 JSON 查询提供了一种近似 TopN 算法的实现。Druid SQL 能够识别出这种模式，生成对应的近似 TopN 查询。

DataFunTalk

第二个例子是半连接。我们知道 Druid 是不支持灵活 JOIN 的，但业务经常会有这样的需求，就是以第一个查询的结果作为第二个查询的过滤条件，用 SQL 表达的话就是 in subquery，或者半连接。Druid SQL 对这种场景做了特殊支持，用户不需要在应用层发起多个查询，而是写成 in subquery 的形式就行了。Druid SQL 会先执行子查询，将结果物化成外层查询过滤条件，然后再执行外层的查询。

最后是一个嵌套 GroupBy 的例子。Druid SQL 能够识别出这种多层的 GroupBy 结构，生成对应的原生嵌套 GroupBy JSON 。

2.2 Druid SQL 架构



下面介绍 Druid SQL 的整体架构。

DataFunTalk

Druid SQL 是在查询代理节点 Broker 中实现的功能 ,主要包含 Server 和 SQL Layer 两个模块。

Server 模块负责接收和解析请求，包括 HTTP 和 JDBC 两类 。对于普通的 HTTP 请求，新增相应的 REST Endpoint 即可。对于 JDBC，Druid 复用了 Avatica 项目的 JDBC Driver 和 RPC 定义，因此只需要实现 Avatica 的 SPI 就行了。由于 Avatica 的 RPC 也是基于 HTTP 的，因此两者可以使用同一个 Jetty Server。

SQL Layer 负责将 SQL 翻译成原生的 JSON 查询，是基于 Calcite 项目实现的。Calcite 是一个通用的 SQL 优化器框架，能够将标准 SQL 解析、分析、优化成具体的执行计划，在大数据领域得到了广泛的使用。图中浅绿色的组件是 Calcite 提供的，浅蓝色的组件是 Druid 实现的，主要包括三个。

首先，DruidSchema 组件为 Calcite 提供查询解析和验证需要的元数据，例如集群中包含哪些表，每张表各个字段的名称和类型等信息。RulesSet 组件定义了优化器使用的转换规则。由于 Druid SQL 只做语言翻译，因此这里都是一些逻辑优化规则（例如投影消除、常量折叠等），不包含物理优化。通过 RulesSet，Calcite 会将逻辑计划转成 DruidRel 节点，DruidRel 包含了查询的所有信息。最后，QueryMaker 组件会尝试将 DruidRel 转成一个或多个原生 JSON 查询，这些 JSON 查询最终提交到 Druid 的 QueryExecution 模块执行。

2.3 API 选择: HTTP or JDBC

Druid SQL 提供了 HTTP 和 JDBC 两种接口，我应该用哪个？我们的经验是，HTTP 适用于所有编程语言，Broker 无状态，运维较简单；缺点是客户端处理逻辑相对较多。JDBC 对于 Java 应用更友好，但是会导致 Broker 变成有状态节点，这点在做复杂均衡时需要格外注意。另外 JDBC 还有一些没有解决的 BUG，如果你使用 JDBC 接口，需要额外关注。

2.4 改进

下面介绍我们对 Druid SQL 做的一些改进。

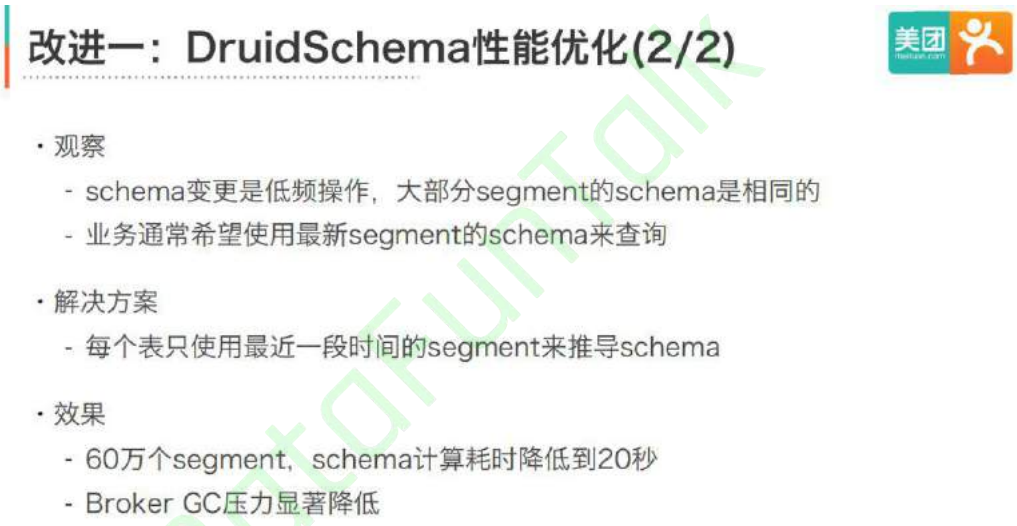
第一个改进是关于 Schema 推导的性能优化。我们知道 Druid 是一个 schema-less 系统，它不要求所有的数据的 schema 相同，那如何定义 Druid 表的 schema 呢？社区的实现方式是：先通过 SegmentMetadataQuery 计

算每个 segment 的 schema，然后合并 segment schema 得到表的 segment，最后在 segment 发生变化时重新计算整个表的 schema。

社区的实现在我们的场景下遇到了三个问题。第一是 Broker 启动时间过长。我们一个集群有 60 万个 segment，测试发现光计算这些 segment 的 schema 就需要半小时。这会导致 Broker 启动后，需要等半个小时才能提供服务。第二个问题是 Broker 需要在内存中缓存所有 Segment 的元数据，导致常驻内存增加，另外 schema 刷新会带来很大的 GC 压力。第三个问题是，社区方案提交的元数据查询量级与 Broker 和 Segment 个数的乘积成正比的，因此扩展性不好。

针对这个问题，我们分析业务需求和用法后发现：首先 schema 变更是一个相对低频的操作，也就是说大部分 segment 的 schema 是相同的，不需要去重复计算。另外，绝大数情况业务都只需要用最新的 schema 来查询。因此，我们的解决方案是，只使用最近一段时间，而不是所有的 segment 来推导

schema。改造后，broker 计算 schema 的时间从半小时降低到了 20 秒，GC 压力也显著降低了。



第二个优化是关于日志和监控。请求日志和监控指标是我们在运维过程中重度依赖的两个工具，比如慢查询的定位、SLA 指标的计算、流量回放测试等都依赖日志和监控。但是 0.12 版本的 SQL 既没有请求日志 ,也没有监控指标，这是在上线前必须要解决的问题。我们的目标有两个：首先能记录所有 SQL 请求的基本信息，例如请求时间、用户、SQL 内容，耗时等；其次能将 SQL 请求和原生的 JSON 查询关联起来。因为执行层面的指标都是 JSON 查询粒度的，我们需要找到 JSON 查询对应的原始 SQL 查询。

改进二：添加SQL的请求日志和监控指标(1/2)

- SQL查询带来的运维变化
  - 慢查询定位
  - SLA计算
  - 流量回放
- 目标
  - SQL和对应的原生查询能互相关联
  - 可扩展的SQL请求日志实现

我们的解决方案已经合并到 0.14 版本。首先，我们会给每个 SQL 请求分配唯一的 sqlQueryId。然后我们扩展了 RequestLogger 接口，添加了输出 SQL 日志的方法。下图是一个例子，对于每个 SQL 请求，除了输出 SQL 内容外，也会输出它的 sqlQueryId，可以用来与客户端的日志做关联。还会输出 SQL 对应的每个 JSON 查询的 queryId，可以用来和 JSON 查询做关联分析。

改进二：添加SQL的请求日志和监控指标(2/2)

- 解决方案 (#6302)
  - 为每个SQL请求分配唯一的sqlQueryId
  - 扩展了RequestLogger接口，支持输出SQL请求日志

```
{
    "query":"SELECT page, COUNT(*) AS Edits FROM wikiticker WHERE __time BETWEEN TIMESTAMP '2015-09-12
00:00:00' AND TIMESTAMP '2015-09-13 00:00:00' GROUP BY page ORDER BY Edits DESC LIMIT 10",
    "context":{
        "sqlQueryId":"c9d035a0-5ffd-4a79-a865-3ffdadbb5fdd",
        "nativeQueryIds":"[490978e4-f5c7-4cf6-b174-346e63cf8863]"
    }
}
```

  - SQL和JSON查询通过sqlQueryId和nativeQueryIds关联
  - 两个新metrics："sqlQuery/time" 和 "sqlQuery/bytes"

第三个改进虽然比较小，但是对服务的稳定性很重要。我们知道，JSON 查询要求用户指定查询的时间范围，Druid 会利用这个范围去做分区裁剪，这对提高性能非常重要。但是 Druid SQL 并没有这方面的限制。用户写 SQL 经常会忘记加时间范围的限定，从而导致全表扫描，占用大量的集群资源，是一个很大的风险。所以我们添加了对 where 条件的检查，如果用户没有指定时间戳字段的过滤条件，查询会直接报错。

三、Druid Security 的实践经验

首先介绍我们在数据安全上面临的问题。当时使用的是 0.10 版本，这个版本在数据安全上没有任何的支持，所有的 API 都没有访问控制，任何人都能访问甚至删除所有的数据，这对业务的数据安全来说是一个非常大的隐患。

我们希望实现的目标有五点：所有 API 都经过认证、实现 DB 粒度的权限控制、所有数据访问都有审计日志、业务能平滑升级到安全集群、对代码的改动侵入性小。

为了实现这些目标，我们首先调研了 Druid 在后续版本中新增的安全功能。

3.1 Druid Security 功能和原理

0.11 版本支持了端到端的传输层加密（TLS），能够实现客户端到集群，以及集群各个节点之间的传输层安全。0.12 版本引入了可扩展的认证和鉴权框架，并且基于这个框架，提供了 BA 和 Kerberos 等认证方式，以及一个基于角色的鉴权模块。

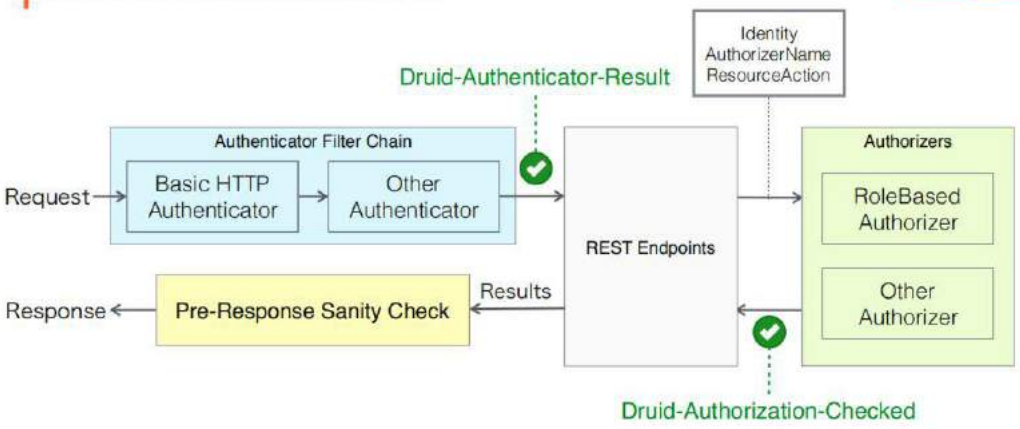
DataFunTalk

社区Security功能简介

- 端到端的传输层加密 (0.11)
- 可扩展的认证框架
  - Allow All (0.11)
  - Kerberos (0.11)
  - Basic Auth (0.12)
  - Anonymous (0.13)
- 可扩展的鉴权框架
  - Role-based Access Control扩展 (0.12)

下面这张图介绍认证鉴权框架的原理和配置。

社区认证鉴权框架



https://imply.io/post/securing-druid

3.2 Druid Security 社区方案缺点

社区方案能满足我们大部分的需求，但还存在一些问题。

第一个问题是我们发现浏览器对 BA 认证的支持很差。因此对于 Web 控制台，我们希望走统一的 SSO 认证。

第二个问题是为了支持业务平滑过渡到安全集群，上线初期必须兼容非认证的请求，当时我们使用的 0.12 版本没有该功能。

第三个问题是社区基于角色的鉴权模块只提供了底层的管理 API ,用户直接使用这些 API 非常不方便。

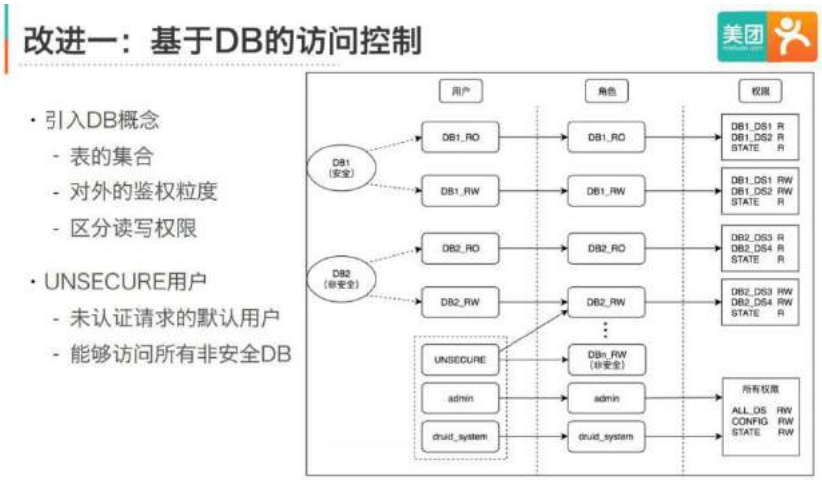
最后一个问题是社区还不支持审计日志。

针对这些问题，我们做了三个主要的改进。

3.3 改进

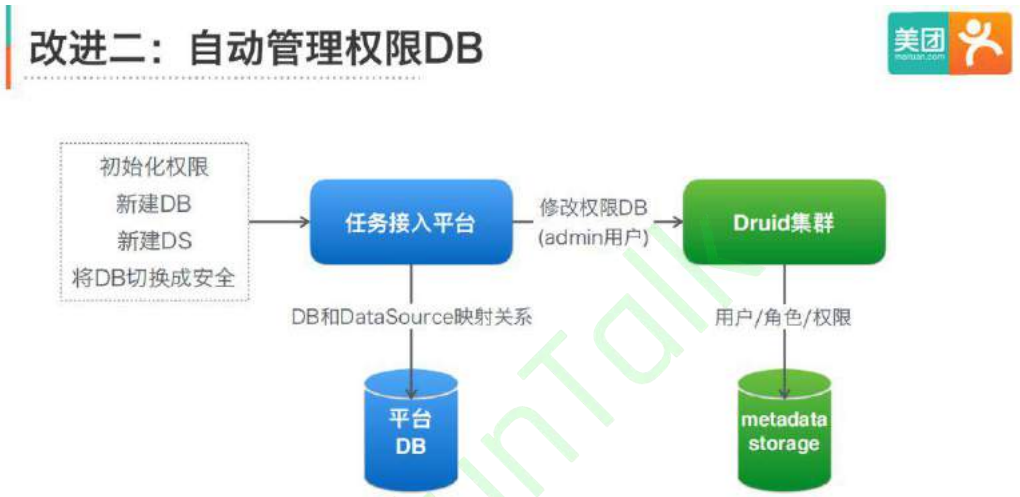
改进一：基于 DB 的访问控制

首先，为了简化权限的管理，我们引入了 DB 的概念，并实现了 DB 粒度的访问控制。业务通过 DB 的读写账号访问 DB 中的表。



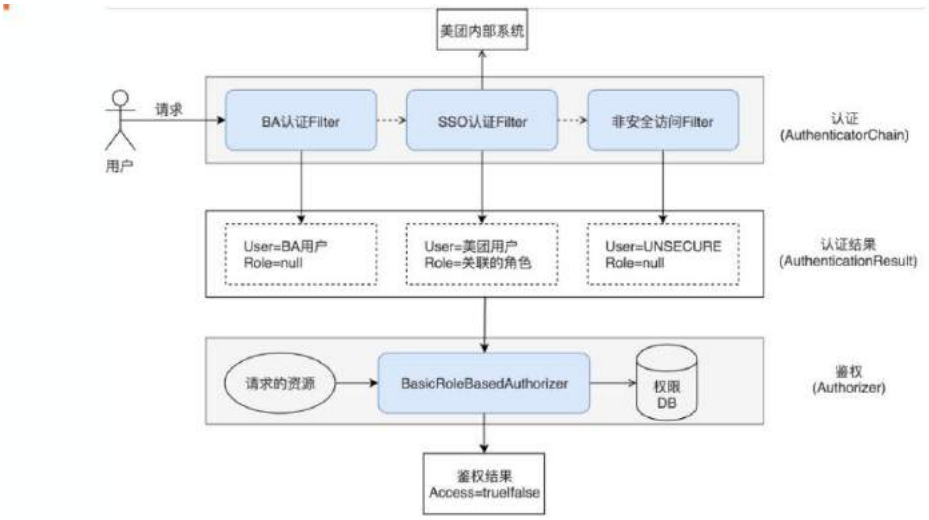
改进二：自动管理权限 DB

通过任务接入平台维护 DB 和 DataSource 的映射关系，并在 DB 和 DataSource 发生变化时，调用鉴权模块接口更新权限 DB。



改进三：支持 SSO 认证和非认证访问

自定义认证链条，通过 SSO 认证 Filter 实现 Web 控制台的 SSO 认证，通过非安全访问 Filter 兜底，兼容非认证的请求。



注意事项

1. 使用 0.13 以上版本 （或者 cherrypick 高版本的 bugfix ）

2. 上线流程

- 启用 basic-security 功能，用 allowAll 兜底
- 初始化权限 DB，创建匿名用户并授权
- 将 allowAll 替换为 anonymous
- 逐步回收匿名用户的权限

3. 上线顺序：

coordinator->overlord->broker->historical->middleManager

四、总结

4.1 关于 SQL

1. 如果还在用原生的 JSON 查询语言，强烈建议试一试

2. 社区在不断改进 SQL 模块，建议使用最新版本

3. Druid SQL 本质上是一个语言翻译层

- 对查询性能和稳定性没有太大影响
- 受限于 Druid 本身的查询处理能力，支持的 SQL 功能有限

4. 要留意的坑

- 大集群的 schema 推导效率
- Broker 需要等 schema 初始化后再提供服务（#6742）

4.2 关于 Security

1. Druid 包含一下 Security 特性，建议升级到最新版本使用

- 传输层加密
- 认证鉴权框架
- BA 和 Kerberos 认证
- RBAC 鉴权

2. 认证鉴权框架足够灵活，可根据自身需求扩展

3. 经历生产环境考验，完成度和稳定性足够好

4. 上线前应充分考虑兼容性和节点更新顺序

网易大数据体系之时序数据技术

嘉宾：范欣欣　编辑：王吉东

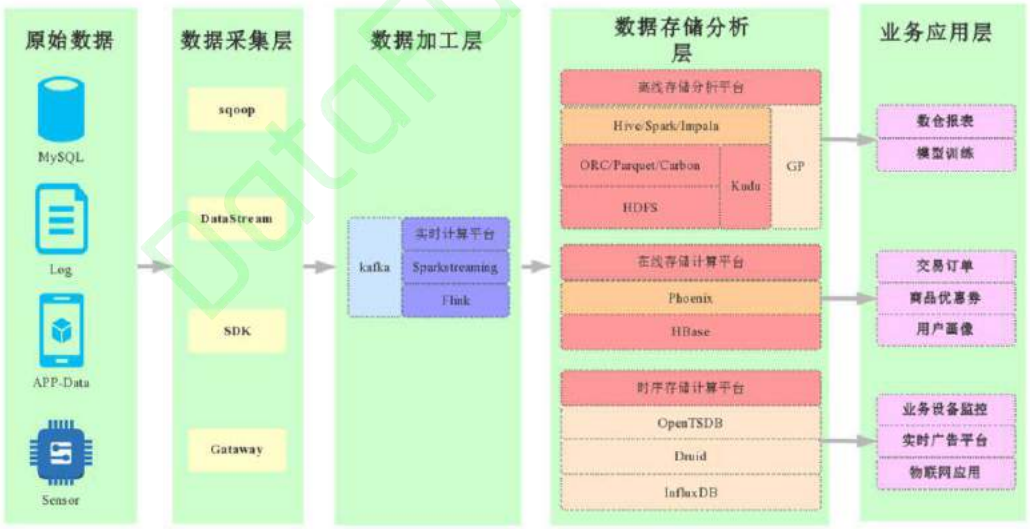
本次分享内容：

时序数据平台主要业务场景

时序数据平台体系架构

时序数据平台核心技术

PART01 时序数据平台主要业务场景



上图为数据的整体架构，大部分公司都是差不多的：

原始数据：MySQL、服务端的 Log、APP-Data、Sensor，大家知道现在穿戴设备很多，比如手表等，这样都会产生很多数据，这些数据都称为时序数据，随着时间的变化不断产生数据。

数据采集层：sqoop、DataStream、SDK、Gataway

数据加工层：数据存在 kafka 里，再经过一些流计算处理（Flink、Sparkstreaming）

数据存储分析层：

1. 离线存储分析平台：技术栈包括最底层的 HDFS、Kudu、GP 等数据存储，在这之上要做很多的计算，包括 Hive、Spark、Impala 等，他的应用场景包括数仓报表、机器学习、模型训练等；

2. 在线存储计算平台：应用的业务场景包括，交易订单，优惠券，用户画像等，这里主要应用的是 HBase；

3. 时间序列存储计算平台：应用场景包括，业务设备监控，实时广告平台，物联网应用，相关的技术包括 OpenTSDB、Druid、InfluxDB 等。

所以会根据不同的业务使用不同的平台来处理相关的数据，对于我们来说最大的工作是在数据存储端。

DataFunTalk

时序数据主要包括时间列、维度列和指标列，这 3 列缺一不可（特别是时间列）。

工业互联网时代，会源源不断产生各种各样的时序数据，时序数据平台有以下几种典型场景：

1. 系统监控：

- 物理机、云主机、容器：CPU、内存、IO 等
- 组件服务：数据库集群、Kafka 集群、HBase 集群

2. 任务监控：

- 查看指定 hadoop 任务耗用内存、CPU、IO 利用率等
- 查看集群消耗资源 TopN 任务、节点等
- 统计集群任务执行耗时

3. 应用性能监控：

DataFunTalk

- 应用调用次数，错误占比，页面加载延迟统计、地域统计分析
- 慢加载追踪，慢 SQL
- 异常会话追踪

4. 链路监控：

- 调用链全息排查
- 全局调用拓扑
- 链路依赖项分析梳理

5. 业务监控统计：

a. 电商

- 业务大盘：查看单量，金额，发货等业务指标
- 异常大盘：查看超卖，库存校准耗时，商品回调耗时，各种类型下单错误等异常指标

b. 广告

- 广告曝光点击消耗实时统计
- 流量地域分配

以基础系统监控服务为例，举例描述两个应用场景：

DataFunTalk

1. 物理机基础硬件指标监控

2. HBase-RegionServe 指标监控

直观地看，以上数据的横坐标都是时间戳，即设备随着时间不断吐数据。

时序数据应用场景的核心特征：

1. 时间区间查询，最近时间区的数据的查询频率远大于历史数据；

2. 多维条件查询，即多维字段随机组合查询；

3. 支持 TTL 机制，数据可以自动过期；

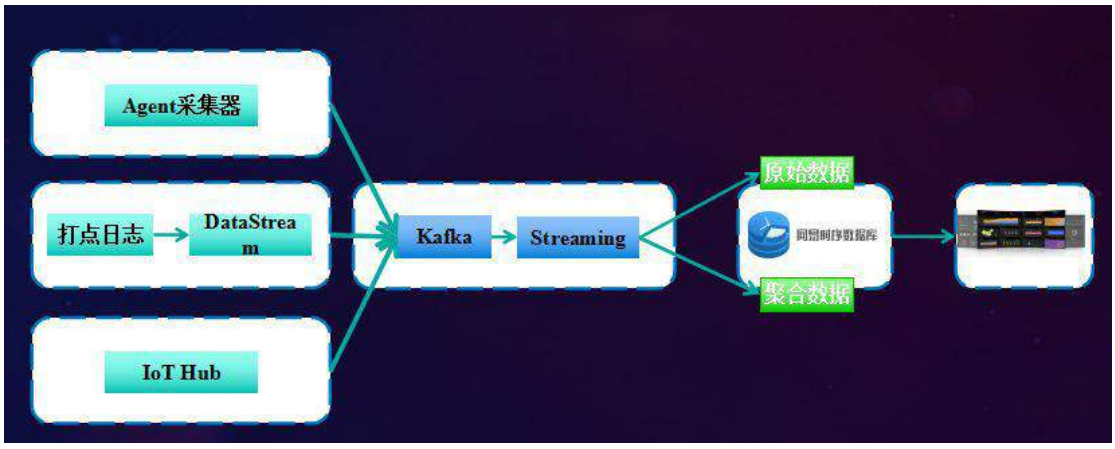
4. 支持高压缩率，数据压缩比要达到 10 以上；

5. 支持高效聚合

6. 支持集群可扩展，服务高可用，数据高可靠。

PART02 时序数据平台架构体系

时序序列平台以监控类系统体系架构为例，如下图所示：



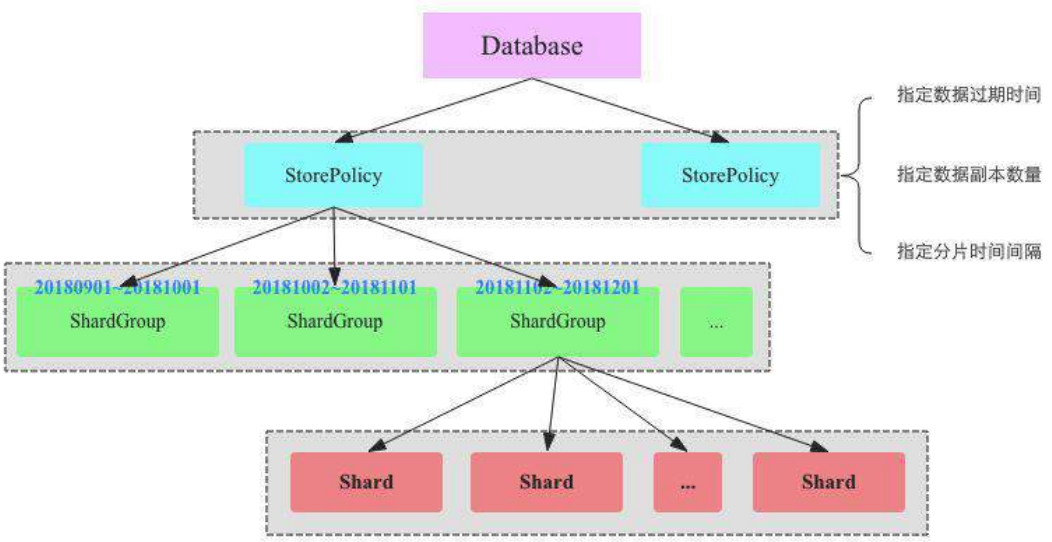
将多数据来源的数据（如 Agent 采集器，打点日志、流式数据，IoT Hub 等），通过 Kalfa 工具（分布式消息队列系统）进行流式处理，汇集到网易时序数据平台上。

PART03 时序数据平台核心技术

在上述架构体系中，数据库作为其核心环节。考虑到 DDB（分布式 MySQL）实时写入性能不足，HBase/ES 等开源 NoSQL 平台多维查询以及聚合计算等功能不够；针对海量时序数据这类应用场景，因此需要专门的时序数据库。

现有市场上较成熟的时序数据库主要包括 Druid、OpenTSDB、InfluxDB 等。网易结合以上各类时序数据库的优缺点，自主研发分布式时序数据库平台，支持高性能写入和读取，支持多维条件查询，支持聚合计算，且运行开销较小，可私有化、分布式部署。

该时序数据库的分布式集群结构与 hadoop，Hbase，Kudu 等架构非常相似：多台服务器用来存储数据，并通过 Raft 保证数据的一致性。数据的分布式计算是在分布式节点(node)上完成的，每个 node 上的数据存储计算系统称之为 shard server。在 Shard Server 的底层数据存储中，原始数据存一份，再按照索引的方式再存一份。



网易时序数据平台的数据存储方式是这样的：

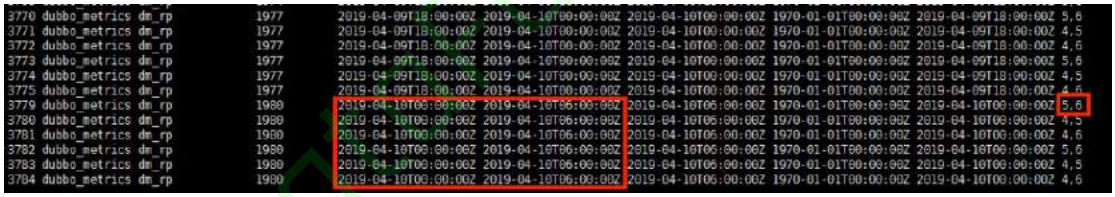
1. DataBase 中建立若干 StorePolicy（类似 MySQL 数据库中的表结构），其作用为：

(1) 指定数据过期时间；

(2) 指定数据副本数量；

(3) 指定分片时间间隔。

2. 每个 StorePolicy 中，按照指定的时间间隔建立若干 ShardGroup。将最新写入的数据（称之为热数据）存到对应的 ShardGroup 中，再通过哈希分片的方式将该 ShardGroup 分片成若干个 Shard，这些 Shard 会分布至整个集群。



以上图为例，同一时间段不同 shard 分布到不同的节点上。例如：左侧大方框中包含的 6 条记录，代表该时间段的记录分布在 6 个 shard 里；右侧小方框中的 5 和 6，代表该 shard 共有 2 个副本，分别分布在集群中的 5 号节点和 6 号节点。

由此可以看出，同一时间段的多个 shard 是分布在多个节点上。这样做的好处是：查找指定时间段的数据时，只需要在相应节点中查到对应的 shard 里的数据即可，避免了全局遍历的情况。

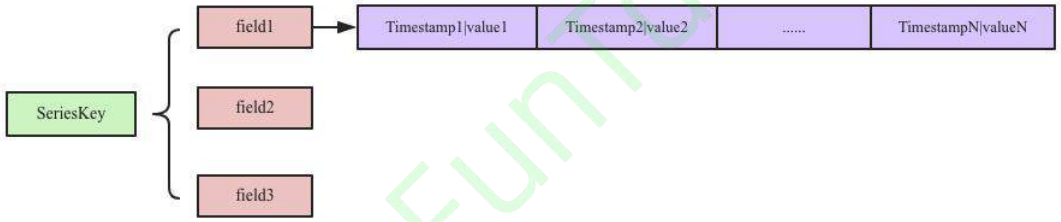
3. 数据在 shard 中的存储方式：

(1) 时间线列式存储：

具体来说，SeriesKey 相当于原始数据（DataSource），具有不同维度；存储的时候，所有的时间存到一起，所有的 value 单独存在一起，这种存储方法叫做列式存储。列式存储有两大好处：

- 每一列数据类型相同，因此压缩率非常高；

❖ 方便对列做聚合计算。



什么是列式存储？

传统的关系型数据库如 Oracle、MySQL、SQL SERVER 等，都采用的是行式存储法(Row-based)，在基于行式存储的数据库中，数据是按照行数据为基础逻辑存储单元进行存储的，每行中的数据在存储介质中以连续存储形式存在。

不同于行式存储，列式存储以列为基础单元，每列的数据在存储介质中以连续存储的形式存在。

针对海量分布式数据背景的 OLAP（on-line analysis processing），列式存储可在内存中高效组装各列数据并形成关系集，可避免全表扫描，显著减少 IO 消耗。

(2) 倒排索引

支持 sql 语句进行条件筛选；在条件筛选的过程中，使用倒排索引方法，实现快速定位，可“直达用户需求”。

正排索引：key-value 中，通过 key 去寻找 value；

倒排索引：通过 value（或包含 value）去寻找对应的 key。

正排索引需要首先对全局进行扫描遍历，进而从中做筛选；而倒排索引可以仅抽取符合条件的 value 值，节省大量的资源。

(3) shard 自动扩容

数据库作为分布式系统，需要考虑系统的扩容（即增加节点）。

增加节点后，为保证数据存储的均衡，将不再更新的数据（即历史数据）作为 cold shard，平均分布到集群的所有节点（包括新增加的节点）中，而实时更新的数据（即 hot shard，随时间不断写入）不做任何处理。

(4) 多级存储优化

考虑到时序数据的一个特点：热数据查询概率远大于冷数据。针对这一特性，做了如下两个优化：

A. 考虑到数据文件可能存在多级索引，针对冷数据，只加载一级索引即 root 索引；针对热数据，会加载所有索引。

B. 因此针对集群服务器配置方面，每个集群 12 块盘，只需要 1 块 ssd 用来存储 hot shard 即可，其余均使用普通 hhd 即可，最大程度降低硬件成本。

cluster 是整个集群，包含 n 个 node；

node 通常掌控独立的资源，包括 cpu 等；一台机器可以多个 node。

shard 一般是从数据角度来说。例如，1000 条数据按 id 分，存 10 份，就是 10 个 shard。

shards 分散在多个 node 上。

最后通过以上手段，来实现刚刚所说的时序数据应用场景的核心特征：时间区间查询，多维条件查询，支持 TTL 机制，支持高压缩率，支持高效聚合，支持集群可扩展，服务高可用，数据高可靠。

DataFunTalk

快手 Druid 精确去重的设计和实现

嘉宾：邓钫元　编辑：王吉东

本次分享内容提纲：

- 快手 Druid 平台概览
- Druid 精确去重功能设计
- Druid 其他改进
- 快手 Druid Roadmap

★ 快手 Druid 平台概览

1. 从业务需求角度考虑，快手为什么选择 Druid ？

快手的业务特点包括超大数据规模、毫秒级查询时延、高数据实时性要求、高并发查询、高稳定性以及较高的 Schema 灵活性要求；因此快手选择 Druid 平台作为底层架构。由于 Druid 原生不支持数据精确去重功能，而快手业务中会涉及到例如计费等场景，有精确去重的需求。因此，本文重点讲述如何在 Druid 平台中实现精确去重。另一方面，Druid 对外的接口是 json 形式（Druid 0.9 版本之后逐步支持 SQL），对 SQL 并不友好，本文最后部分会简述 Druid 平台与 MySQL 交互方面做的一些改进。

2. 下面重点介绍快手 Druid 平台的架构：

和大多数数据平台一样，快手的数据源主要有两种：基于 kafka 的实时数据源和建立在 hadoop 内的离线数据源。将这两种数据源分别通过 kafka index 和 hadoop index 导入到平台核心—— Druid 模型中。

在 Druid 中，将数据进行业务隔离、冷热分层（ 例如：冷数据放在硬盘，热数据放在 SSD ）

外部业务接口方面 ,支持通过 API 自行开发业务模块 ,支持通过 Kwai BI 做可视化 ;同时针对小部分人需求 ,现已支持 tableau (tableau 通过 hive 连接 Druid ）。

数据平台辅助系统包括 metric 监控（ 实时监控 CPU 占用率情况，进程情况，lag 信息等 ）、探针系统（ 用来查询，定义查询热度、维度等信息，有

助于优化 ）、管理系统（ 在原生态 Druid 的 json 接口基础上定制管理系统，实现任务的可视化导入等管理操作 ）

★ Druid 精确去重概述

1. 原生 Druid 去重功能支持情况

a. 维度列

- cardinality agg，非精确，基于 hll 。查询时 hash 函数较耗费 CPU
- 嵌套 group by，精确，耗费资源

社区 DistinctCount 插件，精确，但是局限很大：

① 仅支持单维度，构建时需要基于该维度做 hash partition

② 不能跨 interval 进行计算

b. 指标列

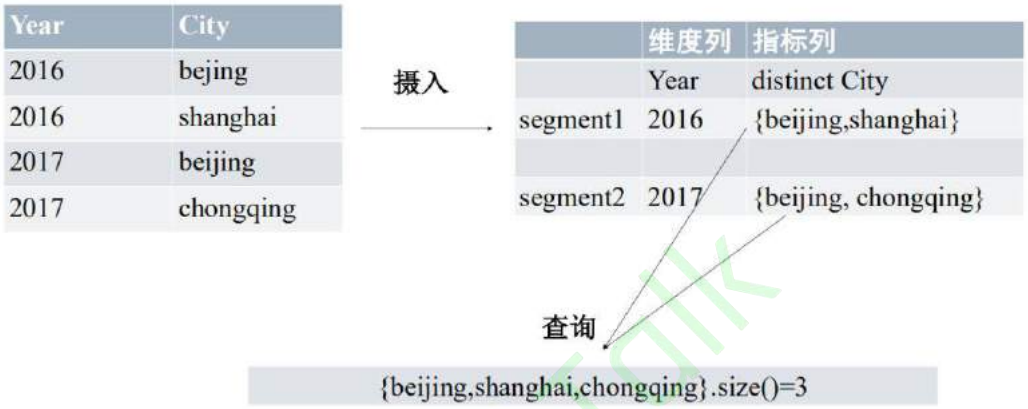
HyperUniques/Sketch ,非精确 ,基于 hll ,摄入时做计算 ,相比 cardinality agg 性能更高。

结论：Druid 缺乏一种支持预聚合、资源占用低、通用性强的精确去重支持。

2. 精确去重方案

a. 精确去重方案：hashset

使用 hashset 的方式存储，具备简单通用等优点，但是资源占用非常大。以下图所示为例，将左上角的原始数据使用 hashset 方法，Year 作为维度列，City 作为指标列；按照维度列分成 2 个 Segment，转换为右上角的数据格式；接下来将指标列聚合到 broker 中，计算出 size，即得到最终结果。整个过程如下图所示：



但是使用 hashset 的去重方式，其资源占用非常大；具体来说，假设有 5000W 条平均长度为 10B 的 string（共 500MB），中间生成的 Hashset 会产生一系列链表结构，导致内存可达到 5G，即内存扩展可达 10 倍。因此，直接使用 hashset 的方法可以是完全不可行的。

在 hashset 方法基础上可以考虑一些优化的思路：例如，通过增加节点，将数据分散到不同机器上，查询时在每台机器上分别聚合进而求和；另一种方

式类似于 MapReduce，计算前利用 shuffle 模型将数据打散。然而这两种优化思路和 Druid 平台属性有冲突，因此都不适用于 Druid 这个平台。

b. 精确去重方案：字典编码+Bitmap

Bitmap 方法，即用一个位来表示一个数据（ 这是理论上的最小值 ）。这种方案的思路是：对要去重的数据列做编码，将 string 或其他类型的数据转化成 int 型的编码，摄入到 Druid 后通过 Bitmap 的形式存储；查询时将多个 Bitmap 做交集，获取结果。Bitmap 的范围是 42 亿，最大占用空间 500M 左右；还可以使用压缩算法，使空间占用大大减少。

因此，字典编码+Bitmap 的方式，优点是存储和查询资源占用少，可以将构建和查询分开（ 即字典不参与查询 ）；缺点是：字典需要做全局编码，其构建过程较复杂。

考虑到 Kylin 曾使用过这种方案，因此快手的后续优化方案选择在这种方案的基础上进行优化。

3. 字典编码方案

a. 字典编码方案的使用

① Redis 方案

DataFunTalk

Redis 可以进行实时编码，可以同时支持离线任务和实时任务，因此，实时处理数据和离线处理数据可以使用相同的方案。

但是 Redis 的 ID 生成速度有瓶颈，最大 5W/s；查询上压力也较大，对交互会造成一定的麻烦；同时对 Redis 的稳定性要求也比较高。

② 离线 MR 方案

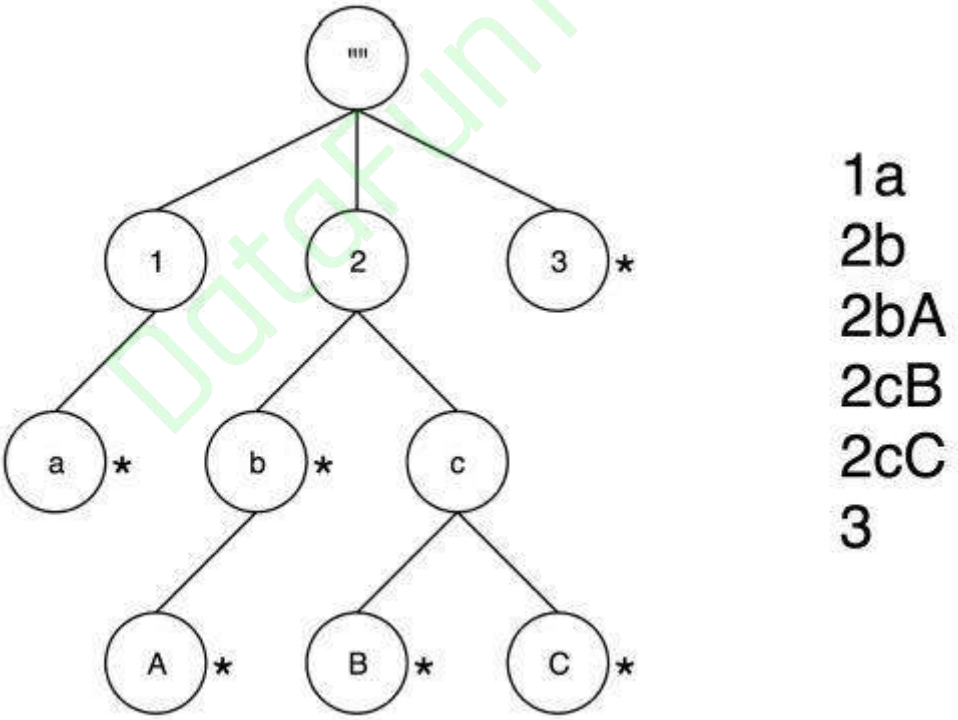
离线 MR 方案，利用 MR 的分布式存储，编码和查询的吞吐量更高，还用用更高的容错性。但是 MR 仅支持离线导入任务。

综合这两种字典编码方案的优劣，得到以下结论：

- 离线任务使用离线 MR 编码，通过小时级任务解决实现准实时
- 实时任务仅支持原始 int 去重（ 无需编码 ）

b. 字典编码方案模型

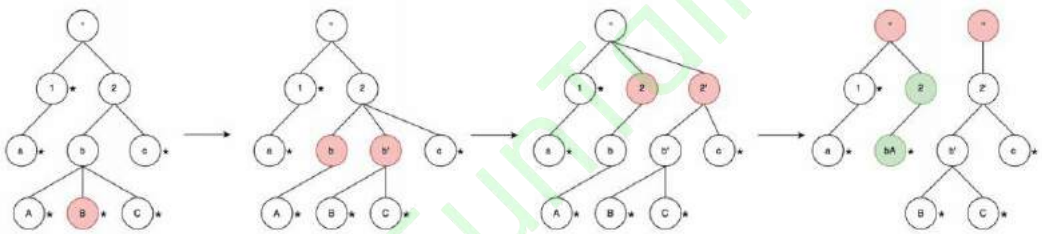
快手的字典编码方案参照的是 Kylin 的 AppendTrie 树模型，模型详见下图。



Trie 树模型主要是使用字符串做编码；快手支持各种类型数据，可将不同类型数据统一转换成字符串类型，再使用 Trie 树模型做编码。Trie 树模型可以实现 Append，有两种方式：

- 按照节点（ 而不是位置 ）保存 ID，从而保持已有 ID 不变。
- 记录当前模型最大 ID，便于给新增节点分配 ID 。

但是这种模型会带来一个问题：假如基数特别大，Trie 树在内存中就会无限扩张。为了控制内存占用率，选择单颗子树设定阈值，超过即分裂，进而控制单颗子树内存消耗。分裂后，每棵树负责一个范围（ 类似于 HBase 中的 region 分区 ）；查询时候只需要查询一颗子树即可。



为节省 CPU 资源，基于 guava LoadingCache，按需加载子树（即懒加载），并使用 LRU 方法换出子树。

LRU（Least Recently Used）是内存管理的一种置换算法，即内存不够时，将最近不常用的子树从内存清除，加载到磁盘上。LRU 算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。

c. 字典并发构建：

字典会存在并发构建的问题，分别使用 MVCC 以及 Zookeeper 分布式锁的方案。

① 使用 MVCC（持久化在 hdfs 上），在读取的时候会使用最新版本，拷贝到 working 中进行构建，构建完成生成新的版本号；如历史数据过多，会根据版本个数及 TTL 进行清理。

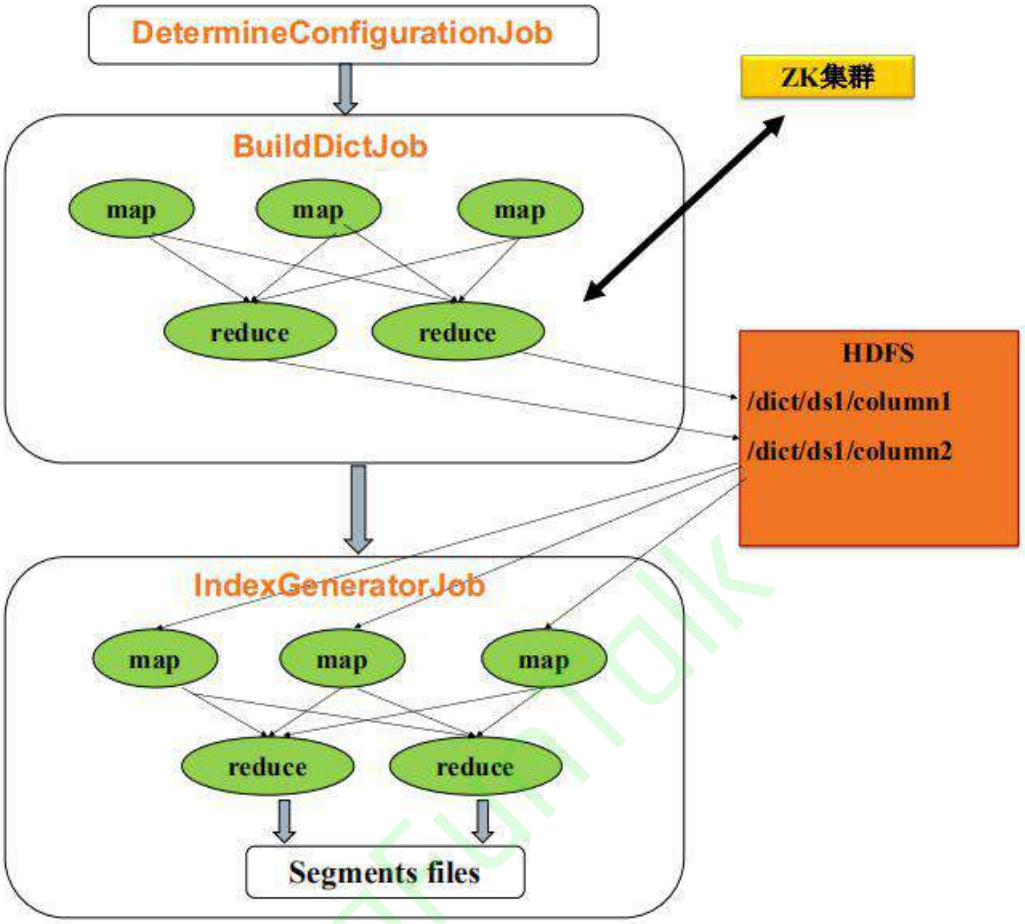
② 构建字典的过程中会操作临时目录，如果存在多个进程同时去写临时目录，会存在冲突问题；因此引入 Zookeeper 分布式锁，基于 DataSource 和列来做唯一的定位，从而保证同一个字典同时只能有一个进程。

d. 精确去重的实现

① 新增 unique 指标存储

- 使用 ComplexMetricSerde 定义一个指标如何 Serde，即序列化与反序列化；
- 使用 Aggregator，让用户定义一个指标如何执行聚合，即上卷；
- 使用 Aggregator 的 Buffer 版本 BufferAggregator 作为 Aggregator 的替代品；
- 使用 AggregatorFractory 定义指标的名称，获取具体的 Aggregator 。

② 精确去重的整体流程介绍



- 使用 DetermineConfigurationJob 计算最终 Segment 分片情况。
- 使用 BuildDictJob，Map 将同一去重列发送到一个 reducer 中（Map 端可先 combine）；每个 reducer 构建一列的全局字典；每列字典构建申请 ZK 锁。
- IndexGeneratorJob，在 Map 中加载字典，将去重列编码成 int；Reducer 将 int 聚合成 Bitmap 存储；每一个 reducer 生成一个 Segment。

此套精确去重的使用方法：

导入时，定义 unique 类型指标

```
"parser":{
    "type":"parquet",
    "parseSpec":{
        "dimensionsSpec":{
            "dimensions":[
                "fre_city"
            ],
            "dimensionExclusions":[

            ]
        },
        "format":"timeAndDims",
        "timestampSpec":{
            "missingValue":1544630400000,
            "column":"p_date",
            "format":"auto"
        }
    }
},
"metricsSpec":[
    {
        "type":"count",
        "name":"count"
    },
    {
        "name":"user_id_uniq",
        "type":"unique",
        "fieldName":"user_id"
    }
],
```

查询时，使用 unique 类型查询

e. 性能优化与测试

① 优化字典查询

· 存在一个超高基数列（UHC）的去重指标：在 IndexGenerator 之前增加 ClusterBy UHC 任务，保证每个 map 处理的 uhc 列数据有序，避免多次换入换出。

· 存在多个超高基数列的去重指标：拆成多个 DataSource，调大 IndexGenerator 任务的 map 内存，保证字典能加载到内存

② Bitmap 查询优化

减少 targetPartitionSize 或调大 numShards，增加 Segment 个数，提升并行度。

使用 Bitmap 做 or 计算时候，建议使用 batchOr 方法，而不是逐行 or，避免 I/O 与 or 计算交替，缓存持续被刷新导致性能降低。关于 batchOr 的选择，rolling_bitmap 提供一些接口，默认选择 naive_or（尽量延迟计算模型），通常情况下其性能较好；priorityqueue_or 使用堆排序，每次合并最小两个 bitmap，消耗更多内存，官方建议 benchmark 后使用；如果结果是较长的连续序列，可以选择手动按顺序依次 inplace or。

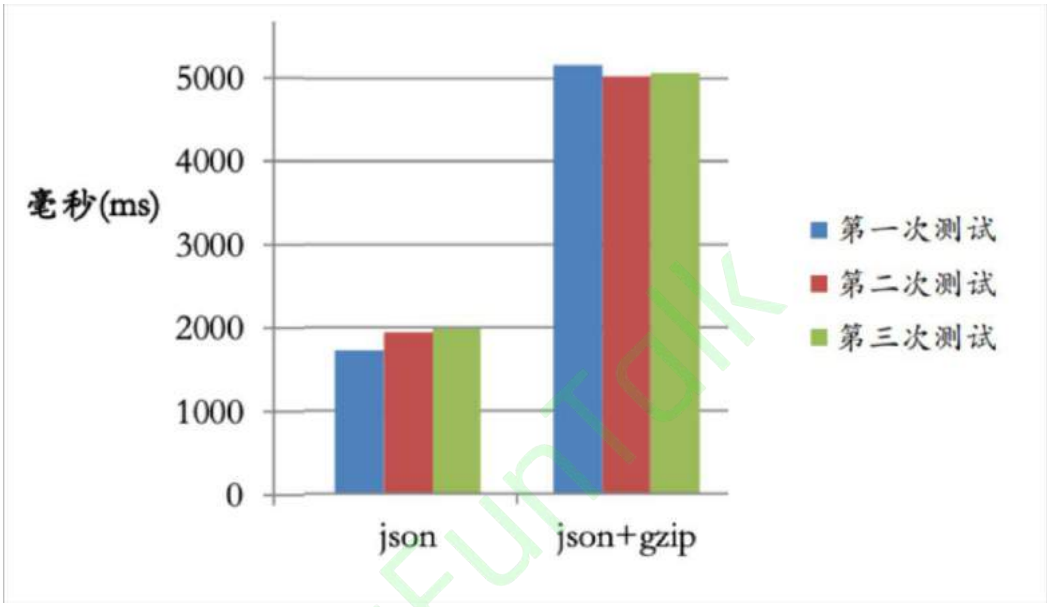
另一个思路是 Croaring-high performance low-level implementation

Rolling bitmap 用 C 语言实现了 Simd，相对于 java 版，Smid 版更多地利用了向量指令（avx2），性能提升了 80%。

快手将 Smid 以 JNI 的方式引入，对 100W 的 Bitmap 做 Or 操作，使用 Java 版本单线程操作，计算用时 13s；用 JNI 调用 Croaring，数据进行反序列化（需要 memcpy 用时 6.5s），计算消耗 7.5s，总用时 14s（慢于 java 版本）。Croaring 暂时不支持原地 inplace 反序列化（ImmutableRoaringBitmap），导致其实际运行效率与官方称的 80%提升不一致。

③ 传输层编码

Broker 与 Historical 之间通过 http 协议交换数据，默认打开 gzip 压缩（也可以选择不压缩，即 plain json）。测试发现 gzip 压缩的过程中会耗用大量 CPU，因此在万兆的网络下建议不压缩。



④ 对上述查询优化步骤依次性能测试：

8 个维度，10 亿基数的数据，选择列 author_id 作为其去重指标，摄入后达 150W 行；10 台 historical 机器：

优化方法	Segment个数	查询时间
base	1	50s
targetPartitionSize=15w	10	7s
batchOr	10	4s
关闭gzip	10	2s

未做优化去重，查询时间 50s；

增加 Segment 数量至 10，查询时间为 7s，提升约 7 倍性能；

将 Or 的策略设置为 BatchOr，查询时间为 4s；

关闭 gzip 压缩，查询时间为 2s。

以上是对 10 亿基数数据作去重的查询时间；如果数据基数在 1 亿以下，查询时间为毫秒级。

★ Druid 其他方面做的改进

1. 资源隔离部署方案

该方案为 Druid 官方推荐部署方案，充分利用 Druid 的分层特质性，根据业务，根据数据冷热，分到不同 proxy 上，保证各个业务相互不受影响。

2. 物化视图

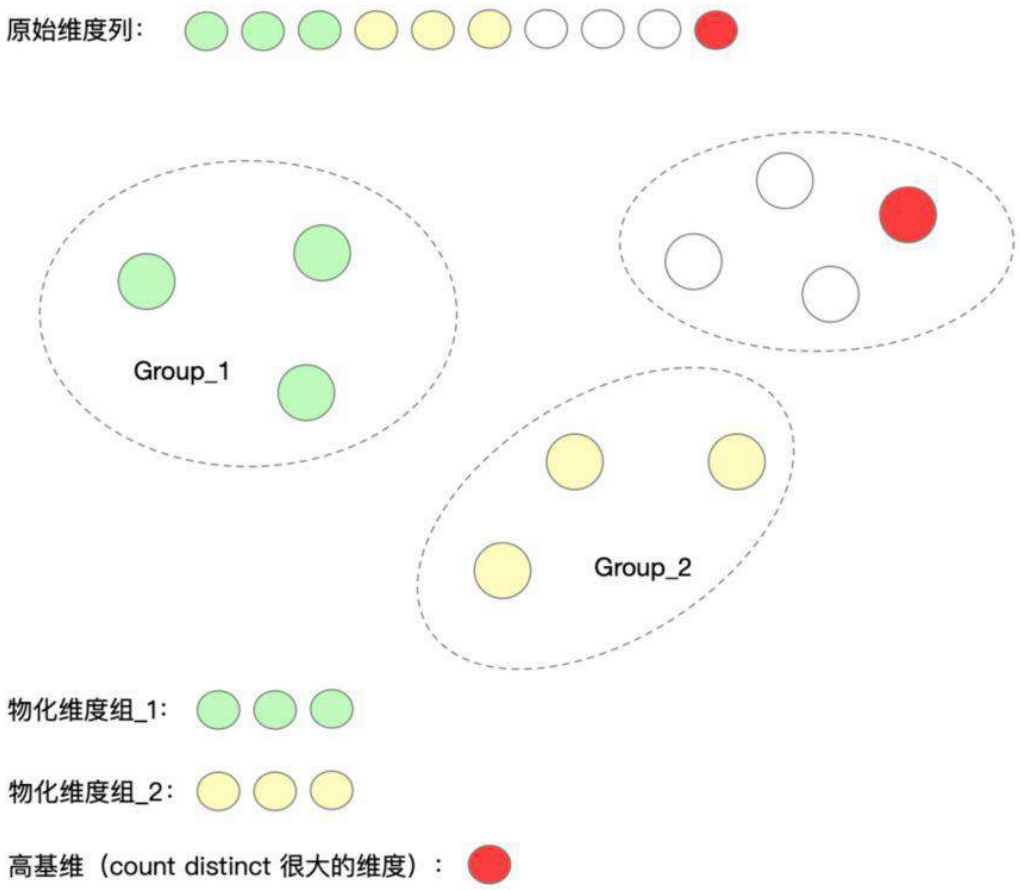
物化视图是包括一个查询结果的数据库对象，可以将其理解为远程数据的的本地副本，或者用来生成基于数据表求和的汇总表，这些副本是只读的。

如果想修改本地副本，必须用高级复制的功能；如果想从一个表或视图中抽取数据时，可以用从物化视图中抽取。物化视图可以通过数据库的内部机制可以定期更新，将一些大的耗时的表连接用物化视图实现，会提高查询的效率。

a. 维度物化

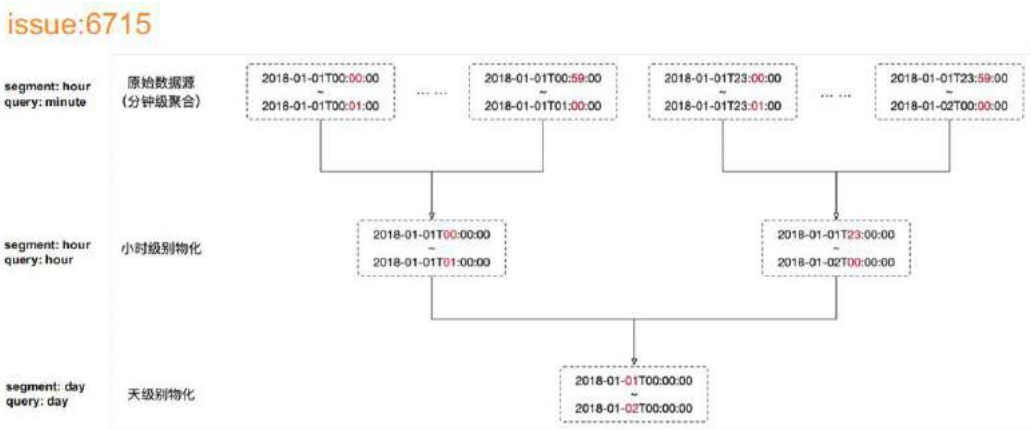
Druid 社区 0.13 开始具备物化视图功能，社区实现了维度的物化。应用场景如下：原始数据有很高维度，而实际查询使用到的维度是原始维度的子集；因此不妨对原始维度做小的聚合（类似 Kylin 中的 cube 和 cuboid）；对经常查询的部分维度做物化。

DataFunTalk



b. 时序物化

除了维度上的物化，还包括时序上的物化，例如将原始数据按照小时聚合、按照分钟聚合。



c. 物化效果

DataSource	物化膨胀率	物化命中率	p99	p99提升
ds1	16%	33%	5s->2s	150%
ds2	3%	60%	6s->2s	200%
ds3	28%	99%	10s->5s	100%
ds4	1%	99%	8s->1s	700%
ds5	1%	95%	3s->0.3s	900%

这里，主要关心物化膨胀率和物化命中率，两项指标会影响最终的查询效率。物化膨胀率，表示物化后的数据存储耗用资源情况；物化命中率，可以描述物化后与实际应用场景的匹配度。表格中，ds4 和 ds5 都做到了物化膨胀率很低，物化命中率很高，因此，查询效率可获得 7~9 倍的提升。

3. Historical 快速重启

快手数据平台硬件资源，大约是 12 块容量 2T 的 SATA 硬盘，数据量平均为 10W segments，数据占用 10T 空间，重启一次大约 40min。

Druid 中，默认重启的时候会加载全部数据；考虑到超过 10T 的元信息在启动时并不需要，可以将加载数据推迟到查询时候。因此，使用 Guava

Suppliers.memoize 命令，延迟数据加载信息，只有在查询的时候才作列的加载。此过程通过一个参数（LazyLoad）控制。

（此代码已提交 Druid 社区：druid.segmentCache.lazyLoadOnStart（pr: 6988））

优化后，重启过程只需要 2min，提升 20 倍。

4. Kafka Index 方面的改进

a. TaskCount 自动伸缩

Kafka index 任务数即 TaskCount 为固定，需要按照峰值任务数设定，这样导致在非高峰时刻会存在资源浪费。

这里实施的一个优化策略是：

- KafkaSupervisor 增加 DynamicTaskCountNotice
- 基于 task 的 cpu use & kafka lag 阈值，进行 25%增减 task count
- 无需重启 supervisor

```
// 1. call this function will killing all pedding tasks and push all running tasks to begin publishing
gracefulShutdownInternal();
log.info(
    "dynamic:killing all tasks that haven't been assigned to a worker yet, and causing all running tasks

// 2、clear all the memory map
taskGroups.clear();
pendingCompletionTaskGroups.clear();
partitionGroups.clear();
sequenceTaskGroup.clear();
log.info("dynamic: clear all the memory map");

// 3、update the taskCount, the spec in mysql database is not update
setCurrentTaskCount(newTaskCount);
log.info("dynamic: update the CurrentTaskCount:[%d]", newTaskCount);
```

b. 精细化调度

Middle Manager 的 indexing task 资源分配从 slot 改成按照内存大小分配（类似于 MapReduce 从 1.0 到 2.0 的改进），具体优化方法如下：

- 区分 Kafka indexing task 和 Hadoop indexing task（一般 Hadoop indexing task 不需要占用内存）
- 允许在提交 task 时指定 task 内存大小。

	任务数量 (个/天)	单Task 当前分配内存量 (GB) f(S)	单Task 精细分配平均内存量 AVG [g(S)]	当前分配 总内存占用（分钟*MB） SUM[T(S) * f(S)]
实时	31521	10	2.407716297 (GB)	19983805375
离线	5969	10	90.2959(MB)	503484523.2

使用这种方式优化，实时数据任务处理可节省超过 65%的内存占用，离线数据任务处理可节省超过 87%的内存占用。

5. 元数据交互

元数据（Metadata），又称中介数据、中继数据，主要是描述数据属性（property）的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能，是关于数据的组织、数据域及其关系的信息，可以看作是一种电子式目录。简言之，元数据就是关于数据的数据。

a. Overlord 与 MySQL 交互优化

Overlord 节点负责接收任务，协调和分配任务，为任务创建锁，并返回任务状态给任务发送方，Overlord 有两种运行模式：本地模式或者远程模式（默认本地模式）。

Overlord 控制台可以查看等待的任务、运行的任务、可用的 worker，最近创建和结束的 worker 等。

Segment 是 Druid 中最基本的数据存储单元，采用列式的方式存储某一个时间间隔（interval）内某一个数据源（dataSource）的部分数据所对应的所有维度值、度量值、时间维度以及索引。

Segment 的逻辑名称结构为：

<dataSource>_<intervalStart>_<intervalEnd>_<version>_<partitionNum>

<dataSource> 表示数据源（或表）；<intervalStart> 和 <intervalEnd> 分别表示时间段的起止时间；<version> 表示版本号，用于区分多次加载同一数据对应的 Segment；<partitionNum> 表示分区编号（在每个 interval 内，可能会有多个 partition）

Segments 在 HDFS 上的物理存储路径下包括两个文件：descriptor.json 和 index.zip 。前者记录的是 Segment 的描述文件（样例见下表），其内容也保存在 Druid 集群的元数据的 druid_segments 表中；index.zip 则是数据文件。

描述文件 descriptor.json 样例：

{"dataSource": "AD_active_user","interval": "2018-04-01T00:00:00.000+08:00/2018-04-02T00:00:00.000+08:00","version": "2018-04-01T00:04:07.022+08:00","loadSpec": {"type": "hdfs","path": "/druid/segments/AD_active_user/20180401T000000.000+0800_20180402T000000.000+0800/2018-04-01T00_04_07.022+08_00/1/index.zip"},"dimensions": "appkey,spreadid,pkgid","metrics": "myMetrics,count,offsetHyperLogLog","shardSpec": {"type": "numbered","partitionNum": 1,"partitions": 0},"binaryVersion": 9,"size": 168627,"identifier": "AD_active_user_2018-04-01T00:00:00.000+08:00_2018-04-02T00:00:00.000+08:00_2018-04-01T00:04:07.022+08:00_1"}

将 druid_segments 增加索引（dataSource，used，end），查询时间从 10s 优化到 1s 。

b. Coordinator 与 MySQL 交互优化

Druid 的 Coordinator 节点主要负责 Segment 的管理和分发。具体的就是 ,Coordinator 会基于配置与历史节点通信加载或丢弃 Segment 。Druid Coordinator 负责加载新的 Segment，丢弃过期的 Segment，管理 Segment 的副本以及 Segment 的负载均衡。

Coordinator 会根据配置参数里的设置的事件定期的执行，每次执行具体的操作之前都会估算集群的状态。与 broker 节点和 historycal 节点类似的，Coordinator 节点也会和 Zookeeper 集群保持连接用于交互集群信息。同时，Coordinator 也会和保存着 Segment 和 rule 信息的数据源保持通信。可用的 Segment 会被存储在 Segment 的表中，并且列出了所有应该被集群加载的 Segment 。Rule 保存在 rule 的表中，指出了应该如何处理 Segment 。

① Segment 发现

将 Coordinator 中默认的全量扫描 druid_segments 表改成增加读取，druid_segments 添加索引（used，created_date），查询时间从 1.7min 优化到 30ms 。

② 整个协调周期

- Segment 之前使用 TreeSet 按时间排序确保优先 load 最近的 segment 现在优化成 ConcurrentHashSet，并分成最近一天和一天前的两个 set，先操作最近一天的；
- Apply rules 的时候对 LoadRule 先判断集群副本和配置的是否一致，是的话就跳过，提升并发能力；
- Cleanup 和 overshadow 两个操作合并；
- 查询时间从 3min 优化到 30s 。

★ 快手 Druid Roadmap

1. 在线化

今年会实现多集群、高可用，着重保证在线业务的 SLA，同时线上业务也要着重权限的管理。

2. 易用性

实现对 SQL 的支持，同时快手的 BI 产品也会做相应的升级和优化。

3. 性能及优化（Druid 内核方面，与社区合作）

- 自适应的物化视图，智能化

- 数值型索引

- 向量化引擎

最后附上我们提交社区的代码：

https://github.com/apache/incubator-druid/pull/7594（精确去重功能）

https://github.com/apache/incubator-druid/pull/6988（historical 快速重启）

DataFunTalk

基于 Kafka 的实时计算引擎如何选择？Spark or Flink ？

作者：哥不是小萝莉　编辑：HohXil

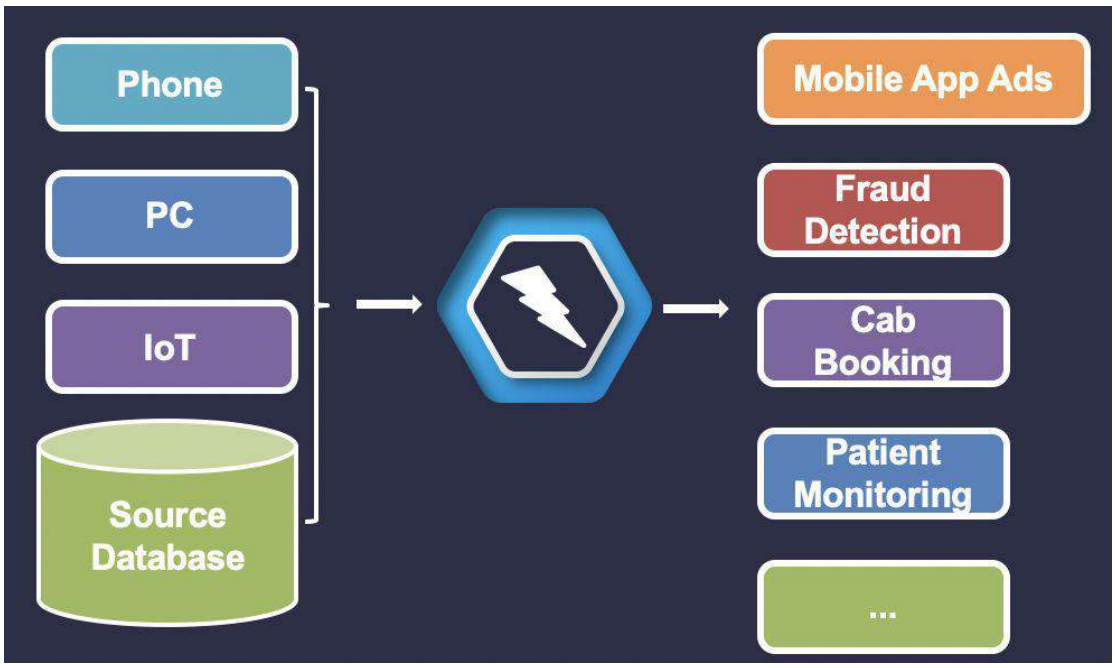
1. 前言

目前实时计算的业务场景越来越多，实时计算引擎技术及生态也越来越成熟。以 Spark 和 Flink 为首的实时计算引擎，成为实时计算场景的重点考虑对象。那么，今天就来聊一聊基于 Kafka 的实时计算引擎如何选择？Spark or Flink？

2. 为何需要实时计算？

根据 IBM 的统计报告显示，过去两年内，当今世界上 90%的数据产生源于新设备、传感器以及技术的出现，数据增长率也会为此加速。而从技术上将，这意味着大数据领域，处理这些数据将变得更加复杂和具有挑战性。例如移动应用广告、欺诈检测、出租车预订、患者监控等场景处理时，需要对实时数据进行实时处理，以便做出快速可行的决策。

DataFunTalk



目前业界有开源不少实时计算引擎，以 Apache 基金会的两款开源实时计算引擎最受欢迎，它们分别是 Apache Spark 和 Apache Flink 。接下来，我们来聊一聊它们的使用场景、优势、局限性、相似性、以及差异性。方便大家在做技术选型时，选择切合项目场景的实时计算引擎。

2.1 如何理解流式与实时？

说起实时计算，可能会说到流式计算，那么流式和实时是否是等价的呢？严格意义上讲，它们没有必然的联系。实时计算代表的是处理数据耗时情况，而流式计算代表的是处理数据的一种方式。

2.2 什么是流式处理？

首先，它是一种数据处理引擎，其设计时考虑了无边界的数据集。其次，它与批处理不同，批处理的 Job 与数据的起点和终点有关系，并且 Job 在处理完有限数据后结束，而流式处理用于处理连续数天、数月、数年、或是永久实时的无界数据。

流处理的特点：

- 容错性：如果节点出现故障，流式处理系统应该能够恢复，并且应该从它离开的位置再次开始处理；
- 状态管理：在有状态处理要求的情况下，流式处理系统应该能够提供一些机制来保存和更新状态信息；
- 性能：延时应尽可能的小，吞吐量应尽可能的大；
- 高级功能：事件时间处理，窗口等功能，这些均是流式处理在处理复杂需求时所需要的功能。

2.3 什么时候适合流式处理？

流式处理可以分析连续的数据流，在这种方式中，数据被视为连续流，处理引擎在很短的时间内（ 几毫米到几分钟 ）内取数、分析、以及响应。下面让我们来看看流式处理的场景使用场景：

DataFunTalk

异常检测：流式处理可以应用于连续的数据流并近乎实时的检测异常。例如，在金融交易数据中，欺诈性交易可以被视为异常，流式处理可以检测到这些，保护银行和客户免受财务损失。

业务流程监控：业务流程涉及特定域中的多个事件。例如，在电子商务业务中，从下单、支付、出库、送货、再到用户签收的所有事件都可以被视为一个业务流程。流处理可用于监控此类流程的异常情况，例如在时间范围内为完成、交付商品时出错等。

告警：流式处理可用于根据指定规则触发告警，满足特定条件，可以实时将告警发送到不同的目标。

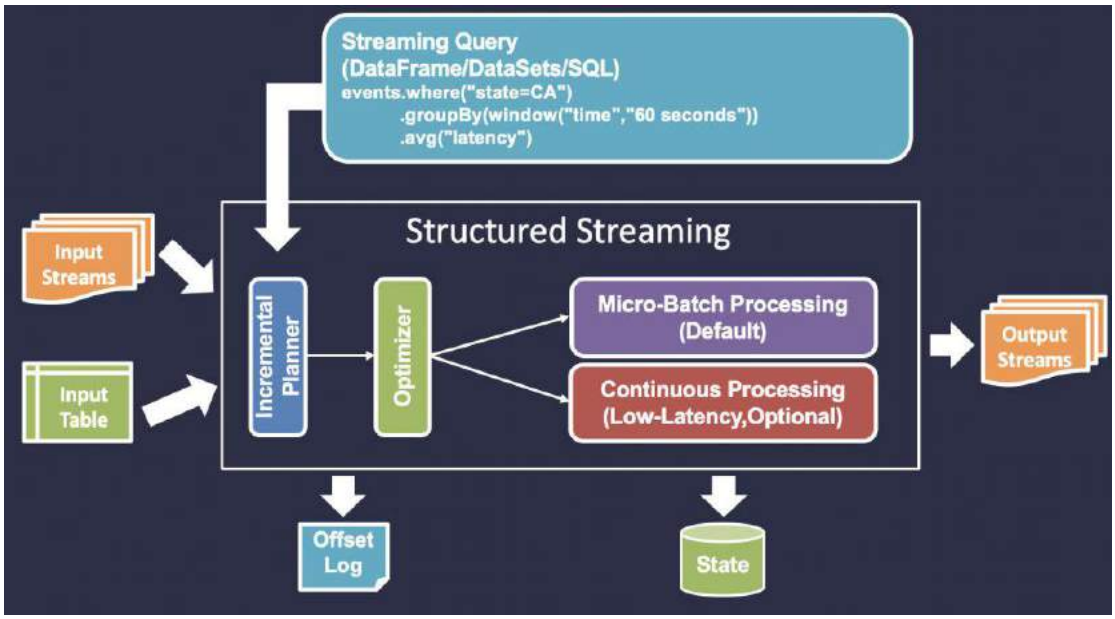
3. Spark

Spark 已成为批处理中 Hadoop 的真正继承者，也是第一个完美支持 Lambda 架构的框架。Spark 受欢迎度极高，成熟并且广泛使用。Spark 免费提供 Spark Streaming，它使用微批处理进行流式传输。在 Spark2.0 之后，添加了许多优秀的功能（例如对 tungsten、watermarks、event time 处理的支持），同时结构化流也更加抽象，截止本篇博客 Spark 发布的可用版本为 2.4.3，可以在最新版本中在微批处理和连续流模式之间进行切换。

3.1 微批处理 & 连续流处理

DataFunTalk

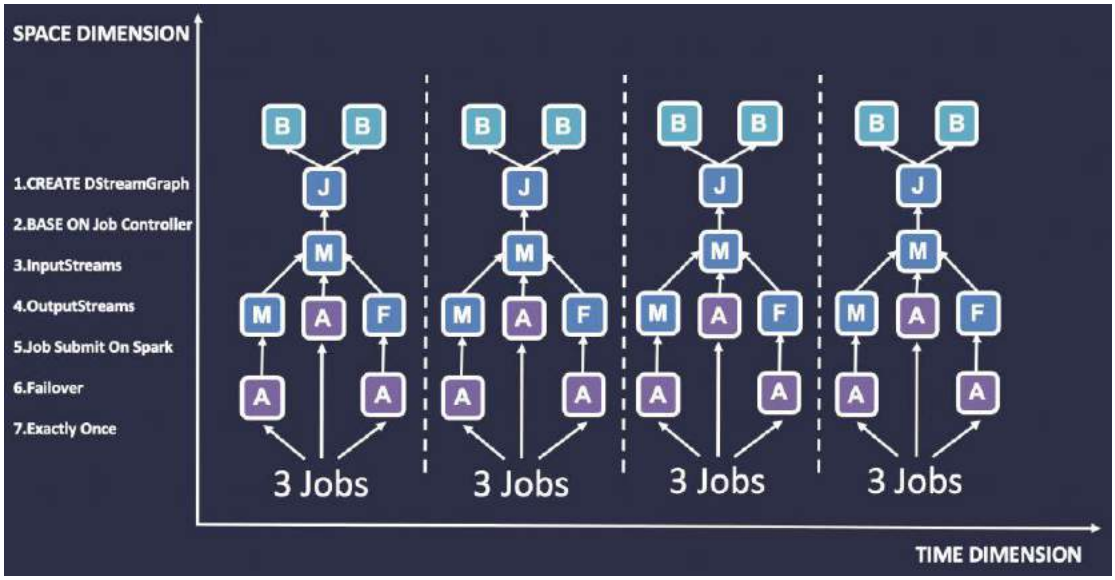
结构化流式传输默认采用微批处理执行，Spark 流式计算引擎会定时检查流数据。在连续流处理中，Spark 不会启动定时任务，而是启动一组长时间运行的任务，这些任务可以连续读取、处理、写入数据。



微批处理中，驱动程序通过将记录 Offset 保存到预写 Log 来检测进度，然后可以使用该 Log 重新进行查询。需要注意的是，在微批处理处理开始之前，需要在下一个微批处理中处理的范围 Offset 保存到 Log 中，以便获取确定性的重新执行和端到端语义。因此，源记录可能需要等待当前的微批处理处理完成，然后记录其 Offset 。

连续流处理中，通过完善和改进算法来检测查询进度，特殊标记的记录被写入到每个任务的输入数据流中。当任务遇到标记时，任务会异步报告处理的最后一个 Offset ，一旦驱动程序收到写入接收器的所有任务的 Offset ，它就会将它们写入预写 Log 中。由于 Checkpoint 完全异步，因此任务可以不间断的继续，并提供一致的毫秒级延时。

3.2 Streaming



对于 Spark Streaming 来说，当不同的数据来源输入进来时，基于固定的时间间隔，会形成一系列固定不变的数据集或者事件集（ 例如 Kafka、Flume 等 ）。这正好和 Spark RDD 基于固定的数据集吻合，从每一个批处理来看，空间维度的 RDD 依赖关系一致，不同的是这 4 个批处理输入的数据规模和数据内容不同，所以生成的 RDD 依赖关系实例不一样。

3.3 优势

列举了 Spark 常见优势，如下所示：

- 支持 Lambda，且在 Spark 中免费使用
- 高吞吐量，适用于不需要子延时的用例

- 容错性，默认使用微批处理
- 高度抽象的 API
- 社区活跃度高
- 支持 Exactly Once

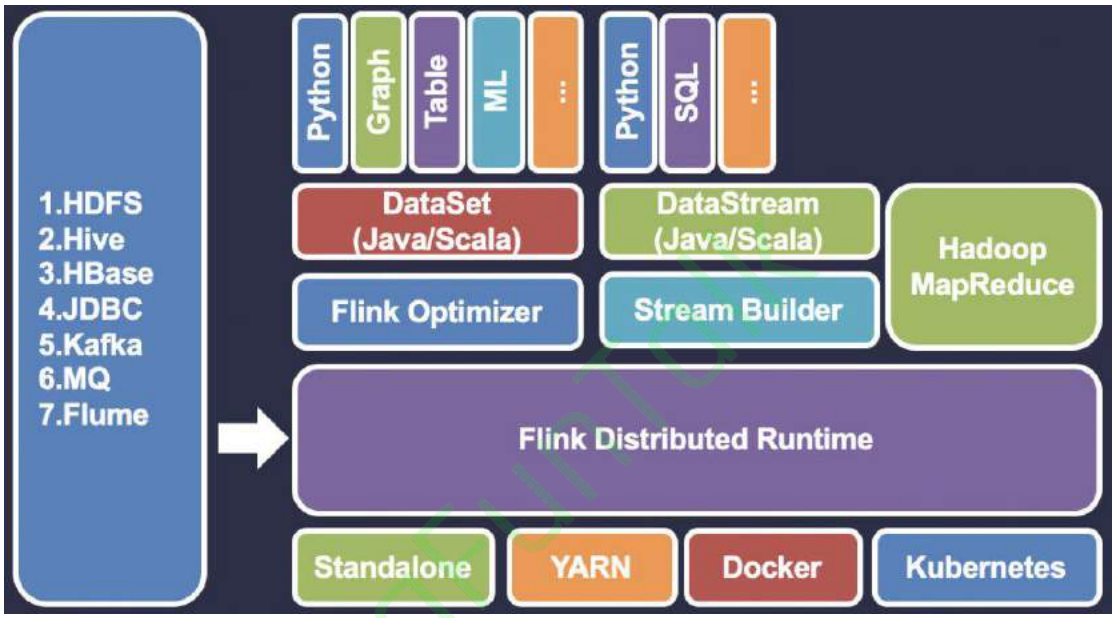
3.4 限制

- 另外，Spark 也有它不足的地方，如下所示：
- 不是真正意义上的实时计算，不能够满足低延时需求
- 需要调整的参数太多，很难做到全面
- 在许多高级功能中落后于 Flink

4. Flink

Flink 也是来自 Spark 类似的学术背景，Spark 来自加州大学伯克利分校，Flink 来自柏林大学。像 Spark 一样，它也支持 Lambda ，但实现与 Spark 完全相反。Flink 本质上是一个真正的实时计算引擎，将批处理作为有限数据流的特殊情况。虽然两个计算框架中的 API 相似，但它们在实现中没有任何相似之处，在 Flink 中，Map、Filter、Reduce 等各个函数实现为长时间运行的运算符（类似于 Storm 中的 Bolt）。

4.1 什么是 Apache Flink？



Flink 是一个开源的实时计算引擎，是实时计算领域的领导者。它拥有出色的图计算和机器学习功能，其底层支持 On YARN 模式，且提供了本地 & 分布式模式，以及 Docker & Kubernetes 等容器部署。

4.2 如何使用 Flink 解决问题？

在低延时场景，需要实时数据，以便能够更快的检测和解决关键事件。例如，在使用 Flink 之前，计算的基本业务指标，实现的延时时间约为 3 到 4 小时，这意味着，如果工程师在早上 10 点左右检测到业务指标变化异常，只能在下午 14 点左右开始排查。如果能够立马解决，则只能在下午 18 左右时来验证解决方案，这样实现起来效率不是很高。

假如你的业务数据是基于时间序列的，那么我们需要使用事件时间来处理在时间窗口内对业务指标进行分组。同时，Flink 也可以很轻松的与存储在 Kafka 和 HDFS 中的业

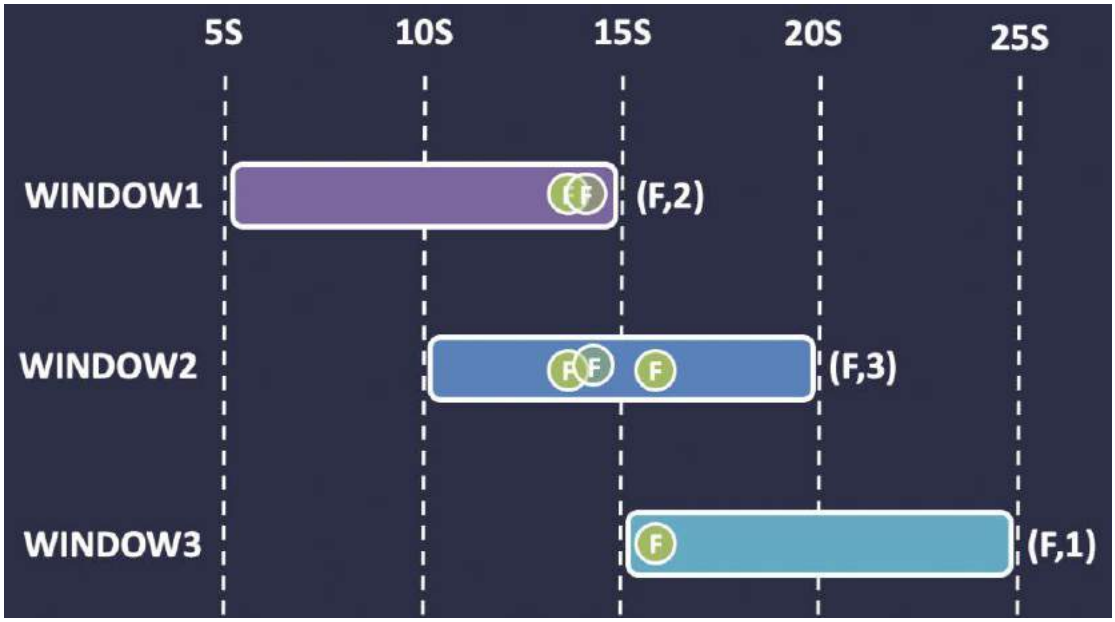
务数据进行集成。另外，Flink 具有良好的非功能特性，便于在生产中运行，易于与不同的监控后端集成（例如 Graphite、Prometheus 等），以及提供良好的 UI 界面。此外，Flink 工作的快速开发周期以及简单的执行模型使得学习曲线平稳，开发效率高。

4.3 什么是窗口和事件时间？

Flink 相比较 Spark Streaming 不仅提供了更低的延时，而且 Flink 还对窗口和事件时间提供了更好的支持。

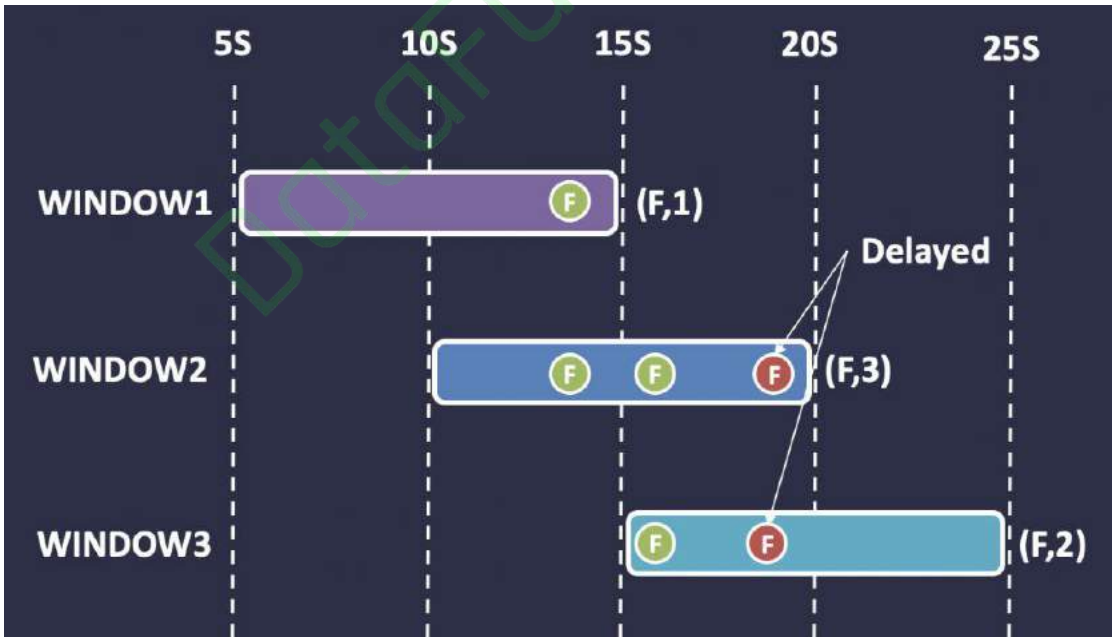
4.3.1 窗口

现实场景中，大部分的数据来源都是无界的，很多情况下，我们会对固定时间间隔的数据进行统计，比如每隔 10 秒统计一下集群服务的 QPS ，此时，窗口机制能够很好的帮助我们实现这类需求。



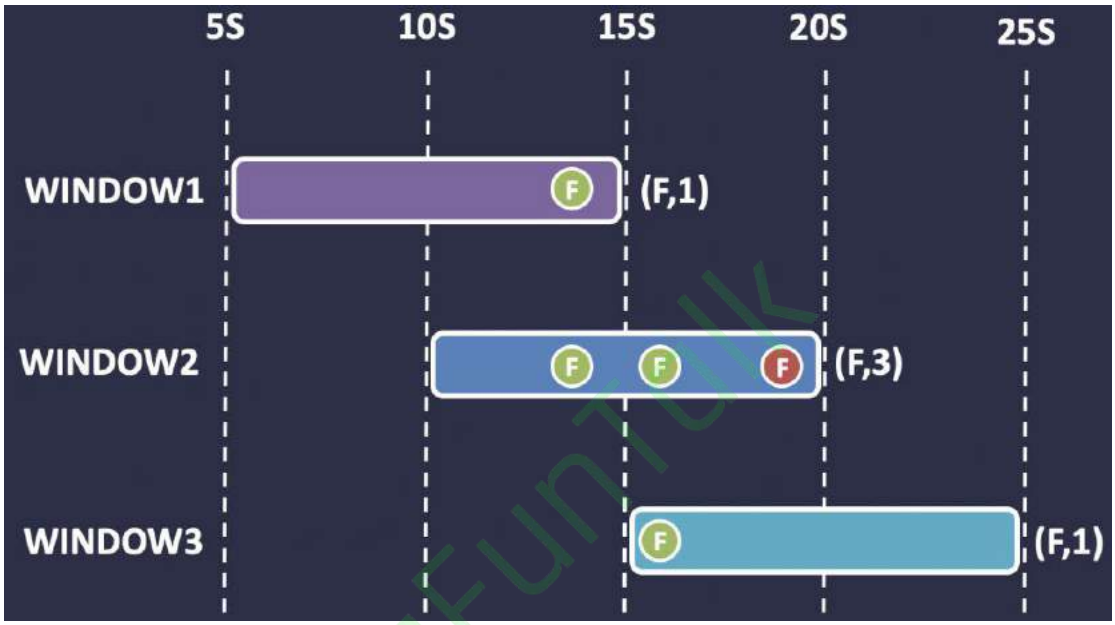
1. 情况一：假设数据源分别在时间 14 秒，第 14 秒和第 16 秒产生消息类型 K 的消息（窗口大小为 10 秒）。这些消息将落入窗口中，如上图所示，在第 14 秒产生的前两个消息将落入窗口 1（5 秒~15 秒）和窗口 2（10 秒~20 秒），第 16 秒产生的第三个消息将落入窗口 2（10 秒~20 秒）和窗口 3（15 秒~25 秒）。每个窗口发出的最终计数分别为（F，2）、（F，3）、（F，1），这是一种理想的状态。

2. 情况二：假设其中一条消息（第 14 秒生产的）由于网络原因到达时延时了 5 秒（第 19 秒到达），那么此时消息在窗口的分布如何呢？延时的消息落入到窗口 2 和窗口 3，因为第 19 秒在 10 秒~20 秒和 15 秒~25 秒这两个窗口。对于窗口 2 来说，计算没有什么问题（因为消息应该落入该窗口），但是它影响了窗口 1 和窗口 3 的结果。



4.3.2 事件时间

现在我们尝试使用事件时间来解决**情况二**的延时问题。要启用事件时间处理，需要一个时间戳提取器，从消息中提取事件时间信息。流式计算按照数据的事件时间来将数据分配到对应的窗口，而不是按照处理数据的时间，处理结果如下图。

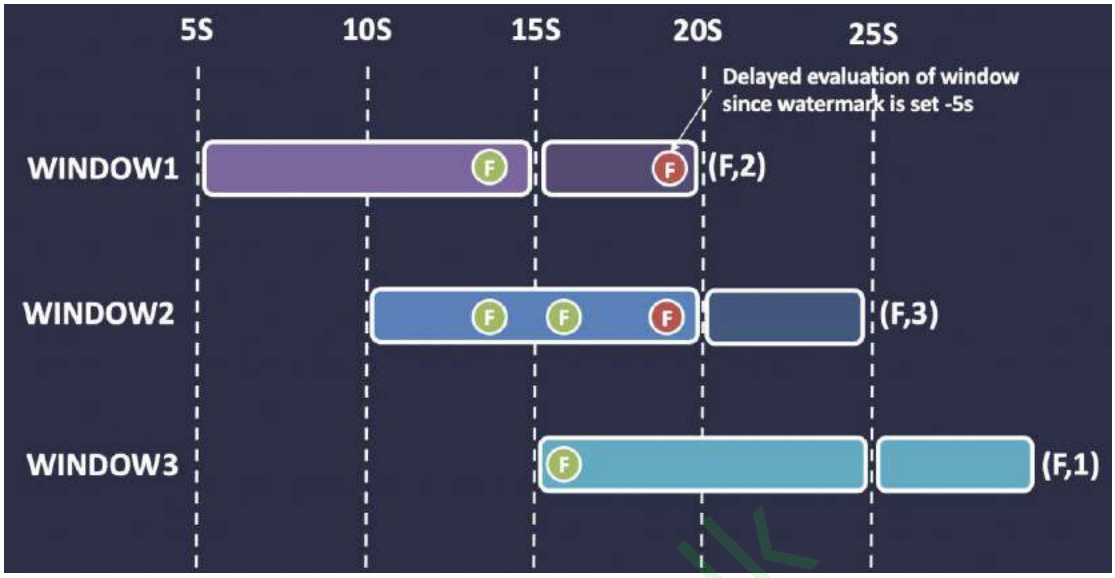


引入事件时间后的结果看起来更好了，窗口 2 和窗口 3 发出了正确的结果，但是窗口 1 仍然是错误的。Flink 没有将延迟的消息分配给窗口 3，因为它现在检查的是消息的事件时间了，并且理解它不在窗口中。但是为什么没有将消息分配给窗口 1 呢？原因在于延迟的消息到达系统时（ 第 19 秒 ），窗口 1 的评估已经完成了（ 15 秒 ）。

4.3.3 水印

为了达到解决**情况二**的问题，达到**情况一**的预期结果。引入水印机制，水印机制可以看作是一种告诉 Flink 一个消息延迟多少的方式。现在将水印设置为当前时间负 5 秒，告诉 Flink 希望消息最多有 5 秒的延迟，这是因为每个窗口在水印通过时被评估。由于

设置的水印时间为当前时间负 5 秒，所以窗口 1（5 秒~15 秒）将在第 20 秒时被评估，以此类推，窗口 2（10 秒~20 秒）将在第 25 秒时进行评估。优化后的结果如下：



最后调整引入水印机制后，得到正确的结果，这 3 个窗口均按照预期的方式发出计数，即（F,2）、（F,3）、（F,1）。

5. 总结（Spark vs Flink）

了解了 Spark 和 Flink 各自的特点后，知道了 Spark Streaming 通过小批量的方式保证了吞吐的情况下，同时提供了 Exactly Once 语义，但是不是严格意义上的实时，而且由于微批处理的方式，对窗口和事件时间的支持比较有限。Flink 采用分布式快照的方式实现了一个高吞吐、低延时，并且支持 Exactly Once 的实时计算引擎，同时 Flink 的实时计算引擎也能更好支持窗口和事件时间。

通过对 Spark 和 Flink 特点的掌握，再结合实际的项目需求、业务场景、以及技术储备，来选取最适合的计算引擎。

DataFunTalk

万亿数据下 Hadoop 的核心竞争力

哥不是小萝莉　编辑：HohXil

1. 前言

在大数据时代，Hadoop 有着得天独厚的优势。然而，每个企业的技术储备和需求特点不同，他们希望从海量的客户数据中挖掘真正的商业价值，像 Google 、Facebook 、Twitter 等这样的企业更是 Hadoop 的最早获益者。那么，今天我们就来聊一聊，万亿数据下 Hadoop 的核心竞争力。

2. 什么是 Hadoop ？

在了解 Hadoop 之前，不得不说的一个名词——“ 大数据 ”。大数据是时代发展和技术进步的产物，大数据的特征如下：

- 庞大的数据容量
- 结构化、半结构化、非结构化的数据类型
- 高效的处理速度
- 高质量的数据

Hadoop 由 Apache 基金会孵化并开源的分布式系统，用户可以在不了解分布式底层设计的情况下，开发分布式应用程序，充分利用集群的优势来进行高速的运算和存储。自从 Hadoop 作为 Apache 基金会开源项目发布以来，它一直备受青睐，这得益于 Hadoop 的可扩展性、低成本、灵活的处理模式等特点。

3. 万亿数据下的难点？

数据量达到万亿规模，这已经是一个很庞大的数据量了。这里难点我们可以分为两种情况，一种是原理上不知道怎么处理，没有具体的实施方案，这个属于技术难题。另一种，虽然有具体的实施方案，也明白其中的原理，但是数据规模太过庞大，这个属于工程上的难点。

数据规模庞大带来的难点主要体现在分布式的要求，因为单个节点不足以在有效的成本和规定的时间内处理完所有的数据。简而言之：

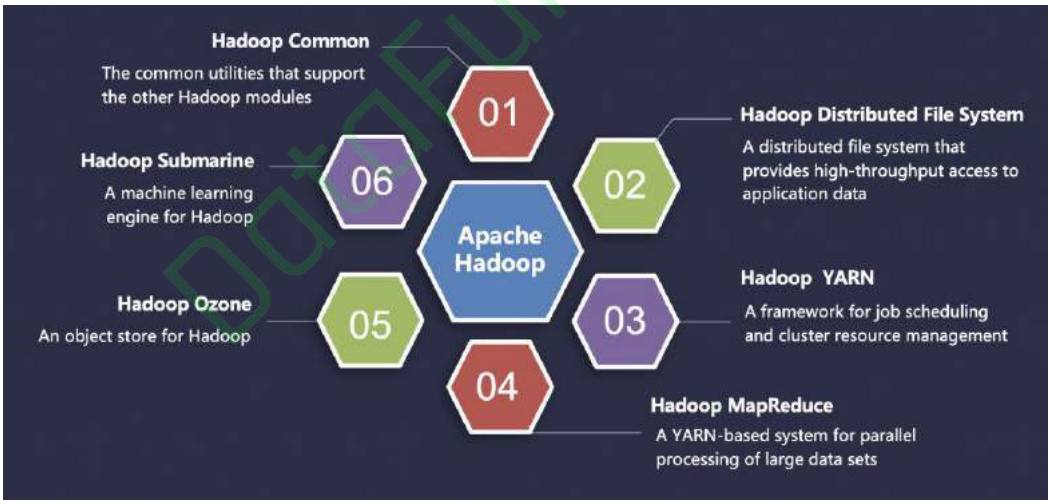
并行化问题：处理数据的应用程序要改造成适合并行的方式；

资源分配管理问题：如何有效的管理提交任务的资源，内存、网络、磁盘等；

容错问题：随着机器数量的增加，可靠性如何保证，例如部分机器硬件出错导致不可用，最终结果的完整性和正确性如何保证。

4. Hadoop 的组成部分有哪些？能做什么？

截止至本篇文章，Hadoop 社区发布了 Hadoop-3.2.0 版本，其核心组成部分包含：基础公共库（Common）、分布式文件存储系统（HDFS）、分布式计算框架（MapReduce）、分布式资源调度与管理系统（YARN）、分布式对象存储框架（OZone）、机器学习引擎（Submarine）。



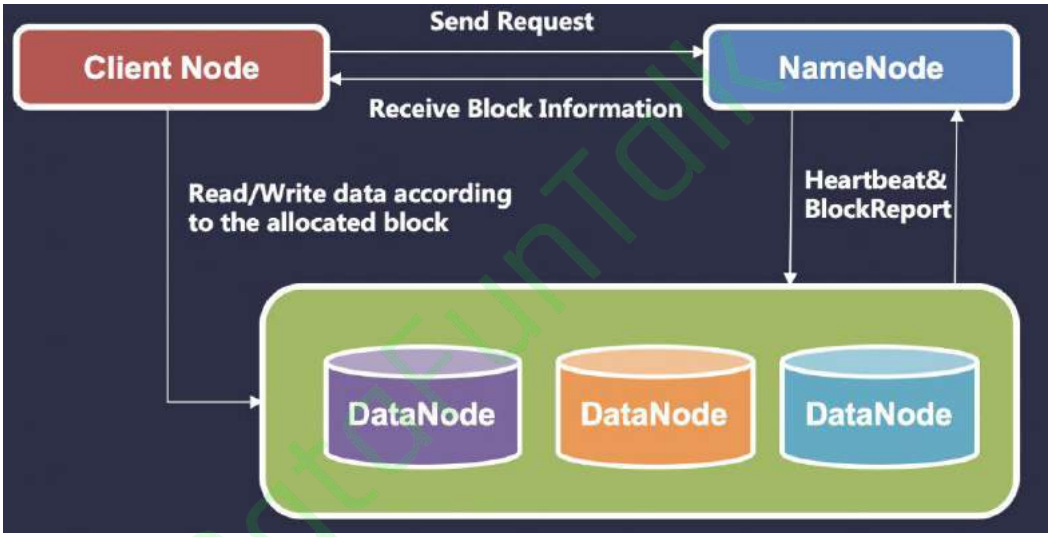
4.1 Hadoop Common

Hadoop Common 属于基础公共库，它是 Hadoop 最底层的一个模块，为 Hadoop 各个子项目提供各种工具，例如配置文件、操作日志等。

4.2 Hadoop Distributed File System

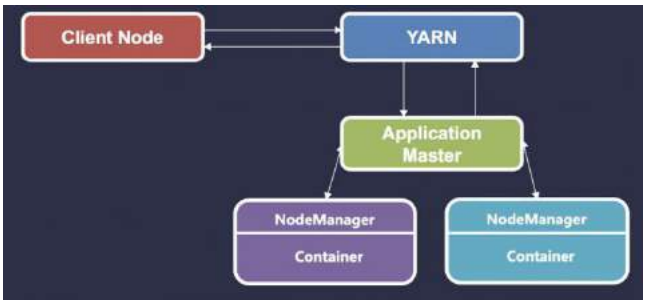
Hadoop Distributed File System 简称 HDFS，它是 Hadoop 的一个分布式文件系统，类似于 Amazon 的 S3 系统，Google 的 GFS 系统。

HDFS 可以处理分布在集群中的大文件，它通过将文件分成数据块来完成此操作。同时，我们可以并行访问分布式数据（例如，在进行数据处理时），各个数据节点进行数据交互形成数据块的副本。



4.3 Hadoop YARN

YARN 是 Hadoop 的一个分布式资源管理框架，可以为上层应用提供统一的资源管理与调度。通过引入 YARN ,Hadoop 集群在资源利用率、资源统一管理、数据共享等方面带来了巨大的好处。



Client 应用提交任务到 YARN ，流程如下：

1. Client 发送请求给 RM

2. ResourceManager (简称 RM) 返回 ApplicationId 给 Client

3. Client 发送 ApplicationId 、QueueName 、用户等信息给 RM

4. RM 寻找合适的 Container ，并将 Client 提交的信息给 NodeManager (简称 NM)

5. 然后在 NM 中启动 AM ，RM 给 AM 分配最大最小资源

6. AM 从 RM 那里获取的可使用资源来申请一些 Container

7. Job 在 Container 中执行，由 AM 返回任务进度，任务执行完成后，AM 向 RM 发送结束任务信息然后退出

4.4 Hadoop MapReduce

MapReduce 是 Hadoop 的一个分布式计算框架，用来处理海量数据。同时，还可以使用 MapReduce 框架来实现一些算法，例如统计单词频率、数据去重、排序、分组等。

4.5 Hadoop OZone

Ozone 是 Hadoop 的可扩展、冗余和分布式对象存储。除了扩展到数十亿不同大小的对象外，OZone 还能在 Kubernetes 和 YARN 等容器环境中有效发挥作用。

可扩展性：OZone 设计之初能够扩展到数百亿个文件和数据块，并且在将来会扩展到更多；

一致性：OZone 是一个强一致性对象存储，它所使用的协议是类似于 RAFT 来实现的；

云集成：OZone 设计之初能够与 YARN 和 Kubernetes 集成使用；

安全性：OZone 能够与 Kerberos 集成，用于控制访问权限，并支持 TDE 和线上加密；

多协议支持：OZone 能够支持不同的协议，例如 S3、HDFS；

高可用：OZone 是一个多副本系统，用于保证数据高可用性。

4.6 Hadoop Submarine

Submarine 是一个允许基础设施工程师 / 数据科学家在资源管理平台（如 YARN）上运行深度学习应用程序（Tensorflow ，Pytorch 等）的项目。

在已有集群运行：Submarine 支持在 YARN 、Kubernetes 或者其他类似的调度框架中使用；

支持多种框架：Submarine 支持多种机器学习框架，例如 TensorFlow 、Pytorch 、MxNet 等；

覆盖整个 ML：Submarine 不仅仅是一个机器学习引擎，它涵盖了整个机器学习过程，例如算法开发、模型批量训练、模型增量训练、模型在线服务和模型管理。

5. Hadoop 的核心竞争力在哪？

Hadoop 如此受人喜欢，很大程度上取决于用户对大数据存储、管理和分析需求的迫切。大数据是目前很多企业面临的一个挑战，由于数据量的庞大、数据类型的复杂 ，特别是非结构化或者半结构化的数据远远多于结构化的数据，一些传统的基于关系型数据库的存储和分析难以满足时，且关系型数据库巨大成本压力也是很多企业考虑的问题，而 Hadoop 给人们提供了解决大数据问题的技术手段。

大数据时代需要 Hadoop ，那么 Hadoop 的核心竞争力在哪呢？

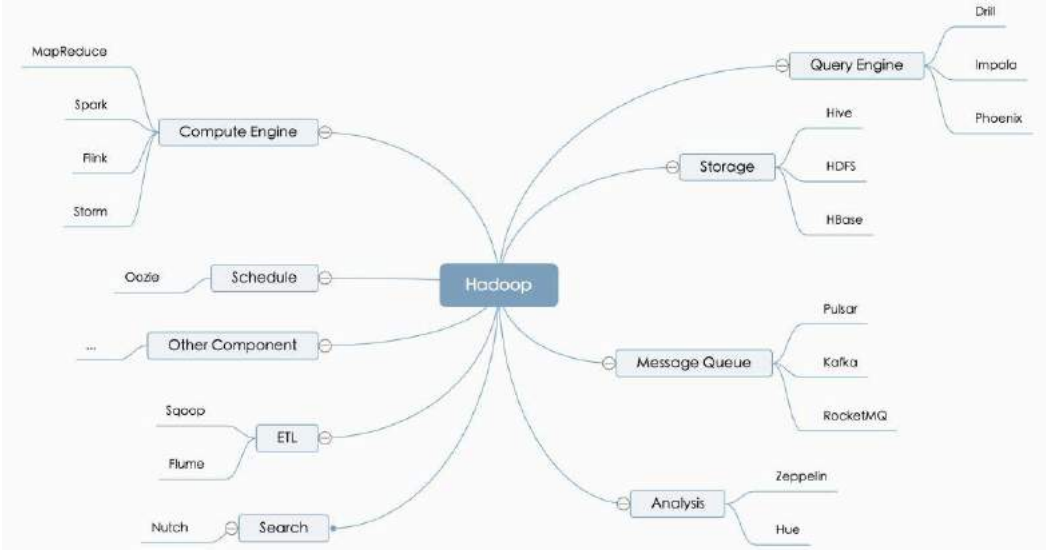
5.1 降低大数据成本

Hadoop 使企业可以高效的管理数据，以降低数据成本，其中包含业务成本、硬件成本、人工成本、存储成本等。通过易用性、权威性、时效性等特性，Hadoop 还可以帮助用户增加数据价值。目前 Hadoop 社区的支持，以及各大 Hadoop 厂商的支持，使得 Hadoop 从一个单独的开源软件逐步演变成一个具有一定规模的生态系统，这些厂商包含 Cloudera 、MapR 、Hortonworks 等，他们在这一生态系统中扮演着不同的角色，例如有系统厂商、监控服务商、数据分析商等。

而使用者可以从这些厂商中提供的系统来简化 Hadoop 的学习成本，快速构建符合自身要求的大数据平台，同时合理利用厂商提供的附属组件来开发出高效、易用的的大数据应用。

5.2 成熟的 Hadoop 生态圈

Hadoop 不是一个“孤岛”系统，它拥有成熟的 Hadoop 生态圈。



利用 Hadoop 生态圈设计满足自身需求的方案，需要考虑一些关键要素：

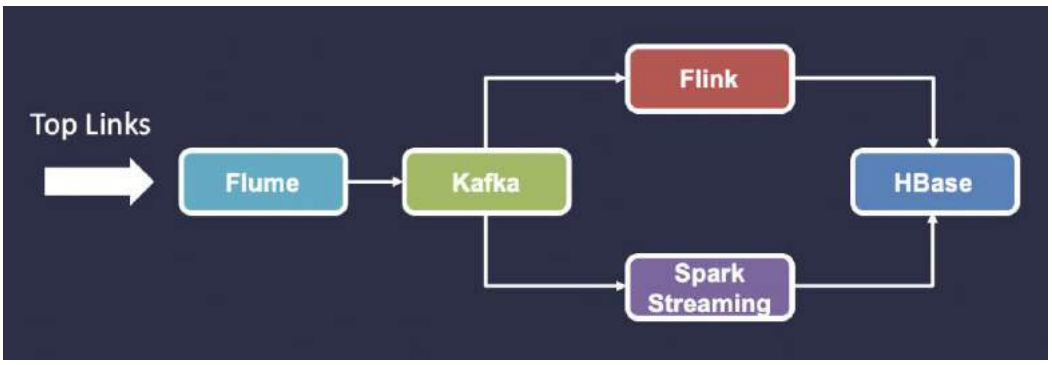
1. 从需求的最终结果开始分析，而不是从可用的工具开始。例如，可用性、一致性等；

2. 对数据处理时效性的评估，例如离线任务（MapReduce 、Hive）、实时任务（Flink、Spark Streaming）；

3. 尽可能使用成熟的方案。

案例一：获取最后一小时的热门链接

将热门链接集中收集，使用 Flume 将链接发送到 Kafka ，然后使用 Flink 或者 Spark Streaming 计算引擎在 1 小时的窗口内分析数据，最后将计算后的结果写入到 HBase 进行存储。



案例二：为用户推荐电影

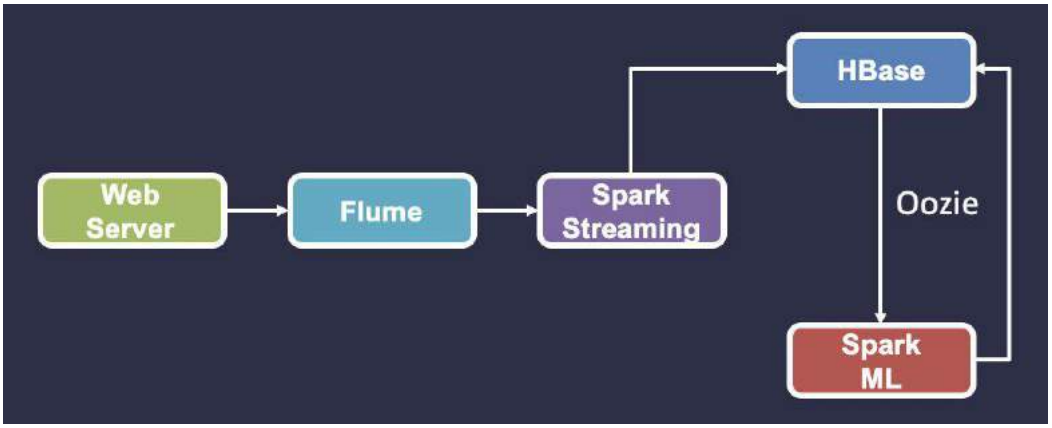
这是一个实时场景，用户喜欢电影，那么用户应立即看到相关电影。

解决思路：每次用户给出评级时，计算建议都是包含权重的，因此我们应该定期根据现有用户行为计算建议。根据对用户行为的理解，可以为给定用户预测所有电影的推荐，然后对其进行排序，并过滤用户已经开过的内容。

组件选取：数据库可以使用 NoSQL 数据库，例如 HBase 。来存储用户评级。计算引擎方面可以选择 Flink 或者 Spark ML 通过 Oozie 定

时调度来重新计算用户电影推荐。然后，使用 Flume 和 Spark Streaming 用于流式传输和处理实时用户行为。

工作流程：Web 服务器将用户评级发送给 Flume ，后者将其传递给 Spark Streaming ，然后将结果保存到 HBase 中。接着，使用 Oozie 定时调度执行 Spark ML 应用来重新计算电影推荐并将结果保存到 HBase 中。

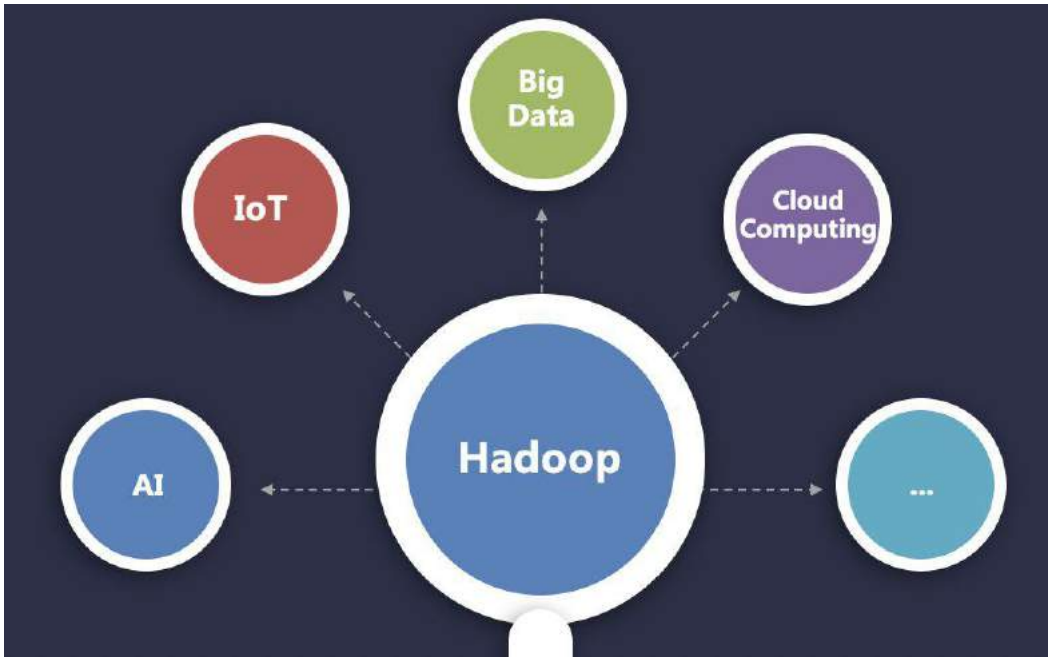


6. 是否一定要选择 Hadoop ？

与传统数据库系统相比较，开源的 Hadoop 有自己的优势。尤其是 Hadoop 既能处理关系型数据库中的结构化数据，也能处理视频、音频、图片等非结构化数据。并且 Hadoop 还能够根据数据的规模和问题的复杂度轻松的扩展。那是不是一定要用 Hadoop ？

每个企业都有自己的特殊需求，都有自己的技能栈，如果已经购买了成熟的数据库产品，没有必要舍弃这些产品，要确保对 Hadoop 足够的了解，不要盲目的“ 跟风 ”。

然而，Hadoop 是解决大数据的一种技术手段，这个是一个趋势，例如 Hadoop 与 AI 、IoT 等领域的结合使用。了解和掌握 Hadoop 是有所必要的，可以从一些小的项目尝试积累更多经验。



Apache Doris：一个开源 MPP 数据库的架构与实践

嘉宾：赵纯　编辑：王吉东

分享提纲：

- Doris 背景介绍
- 适用场景 & 案例介绍
- Doris 整体架构
- Doris 关键技术

Doris 背景介绍

介绍 Doris 的整体架构，以及 Doris 的一些特性。

一、Doris

Doris 是分布式、面向交互式查询的分布式数据库，主要部分是 SQL，内部用到 MPP 技术。

什么是 MPP?

MPP（Massively Parallel Processing），即大规模并行处理，在数据库非共享集群中，每个节点都有独立的磁盘存储系统和内存系统，业务数据根据数据库模型和应用特点划分到各个节点上，每台数据节点通过专用网络或者商业通用网络互相连接，彼此协同计算，作为整体提供数据库服务。非共享数据库集群有完全的可伸缩性、高可用、高性能、优秀的性价比、资源共享等优势。简单来说，MPP 是将任务并行的分散到多个服务器和节点上，在每个节点上计算完成后，将各自部分的结果汇总在一起得到最终的结果（与 Hadoop 相似）。

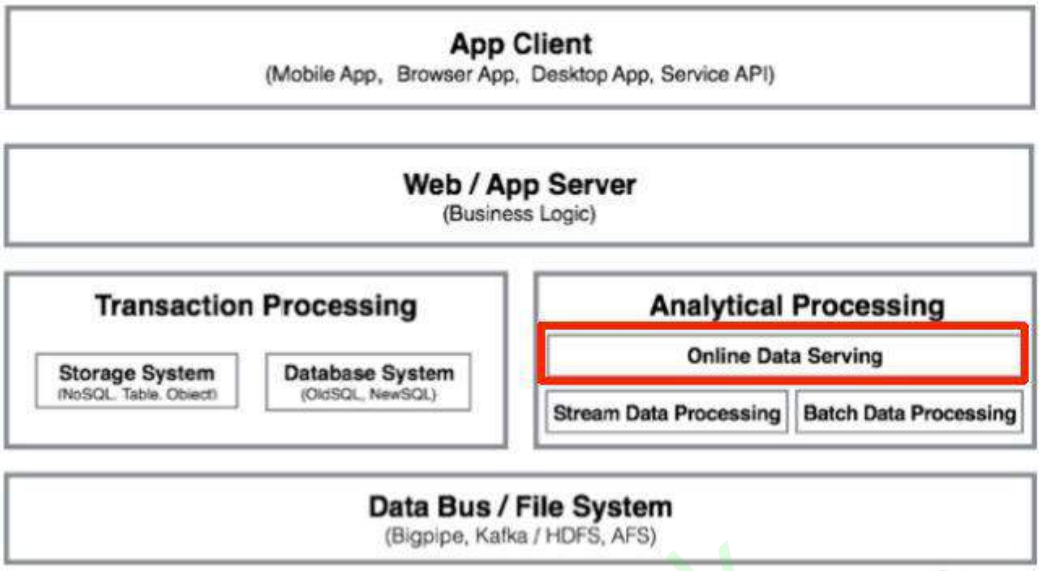
Doris 主要解决 PB 级别的数据量（如果高于 PB 级别，不推荐使用 Doris 解决，可以考虑用 Hive 等工具），解决结构化数据，查询时间一般在秒级或毫秒级。

Doris 由百度大数据部研发（之前叫百度 Palo，2018 年贡献到 Apache 社区后，更名为 Doris），在百度内部，有超过 200 个产品线在使用，部署机器超过 1000 台，单一业务最大可达到上百 TB。

百度将 Doris 贡献给 Apache 社区之后，许多外部用户也成为了 Doris 的使用者，例如新浪微博，美团，小米等著名企业。

二、Doris 定位

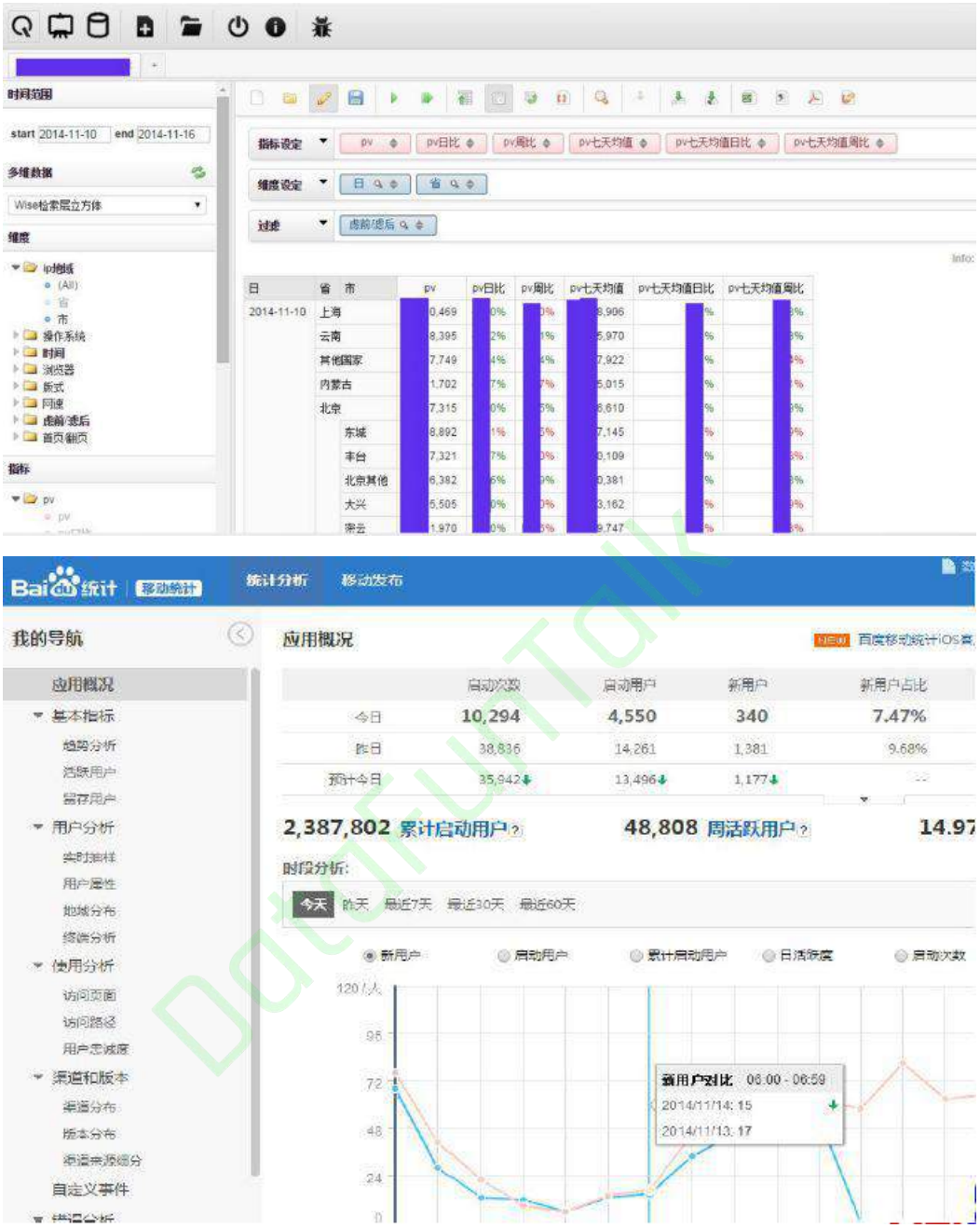
在数据分析处理框架中，Doris 主要做的是 Online 层面的数据服务，主要处理的是数据分析方面的服务。



Doris 的目标是：实现低成本（主要针对商业产品），可线性扩展，支持云化部署，高可用，高查询性能，高加载性能。

简要介绍两个百度内部业务中 Doris 的应用案例：

A. 百度统计在线报表



B. 百度另一内部业务的多维分析

可以手动拖拽维度、指标等，进行查询。

适用场景 & 案例介绍

一、适用场景

1、对数据分析、统计

数据分析大体上可以分为两大类场景：一种偏向于报表类的，另一种偏向于多维分析的。

2、报表

报表类数据分析，数据分析以及查询的模式相对比较固定，而且后台 SQL 的模式往往都是确定的。针对此类应用场景，选择使用 MySQL 存结果数据，用户可从界面选择执行批处理以及发送邮件。在 Doris 平台中，报表类查询时延一般在秒级以下。

3、多维分析

这里提到的多维分析，同样要求数据是结构化的，适用于查询相对灵活的场景，例如数据分析条件以及聚合维度等方面不是很确定，一般将此类数据分析定义为多维分析。相对于报表类分析，多维分析的查询时延会稍慢，大约在会在 10s 的级别。

二、具体案例

案例分析 1：百度统计

百度统计，为网站站长提供流量分析，网站分析，受众分析等多种分析服务。服务网站数量超过 450W，每天查询量达到 1500W，QPS（Queries Per Second，每秒查询率 ）峰值超过 1400，每日新增数据量超过 2TB；数据导入频次为 5 分钟，平均查询时延 30ms。

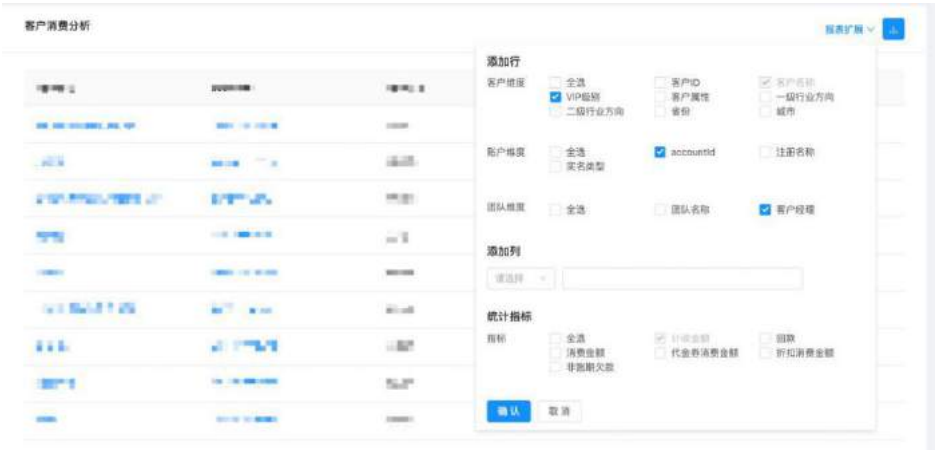
案例分析 2：百度云系统

1、百度云交易系统

百度云交易系统，主要提供订单、账单、扣费、交易流水等 TB 级别量数据的存储和实时查询服务，数据量约 12TB，每 5 分钟导入。

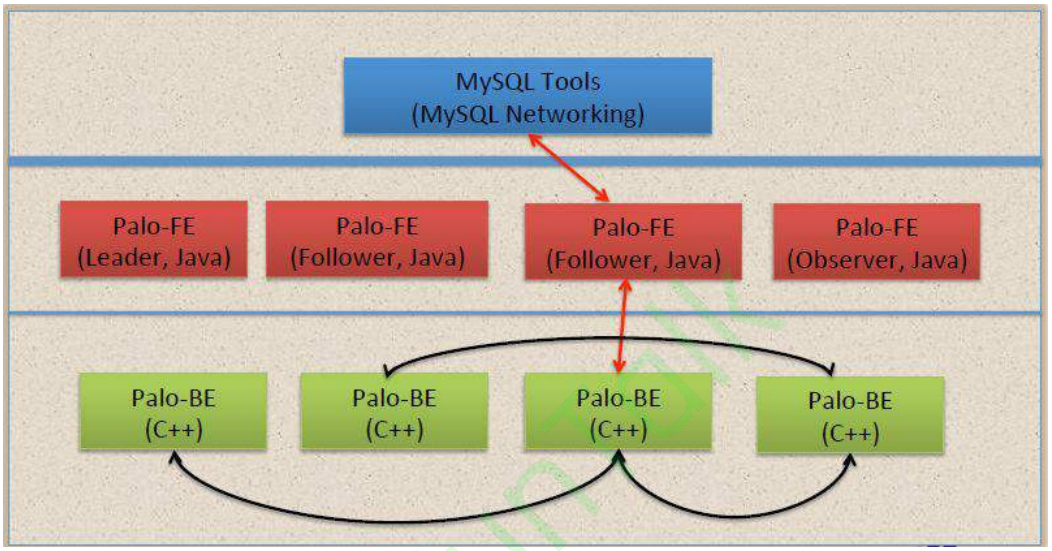
2、百度云数据中心

百度云数据中心，主要提供百度云经营分析，产品分析，用户分析等多种分析服务；提供多维度（100+）分析，具备高性能（ 秒级 ）BI 能力；百度云数据中心的日处理数据量约 1T（ 略少于百度统计 ,因此查询效率略高于百度统计 ），分钟级导入。以下是百度云数据中心页面的一个截图：



Doris 整体架构

一、Doris 整体架构

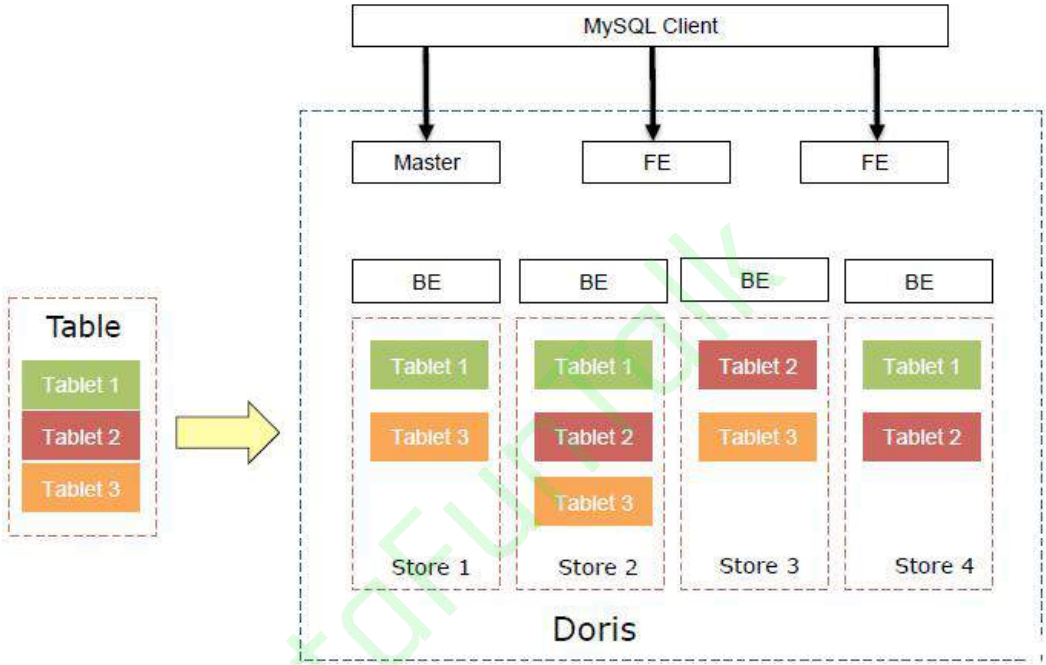


Doris 的整体架构和 TiDB 类似，借助 MySQL 协议，用户使用任意 MySQL 的 ODBC/JDBC 以及 MySQL 的客户端，都可以直接访问 Doris。Doris 中的模块包括 FE 和 BE 两类：FE 主要负责元数据的管理、存储，以及查询的解析等；一个用户请求经过 FE 解析、规划后，具体的执行计划

会发送给 BE，BE 则会完成查询的具体执行。BE 节点主要负责数据的存储、以及查询计划的执行。目前平台的 FE 部分主要使用 Java，BE 部分主要使用 C++。

二、Doris 数据分布

如果从表的角度来看数据结构，用户的一张 Table 会拆成多个 Tablet，Tablet 会存成多副本，存储在不同的 BE 中，从而保证数据的高可用和高可靠。



三、Doris 的使用方式

Doris 的使用方式和 MySQL 类似，创建 database，创建 table，导入数据、执行查询等。具体详见下图：

```
./mysql -h PALO_FE_HOST -P PALO_FE_PORT -uYOUR_USERNAME -pYOUR_PASSWORD

CREATE DATABASE example_db;

USE example_db;

CREATE TABLE ps_stats_tbl (
    siteid    INT,          DEFAULT '10',
    day       DATETIME,
    citycode SMALLINT,
    username VARCHAR(32) DEFAULT '',
    pv        BIGINT   SUM DEFAULT '100'
) DISTRIBUTED BY HASH(siteid) BUCKETS 32;

LOAD LABEL ps_stats_20150717 (
    DATA INFILE("hdfs://host:port/ps_stats_data")
    INTO TABLE ps_stats_tbl
);

SHOW LOAD WHERE LABEL = "ps_stats_20150717";

SELECT siteid, sum(pv) FROM ps_stats_tbl WHERE day = "2015-07-17" GROUP BY siteid;
+-----------+----------+
| siteid    | sum(pv) |
+-----------+----------+
| 23143     |   114996 |
| 12345     |   318925 |
+-----------+----------+
2 rows in set (0.02 sec)
```

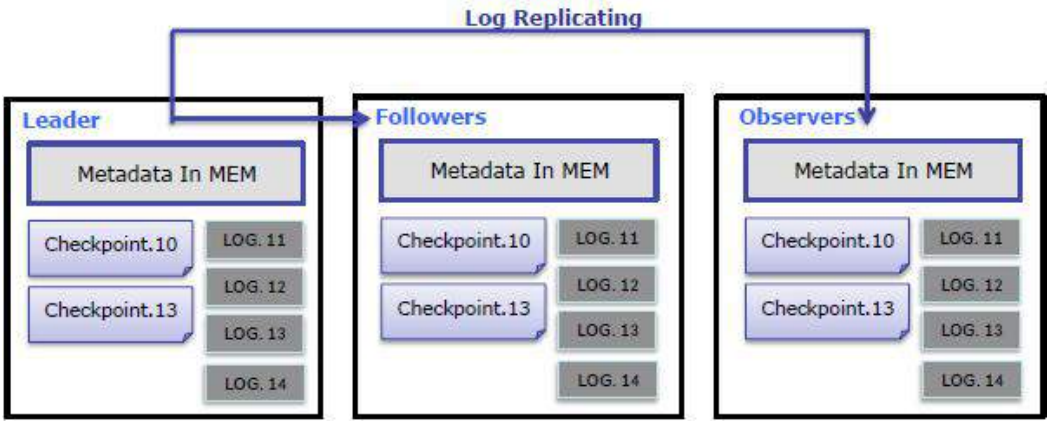
Doris 关键技术

一、数据可靠性

1、元数据使用 Memory+Checkpoint+Journal（分别是什么？），使用 BTBJE（类似于 Raft）协议实现高可用性和高可靠性。

2、Doris 内部自行管理数据的多副本和自动修复。保证数据的高可用、高可靠。在服务器宕机的情况下，服务依然可用，数据也不会丢失。

DataFunTalk



二、易运维

无外部依赖：

- Doris 部署无外部依赖，只需要部署 BE 和 IBE 即可搭建起一个集群。
- 支持 Online Schema Change
- 支持在线更改表模式（ 加减列，创建 Rollup），不会影响当前服务，不会阻塞读、写等操作；这种执行是异步的（ 用户不需要一直盯在那里 ）

数据库同步操作和异步操作：

同步，是所有的操作都做完，才返回给用户结果；即写完数据库之后，再响应用户，用户体验不好；

异步，不用等所有操作等做完，就相应用户请求；即先相应用户请求，然后慢慢去写数据库，用户体验较好。缓存机制（也就是消息队列），就是异步操作的一个典型应用。

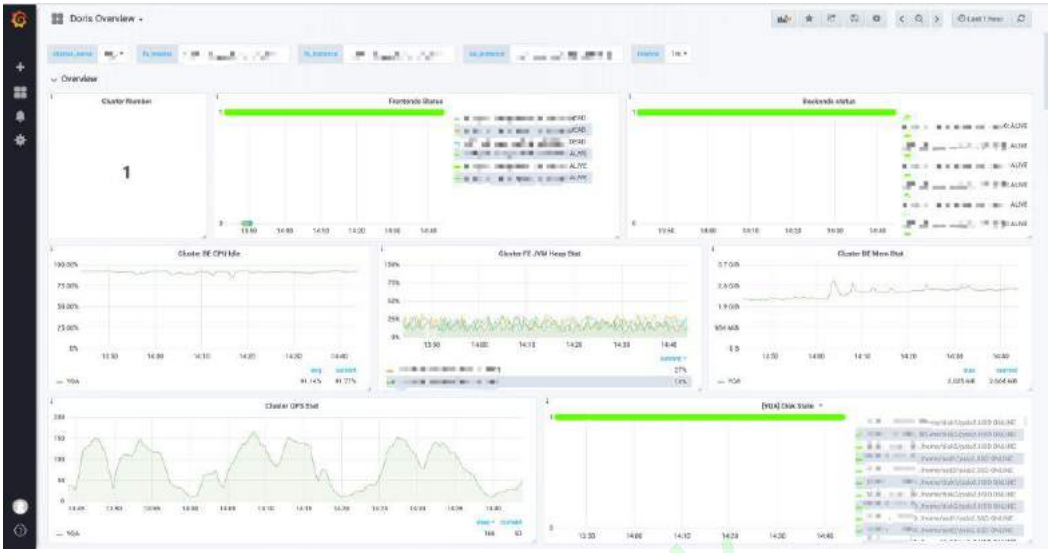
1、副本自动均衡

传统数据库的扩（缩）容比较麻烦，有时甚至需要重做数据；而 Doris 数据库只需要一条 SQL（ 无须额外操作 ）即可实现扩（缩）容。

2、内置监控

使用 Prometheus、Grafana 将监控项指标列出。

下图为 Doris 默认配置面板：



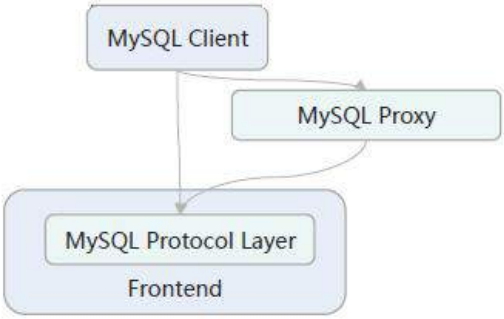
三、MySQL 兼容性

MySQL 的兼容性从两方面体现：

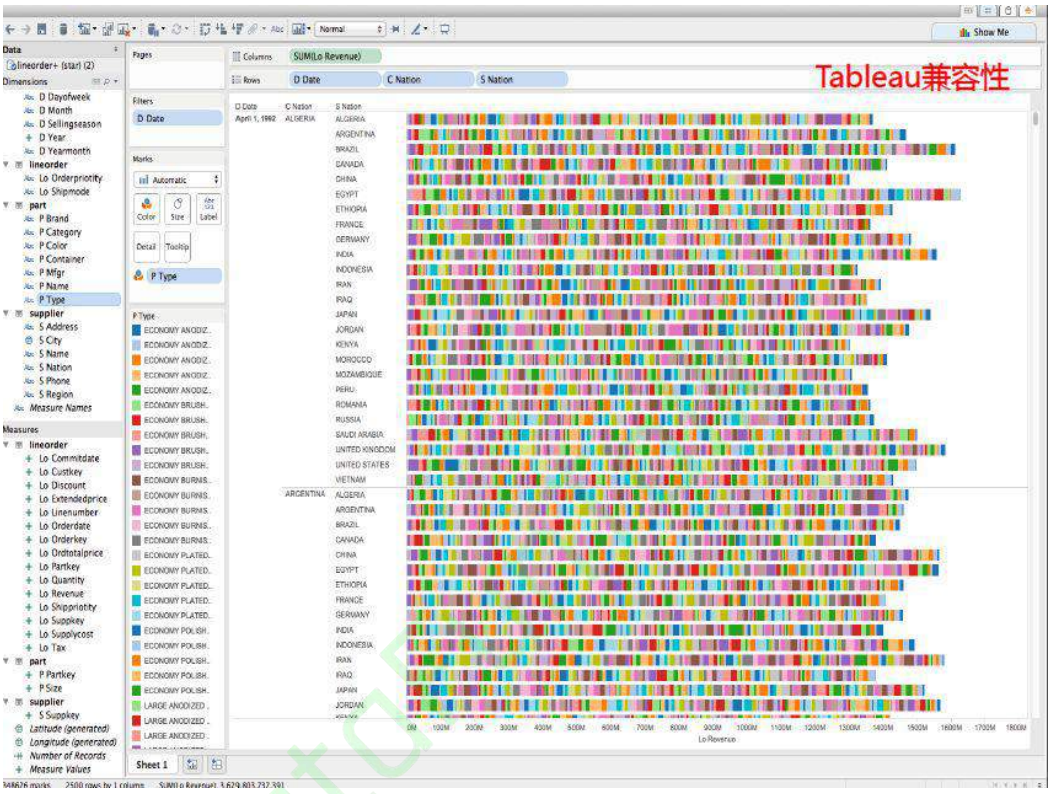
1、兼容 MySQL 的网络协议（MySQL Network Protocol）

2、兼容 MySQL 语法，使用 MySQL 语法可对 Doris 数据库进行查询

- 关于 MySQL 语法的兼容性，前文已经描述过，这里不再赘述；
- 关于 MySQL 的网络协议的兼容性，举个简单的例子：MySQL 的调度服务器 Proxy，可以直接用作 Doris 的 Proxy。



Doris 的前端展示，可以使用 MySQL 专属展示器 Tableu。下图就是使用 Tableu 将 Doris 数据可视化的一个范例。



Doris 与 R 语言可以实现无缝对接，用 R 语言可直接操作 Doris 数据库，进行数据分析、数据挖掘等工作。

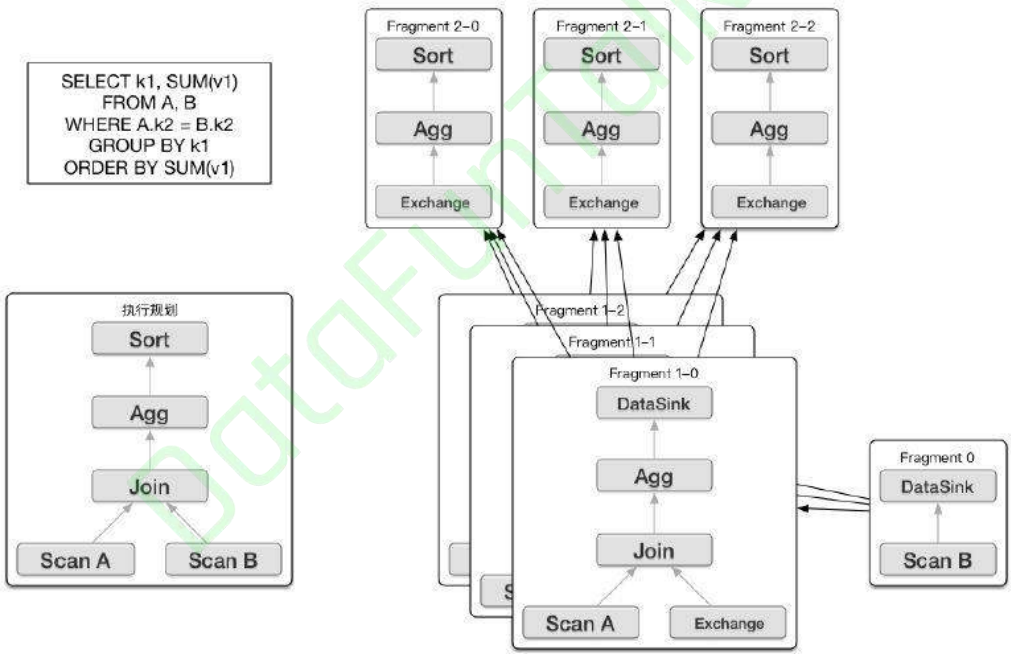
四、支持 MPP

MPP 即 Massively Parallel Processing，大规模并行处理，即海量数据并发查询。以下图为例：

执行

```
SELECT k1,SUM(v1) FROM A,B WHERE A.k2=B.k2 GROUP BY k1 ORDER BY SUM(v1)
```

语句，该语句包含了合并、聚合计算、排序等多种操作；在执行计划的时候，MPP 将其拆分成多份，分布到每台机器执行，最后再将结果汇总。假如有 10 台机器，在大数据量下，这种查询执行方式可以使得查询性能达到 10 倍的提升。



Doris 数据模型

一、Doris 数据模型特点

1、键值对存储形式：

类似于字典搜索查询的键值对格式，Doris 中所有数据分成两列：Key 列和 Value 列。如下图所示，Time、Id、Country 列共同组成 Key 列，Clicks、

Cost 列为 Value 列。Key 列有序可进行快速查找，Value 列可以按照具体聚合类型内部完成数据聚合。

Time	Id	Country	Clicks	Cost
2013/12/31	1	US	10	32
2014/01/01	2	UK	40	20
2014/01/01	2	US	150	80

2、Key 列全局有序排列，查询时方便快速定位查找。

Doris 数据模型的一个显著特点是 Key 列全局唯一，因此存在相同 Key 值的不同 Value，则后面的数据与前面的数据自动做（SUM，MIN，MAX，REPLACE）等聚合处理。例如，下图中绿色方框中的两行，相同的 Key 值对应不同的 Value；因此，新的 Value 到达后，与前数据作 SUM 处理，得到最新的数据，不仅提升效率，还可提高数据处理的准确性。

Time	Id	Country	Clicks	Cost
2013/12/31	1	US	10	32
2014/01/01	2	UK	40	20
2014/01/01	2	US	150	80

+

Time	Id	Country	Clicks	Cost
2014/01/01	1	US	5	3
2014/01/01	2	UK	60	30
2014/01/01	2	US	50	20

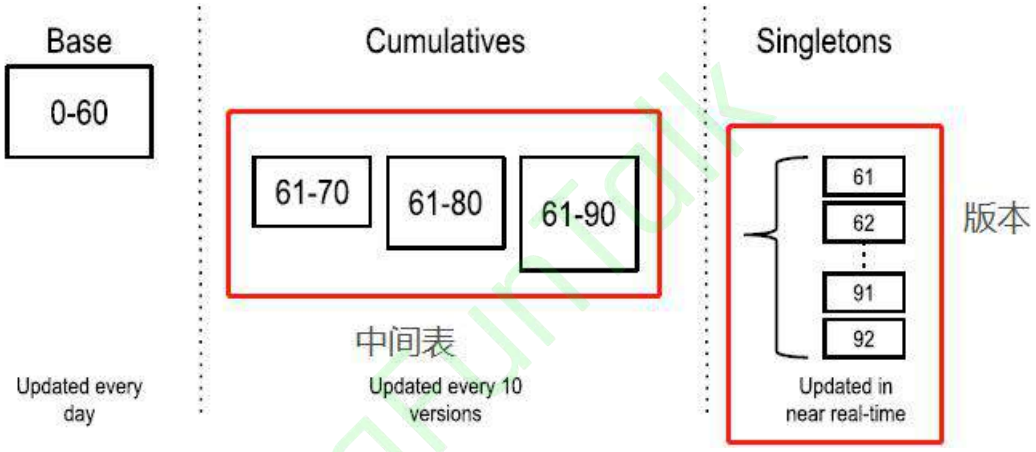
=

Time	Id	Country	Clicks	Cost
2013/12/31	1	US	10	32
2014/01/01	1	US	+5	+3
2014/01/01	2	UK	40+60	20+30
2014/01/01	2	US	150+50	80+20

二、聚合计算说明：

本部分具体描述相同 Key 值下 Value 值的聚合。

数据的导入是分版本的（例如下图右侧 91，92 版本），每一个版本之间 Key 相同的数据汇总到中间表中，通过聚合处理最终输出为左侧的 base。



三、按列存储

1、Doris 的数据是按列存储的，每一列单独存放。

2、查询时，只访问查询涉及的列，大量降低 I/O。

3、数据类型一致，方便压缩。

4、数据包建索引，数据即索引。

5、利用原始过滤条件以及 min、max 和 sum 等智能索引技术，将数据集查询范围尽可能地缩小，大大减少 I/O，提升查询效率。

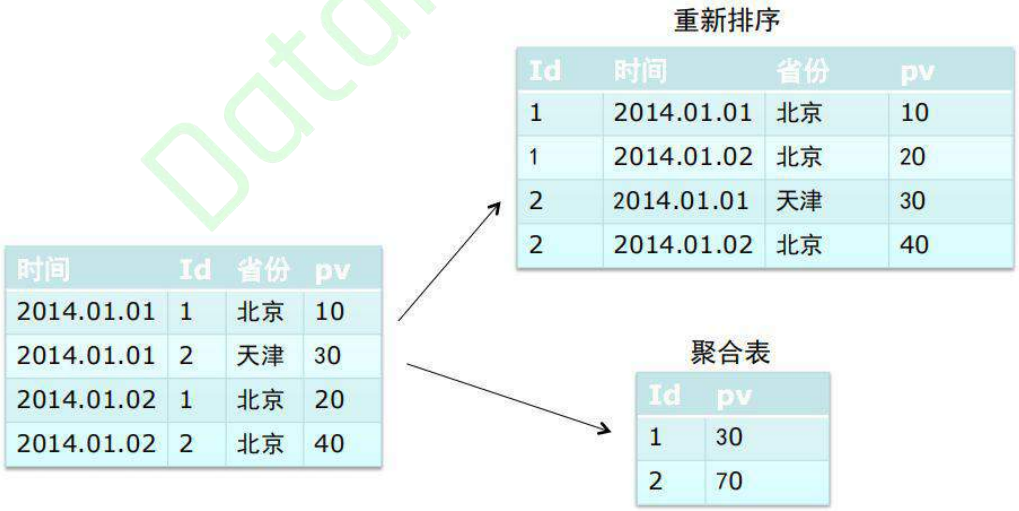
四、物化视图

物化视图是提取某些维度的组合建立对用户透明的却有真实数据的视图表格。Doris 的物化视图可以保证用户在更新时，直接更新原始表，Doris 会保证原表、物化视图原子生效。在查询的时候用户也只需指定原始表，Doris 会根据查询的具体条件，选择适合的物化视图完成查询。

通常用户可以通过物化视图功能完成以下两种功能。

1、更换索引列进行重排列

2、针对指定列做聚合查询



五、两层分区与分级存储

两层分区：

1、方便新旧数据分离，使用不同的存储介质（例如新数据使用 SSD，历史数据 SATA）

2、分区减少了大量历史数据不必要的重复 BE/CE，节省了大量的 IO 和 CPU 开销

3、两层分区的方法简化了表的扩容，便于 shard 调整（例如，前期不必建立过多 shard，后期随着业务增长客随时调整 shard 数）

分级存储

用户可以指定数据放到 SSD 上或者 SATA 盘上，也支持根据 TTL 将冷数据从 SSD 迁移到 SATA 上，高效利用 SSD 提高查询性能。

```
CREATE TABLE example_tbl (
    k1 DATE,
    k2 INT,
    v1 VARCHAR(2048) REPLACE,
) PARTITION BY RANGE (k1) (
    PARTITION p1 VALUES LESS THAN ("2014-01-01")
        properties ("storage_media"="ssd", "storage_cooldown"="2015-06-01 10:00:00"),
    PARTITION p2 VALUES LESS THAN ("2014-06-01")
        properties ("storage_media"="ssd"),
    PARTITION p3 VALUES LESS THAN ("2014-12-01")
        properties ("storage_media"="hdd"),
) DISTRIBUTED BY HASH(k2) BUCKETS 32;

```

六、Doris 在 Elasticsearch 的应用

简介：

1、ES 的优点是索引，可支持多列索引，甚至可支持全文语义索引（如 term，match，fuzzy 等）；然而其缺点是没有分布式计算引擎，不支持 join 等操作

2、与 ES 相反，Palo 具备丰富的 SQL 计算能力，以及分布式查询能力；然而其索引性能较低，不支持全文索引。

3、Doris 在 ES 开发的过程中，分别借鉴 ES 和 Palo 的长处，支持了 Elasticsearch 多表 Join 操作，同时引入 Elasticsearch 的语义搜索功能，扩充了 Doris 的查询能力。

使用方式：

第一步：建立一张 ES 的外部表。

```
CREATE EXTERNAL TABLE `doris` (
  `id` bigint(20) COMMENT "",
  `title` varchar,
  `content` varchar(200)
) ENGINE=ELASTICSEARCH
PROPERTIES (
"hosts" = "127.0.0.1:8200",
"user" = "root",
"password" = "root",
"index" = " doris",
"type" = "doc"
);
```

第二步：在 ES 外部表中导入一些数据：

```
POST index/doc/_bulk
{"id":1,"title":"aaa","content":"doris elasticsearch term query"}
{"id":2,"title":"bbb","content":"doris fuzzy query"}
{"id":3,"title":"ccc","content":"elasticsearch lucene"}
{"id":4,"title":"ddd","content":"kudu similar with doris"}
{"id":5,"title":"eee","content":"grenplum is a OLAP Database"}
```

第三步：使用和 ES 一样的搜索语句，进行全文检索查询：

```
mysql> select * from doris where esquery('content', '{
    '>         "match": {
    '>           "content": "doris elasticsearch"
    '>         }
    '>     }');
+------+-------+---------------------------------+
| id   | title | content                         |
+------+-------+---------------------------------+
|    1 | aaa   | doris elasticsearch term query  |
|    2 | bbb   | doris fuzzy query               |
|    3 | ccc   | elasticsearch lucene            |
|    4 | ddd   | kudu similar with doris         |
+------+-------+---------------------------------+
4 rows in set (0.03 sec)
```

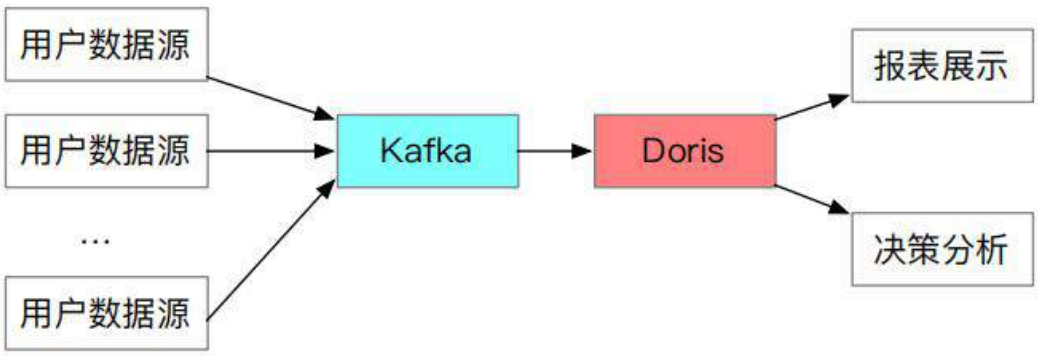
类似于上图这样的搜索语句，在 SQL 中比较难以表达，但是在 ES 中较容易实现。

七、Kafka 消息队列加载

1、Doris 内部支持订阅 Kafka 数据流，实现直接对接 Kafka：

```
CREATE ROUTINE LOAD my_job ON db1.tbl1
COLUMNS DETERMINATED BY ','
COLUMNS (k1, k2, v1, v2, v3 = day(k1))
FROM KAFKA
(
    "kafka_broker_list" = "xxx",
    "kafka_topic" = "topic"
)
```

2、用户数据源经 Kafka 消息队列收集后，可以依次进入到 Doris 中，通过 Doris 做报表展示和决策分析等工作。



3、优点

- 无需额外组件，用户可直接通过命令实现 Kafka 消息订阅。

- 精确传输，秒级延迟。
- Doris 可自动感知 Kafka 中 partition 变化，合理调度并发导入。
- 在数据导入这一过程中，支持对 Kafka 原始数据做二次处理（如转换，过滤等）。

八、Doris 其他特性

- 原子性——即一批数据要么都生效，要么都不生效。
- 支持单机多盘
- 向量化执行
- UDF（User Defined Function 用户自定义函数 ）
- 内置 HLL 类型，快速计算 UV

TiDB 的 HTAP 之路：过去，现在和将来

嘉宾：申砾　编辑：Hoh Xil

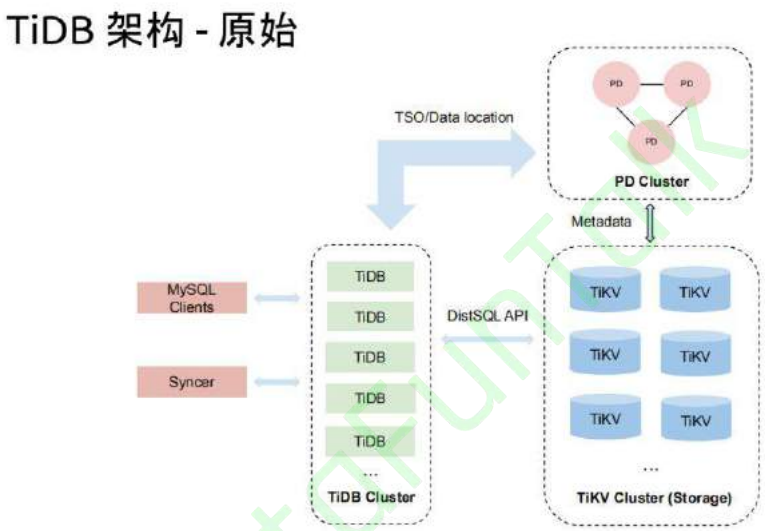
TiDB 有很多故事

每个故事都可以有多个视角，这是一个从 AP 视角讲 HTAP 故事的分享，当然还有技术讨论。

我们开始做 TiDB 时只是想做 TP 数据库，替换 MySQL 分库分表方便用户使用，但是后续用户使用中发现，除了 TP 功能之外，还有很多分析场景也期望数据库来解决。今天主要讲一下 TiDB 在 HTAP 这个领域过去的探索、现在状态和将来的计划。

从 TiDB 的上古时代说起

最开始做的低版本就是想实现水平扩展：数据存储进来将数据分片，加节点能够解决存储空间不够的问题，支持 MySQL 的语法、协议和行为，方便用户将数据存储进去。比如在多个节点中单个节点失效时，保证整个集群还是能够提供服务，做一个实现高可用、分布式的数据库。

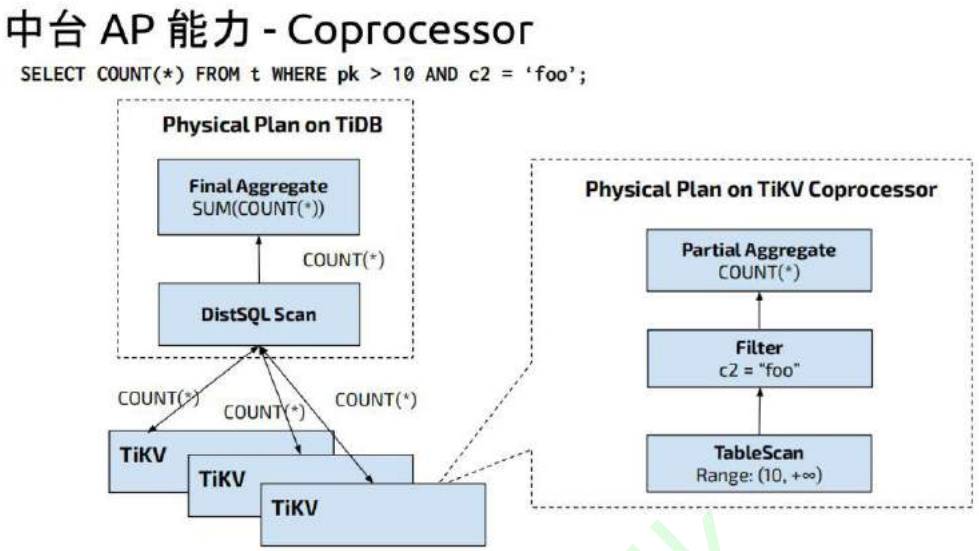


上图是 TiDB 的架构，TiKV 是将数据存储进去，PD 是其调度节点，数据从 DistSQL 写进去。从外面就是 MySQL 服务，调用 MySQL 相关协议进行数据同步下一个 TiDB，这样只要加节点就能实现无限存储，前期测试扩展到几十 T 甚至上百 T 都是可以实现的。早期更多的是类似于一个存储系统，计算能力较弱。相对于 MySQL，MySQL 是小号 T 恤，而 TiDB 是大号 T 恤，其他功能尽可能接近。

用户反馈体验很好，先前需要分库分表，现在不需要分库分表，当业务场景很多时，推动一次业务变更非常困难，拥有一个简单扩展的数据库能解决很多问题。但是在早期对一个数据库而言，尤其是存储交易数据的数据库而言，用户会对其安全性不放心。因此可以存储一些边缘场景，如存日志数据，当其数据丢失可以从 Hadoop 中恢复。但用户反馈数据库挺好，可以做一些数据分析，而前期我们的想法是想做一个承载交易数据的数据库，而三四个用户前期都是做分析的。如游戏客户将游戏投放到广告页面，分析广告投放的效果或者分析流水型的数据分析，当时做了一个并行的 hash join，而 MySQL 没有 hash join，运行还挺快的。

后来从 TP 领域贴近用户场景中，解决有些现有数据库无法解决的问题，如将上百个分片的 MySQL 数据同步，做一些规则变换允许用户进行裁剪和过

滤，随着用户增多，越来越多的数据同步过来，对数据库的承载能力也是一个考验。



在进行数据同步过程中需要进行计算，在 TiDB alpha 阶段计算很简单，包括优化器、执行引擎都很粗糙。我们在此基础下进行优化，1.0 beta 版本的时候做了一个分布式计算框架，让 TiKV 能够承载一部分计算开销，在 TiDB 进行 SQL 优化，选择一部分算子，尽可能多的将算子推到 TiKV 中，将数据从本地导出，在本地做 filter、聚合、limit 等流式算子，然后将数据返回到 TiDB Server 中做一个最终的聚合，同时我们做了很多自己的 join，可以在很多传感器中调用 MySQL，运行效率比较快，能够允许用户在数据量大的时候增加节点，提高计算并发解决问题。

我们分析了用户用我们产品的初衷，在产品早期让用户相信产品是稳定的是很难的事情，除了使用还有数据库运维、日常处理等都需要学习。如游戏运营人员分析游戏日志时分了很多 MySQL 库，写很多 SQL 操作，多个数据库将数据取出来放到 Excel 中，然后通过 Excel 公式进行最后的数据聚合，这样做非常麻烦，现在使用 TiDB 可以直接聚合，非常简单方便。能够实现无障碍跨分片操作，不用额外操作，同时还有实时性，以前是很多 MySQL 数据通过 ETL 拉出来放到 Hadoop 中，再用 Hive 计算，虽然计算能力强，但无论是 ETL 实时性也好，还是维护流水线的复杂度，都很难实时。使用 TiDB 后没有这些操作后可以专注于业务，更加方便，当数据丢失还有备库，这种方式叫中台，可以实现海量存储汇聚。

Everyone Happy Now ?

这样实现后还是出现了一些问题，在简单的 TP 场景下，点查联系、低频率写入对优化器负担较小，对执行引擎也没什么压力。但是用户在 AP 场景中越用越深，数据量越来越多，一些场景中，会遇到问题，比如一些 Query 跑得慢，还有就是会遇到 OOM。因为计算是在 TiKV 上经过聚合、filter 后还需要在 TiKV server 中单点聚合，可能会遇到 CPU 瓶颈，TiKV 内存小，使用也不精细，还有就是没有自动算子选择，还是手动设置，对用户来说很不方便；还有就是只能通过 MySQL 协议和外界交互，没有和其他数据库平台打通，还是需要 ETL 将数据从 TiDB 中通过 MySQL 协议读取写到其他地方或者通过 Spark、JDBC 连接数据库。

不匹配的算力

随着用户越来越多，场景也越来越多，TiDB 的存储扩展能力是非常强的，能存储很多数据，但是早期版本中，计算能力相对存储能力偏弱。这方面不光包括复杂的计算如何去处理，也包括整个计算的负载怎么去分配；存储的调度相对简单，但是计算的调度很难分配，这样导致一些业务出现问题。

不匹配的算力

- TiDB 之间无法直接交换数据
- TiKV 之间也无法在计算过程中交换数据
- 海量存储（TiKV），半单机计算（TiDB）
  - 只能通过 TiDB 服务器 Scale-Up 改善
- Coprocessor 无法处理需要数据交换的算子
  - Join，Full Aggregation，Distinct

PingCAP TiDB

抉择

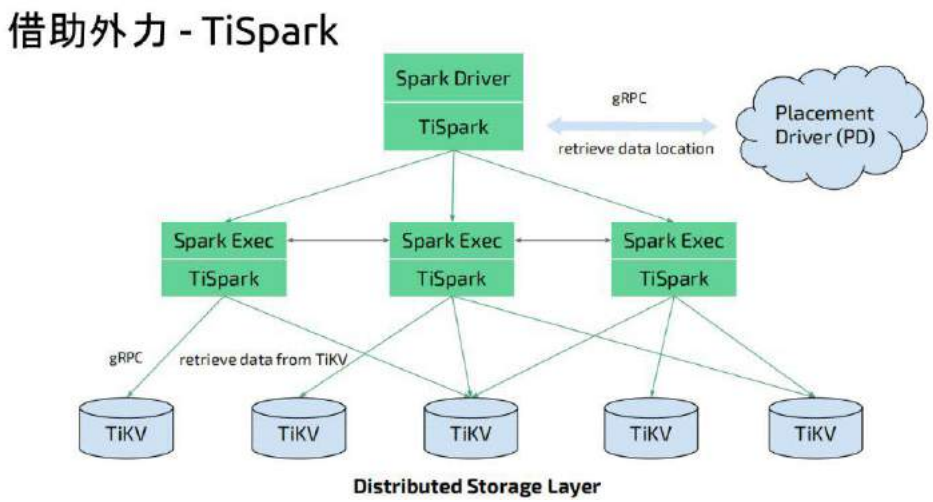
- 要么将 TiDB 或 TiKV 串起来
  - 完全重构优化器和执行器，打造 MPP Engine
  - 风险大，时间长
- 要么，借助外力，拥抱生态
  - 需要一个开源分布式计算框架
  - 成熟度高，用户群广泛

PingCAP TiDB

我们的数据最终都要汇聚到一个 TiDB 上做聚合、join 等操作，每个 TiDB 间无法交换数据，当两个大表做 hash join 时，如果使用内存过多会出现 OOM 的问题，如果通过分布式 join 就能很好解决这个问题。TiKV 间也不

能交换数据，只是相对于多个 Map 加一个 Reduce 的模式；还有就是 join 和 Distinct 无法推到 TiKV 上做，在 TiKV 只能进行流式计算。

TiSpark



那么解决方案可以是自己去做执行优化算子，借鉴 Spark、Hive 等专业分析引擎，运用所有节点集群，做一个真正的 MPP 执行引擎；或者借用一些现有的成熟的分析系统，更好地运用存储进来的数据。我们的选择是借助 TiSpark，在 Spark 中加入 TiDB driver，Spark 在优化器中有部分扩展点，可以将一部分查询优化截取出来，告诉优化器哪些可以做优化哪些算子不能优化，最后返回所需要的数据格式，将 table scan、 inner scan、聚合等简单操作通过 driver 让 TiKV 做初步数据预处理，再将数据返回 Spark 做更重的 join。这样不仅有了一个很好的计算平台，也有了一个良好的计算生态。

TiSpark 具有完整继承 Apache Spark 生态圈，无缝衔接大数据生态圈、成熟的分布式计算平台等优点。

应用 TiSpark 比较重的用户是易果生鲜，先前应用的是 SQL Server、MySQL，想做数据分析，维护的是 ETL 流水线。应用 TiSpark 后基本能做到实时分析，将多个数据源的数据汇总到一个 TiDB 集群中与各种下游分析需求对接，能做实时分析。实现中台数据汇总加实时数据分析。

应用 TiSpark 并不能解决所有问题，运行速度并不是很快，不能利用底层的统计信息做更精细的分析，再者在做有一定并发能力但并不复杂的场景时，分析能力不强，如利用 TiDB 做点查，有少量 key 的点查还是很快的，但是在 Spark 中并发达不到那么高，其 job 启动、调度并不轻量。其次其运维和使用没有 TiDB 简单，因为 TiDB 的接口管理使用与 MySQL 类似。

除了 TiSpark 之外，我们在 TiDB 本身也做了很多工作，优化 TiDB 优化器，早期做大数据分析要写很多 join 算子 hint、并发度多少、聚合算子选择都需要人为选择。一步步优化，从 RBO 到 CBO 到更好的 CBO，不但包括优化器本身、data 模型、plan 搜索，以及如何快速搜集大量统计信息，利用搜集的、扫描的真实数据信息来修正现有的统计信息。在执行模型方面，由一个纯的经典火山模型变成 Batch 加部分向量化的的执行引擎，当优化器预测基础 size 比较大时，在算子之间调用时，不是一次拿一条而是一次拿一

批，batch size 可以实时调节，降低了内存消耗，因为内存分配次数比分配大小更重要些，对整个性能会有很好的优化。在 batch 内部通过 Apache Arrow 格式实现，数据会有一定的压缩，更经典，更重要的是可以对内存使用进行记账，分批次记账比一条条记账更高效；同时对部分算子进行向量化处理，后期将向图形向量化方向发展。TiDB 是一个混合负载机制，目的也是做一个通用负载数据库，不想借助其他特许硬件；同时也支持分区表，对数据进行裁剪。

TiDB 2.0 和 1.0 对比 ,TPC-H benchmark 结果中 query 运行速度明显要快很多，有些查询在 1.0 版本中无法查询现在也可以查询。但是在 2.0 中有些查询比较慢，原因是在 hash 聚合是一个单线程算子。在分析中比较重要的算子就是聚合和 join，join 在 tpch 中占一半，聚合占三分之一。我们的目标就是希望 TiDB 在不借助 Spark 情况下也能和主流分析工具媲美。在 TiDB 2.1 版本中，我们做了进一步的优化，特别是多线程的聚合算子，对 TPC-H 结果带来了进一步的提升。

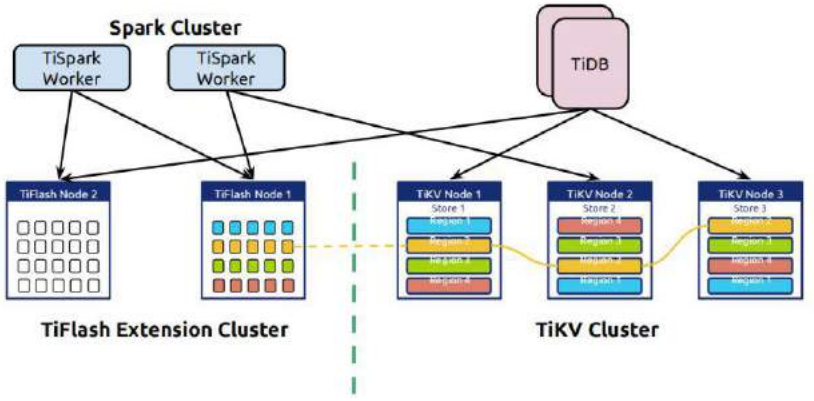
核心矛盾

TiDB 2.1 版本在很多分析场景中已经跑的不错，但是在面对需要对大表进行全表扫描时，行存模式明显不满足需求，无论是 scan 或者简单 filter 或者 IO 压力都明显不如内存。还有在 HTAP 场景中需要解决 AP 负载不影响 TP 负载。TiDB 内部提供的解决方案就是让分析的负载优先级更低，可以运行低优先 SQL，还有就是依据优先级做作业调度等，还有就 TiDB 实例可以无限拓展，可以一部分给 TP 用，一部分给 AP 用，但是最终 TiKV 对磁盘、网络的占用还是无法避免；还有就是大查询执行下，有几个在线小查询，也没有 CPU 抢占机制；这些问题都是无法避免的。

TiFlash

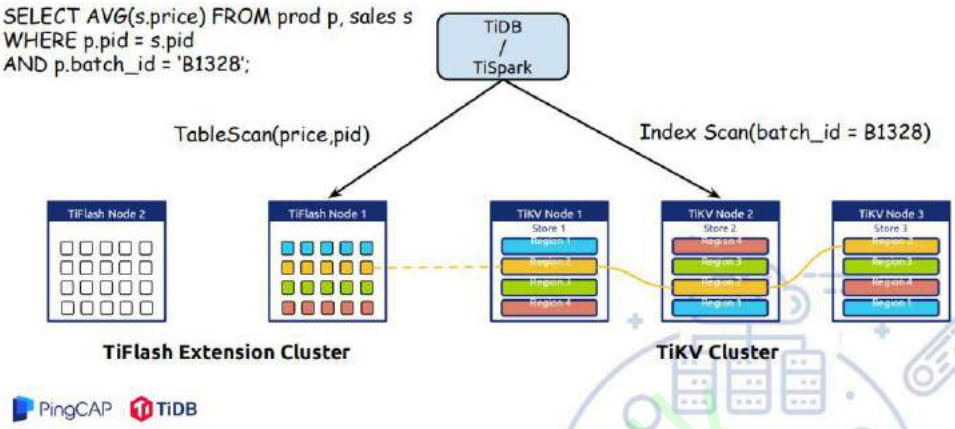
TP 和 AP 之间对于存储格式的偏好不同，TP 一定要行存，需要低延迟，而 AP 一定要列存。一些数据库尝试过一个集群中同时有行存和列存，比如 Oracle 有列存插件。我们也开发了一个新项目——TiFlash，简单来说是一个 TiKV 的列存扩展。可以通过 Raft 日志将 TiKV 使用的数据实时同步出来，通过 Raft log 将数据同步到 TiFlash 实例，它会将行存的数据转到列存，然后存到本地列存引擎中。开发借用了部分 ClickHouse 代码，因为 ClickHouse 是一个以速度见长的引擎 ,在向量化方面做的比较细。通过 Raft 日志可以同步 TiKV 上所有数据，包括 MVCC、事务状态，能够实现数据的事务一致性读取 ,唯一影响的是 TiKV 要同步一分日志过来 ,Raft learner 只会一次写入不会参与投票，对线上写入延迟影响不大。

TiFlash 架构



TiFlash 当前版本架构：数据从 TiDB 集群中写入进来后，通过 Raft 日志同步到 TiFlash 节点 ,通过 Spark 进行数据读取。TiFlash 以 Raft Learner 方式接入 Raft Group，使用异步方式传输数据，对 TiKV 产生非常小的负担。当数据同步到 TiFlash 时，会被从行格式拆解为列格式。在读取时会有一个校验机制，在读数据时会进行一个 RaftCommand 判断是否有读取数据，当数据同步到读取请求发送时间点的进度时，TiFlash 才提供读取。虽然有一次 Command，但是如果分析比较重，整个负载还是比较小的。

融合

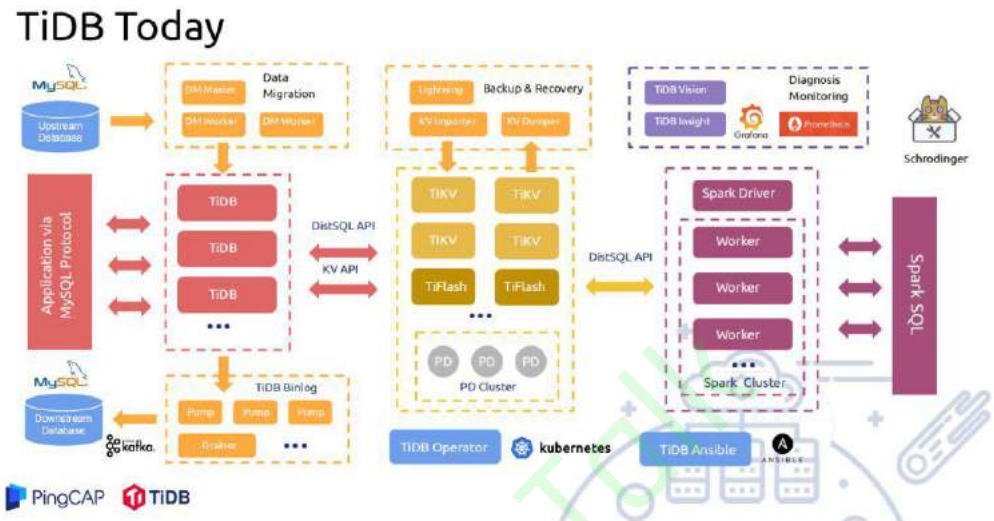


TiFlash 的下一个版本会做计算的融合，将入口切换转为一个，实现 TiDB 可以同时读取行存和列存副本。有些 SQL 部分选择列存部分选择行存，交给优化器来选择，原理是将列存当做一个特殊的索引，让列存读到与行存一样的数据。目前 TiFlash 已经进入 beta 版，在一些用户那边已经开始进行 PoC，预计在年内发布第一个正式版本。

至此为止

- TiDB = ~~X% TP + Y% AP~~ = HTAP
  - TiDB 并不需要你选择 TP 还是 AP，它就是 HTAP
- 一套平台，兼容行存列存
  - 无痛数据同步
- 当主 TiDB 集群承担 TP 服务时，方便地在列存上进行分析
- 或者，将列存当做索引，和行存共同提供混合服务

目前的 TiDB



将存储引擎的数据通过 Raft learner 同步出去可以做更多的事情，保证日志是一个近实时同步方式。计算引擎也可以替换成不同的东西，寄存到自己的引擎中做自己的事情。希望用这套场景解决 AP 与 TP 相互干扰的状况，帮

助用户简化使用。在 TP 中统计信息收集比较麻烦，当数据有上百亿表，这种收集统计信息是非常耗时的，当统计信息不准可能会导致业务选错索引。对于这种不需要完全实时，也不要求百分百准确的统计信息收集工作，可以让其后台执行，交给 TiFlash 引擎来做。

存储集群不再区分是何种引擎，是一个行列共存的存储引擎。可以实现数据实时同步，可以实现快速将几 T 或者几十 T 的数据快速导入集群中。有了这么多组件后维护起来比较麻烦，可以使用 TiDB Operator，让 K8s 帮助管理实例。

还有哪些问题？

目前的架构已经足够强大，但是还有些问题没解决。Spark+TiFlash 架构中，底层引擎还不支持 MR 模型，Spark 的 MR 模型还是略重，跑起来速度一

DataFunTalk

般。后期希望自己写一套 MPP 引擎，让 TiFlash 节点之间可以交换数据，减少 Spark 端的工作任务，加快查询速度；还有 SQL 语句推下去做计算，统一协处理器层，让 TiKV 和 TiFlash 都能组成 MPP 集群。同一套代码，同一套引擎，让 TiFlash 也支持 SQL 行为，这样才能接上 MySQL 引擎。

还有写入还是要借助 TiKV，通过 TiDB 和 TiKV 写入行存，Raft 日志同步到列存中，在写入过程中吞吐不够，将数据从行存转入列存流水线较长，开销也会加大。

另一个就是需要加强写入，TiDB 对一次性写入的数据大小是有限制的，我们希望提供一套快速且支持大数据量的写入，支持几个 G 的数据原子写入。TiFlash+TiDB 能解决海量数据计算问题，但是计算后不能将数据原子写回，会导致整条业务逻辑不通。有了大数据量回写之后，TiFlash 可以承担 ETL 的功能。

最后就是可以用不同引擎解决不同问题，就像将行存列存引擎接到 TiKV 中来解决不同的问题，我们还可以添加更多的引擎。此外通过更好的调度，将不同的引擎调度到不同的机器实例上，实现完美的负载隔离，并且分散不同类型的业务负载到不同类型的引擎。

DataFunTalk

快手 HBase 在千亿级用户特征数据分析中的应用与实践

嘉宾：陈杨　编辑：Hoh Xil

快手建设 HBase 差不多有 2 年时间，在公司里面有比较丰富的应用场景：如短视频的存储、IM、直播里评论 feed 流等场景。本次只分享其中的一个应用场景：快手 HBase 在千亿级用户特征数据分析中的应用与实践。为什么分享这个 Topic？主要原因：对于大部分公司来说，这都是一个普适的场景，因为很普遍，所以可选择的分析引擎也非常多，但是目前直接用 HBase 这种分析用户特征的比较少，希望通过今天的分享，大家在将来遇到这种场景时，可以给大家提供一个新的解决方案。

本次分享内容包括：

业务需求及挑战：BitBase 引擎的初衷是什么；

BitBase 解决方案：在 HBase 基础上，BitBase 的架构是什么样；

业务效果：在快手的实际应用场景中，效果如何；

未来规划：中短期的规划。

业务需求及挑战

1. 业务需求

用一句话来概括业务需求：在千亿级日志中，选择任意维度，秒级计算 7-90 日留存。

如上图所示。左边是原始数据，可能跨 90 天，每一天的数据可以看作是一张 Hive 宽表，在逻辑上可以认为每行数据的 rowkey 是 userId（这里不严谨，userId 可能是重复的），需要通过 90 天的原始数据计算得到右边的表，它的横轴和纵轴都是日期，每个格子表示纵轴日期相对于横轴日期的留存率。

该需求的挑战：

日志量大，千亿级；

任意维度，如 city、sex、喜好等，需要选择任意多个维度，在这些维度下计算留存率；

秒级计算，产品面向分析师，等待时间不能过长，最好在 1-2 秒。

2. 技术选型

面对这些问题，我们当时的技术选型：

① Hive，因为大部分数据可能是存在 Hive 里，可以直接写 SQL 计算，该方案不用做数据迁移和转换，但是时延可能是分钟到小时级别，因此否定了这个方案。

② ES，通过原始数据做倒排索引，然后做一个类似计算 UV 的方式求解，但是在数据需要做精确去重的场景下，它的耗时比较大，需要秒到分钟级。

③ ClickHouse，ClickHouse 是一个比较合适的引擎，也是一个非常优秀的引擎，在业界被广泛应用于 APP 分析，比如漏斗，留存。但是在我们的测试的中，当机器数量比较少时（ <10 台 ），耗时依然在 10 秒以上。

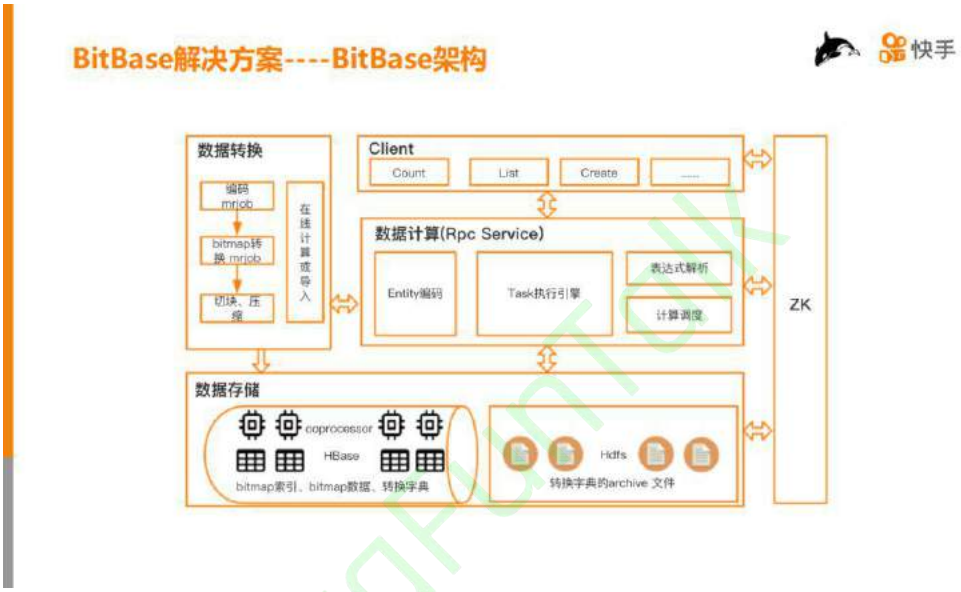
立足于这种场景，是否存在其它解决方案，延迟可以做到 2-3 秒（复杂的场景 10 秒以下），同时支持任意维度组合？基于 HBase，结合业界简单/通用的技术， 我们设计并实现了 BitBase 解决方案，用很少的资源满足业务需求。

BitBase 解决方案

1. 数据模型

如上图所示，首先将原始数据的一列的某个值抽象成 bitmap（比特数组），举例：city=bj，city 是维度，bj (北京) 是维度值，抽象成 bitmap 值就是 10100，表示第 0 个用户在 bj，第 1 个用户不在北京，依次类推。然后将多维度之间的组合转换为 bitmap 计算：bitmap 之间做与、或、非、异或，举例：比如在北京的用户，且兴趣是篮球，这样的用户有多少个，就转换为图中所示的两个 bitmap 做与运算，得到橙色的 bitmap，最后，再对 bitmap 做 count 运算。count 表示统计“1”的个数，list 是列举“1”所在的数组 index，业务上表示 userId。

2. BitBase 架构



整个 BitBase 架构包括五部分：

数据存储：主要存储两类数据，一类数据是 bitmap 索引和数据，另一类是转换字典的归档文件（见后面描述）。

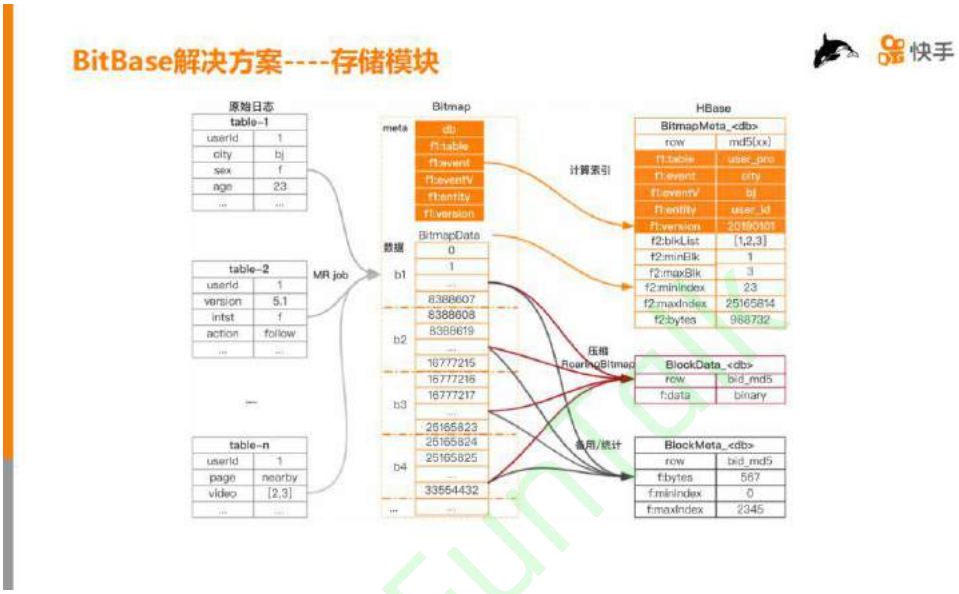
数据转换：有两种方式，第一种是通过 mrjob 转换，第二种是在线计算或导入；

数据计算：负责计算和调度，并把 IO 数据计算结果返回给 Client；

Client：站在业务的角度，把它们的业务逻辑分装成一个个业务的接口；

ZK：整个系统是一个分布式的服务，用 ZK 做管理。

3. 存储模块



用数据存储设计的核心目的是让计算更快。

如上图，左边为一天的原始数据，包括多个 table，通过 mrjob 或者 rpc 的方式转换成中间的 bitmap。

bitmap 分为两部分，第一部分为 meta 信息（橙色部分），第二部分是 data 信息：

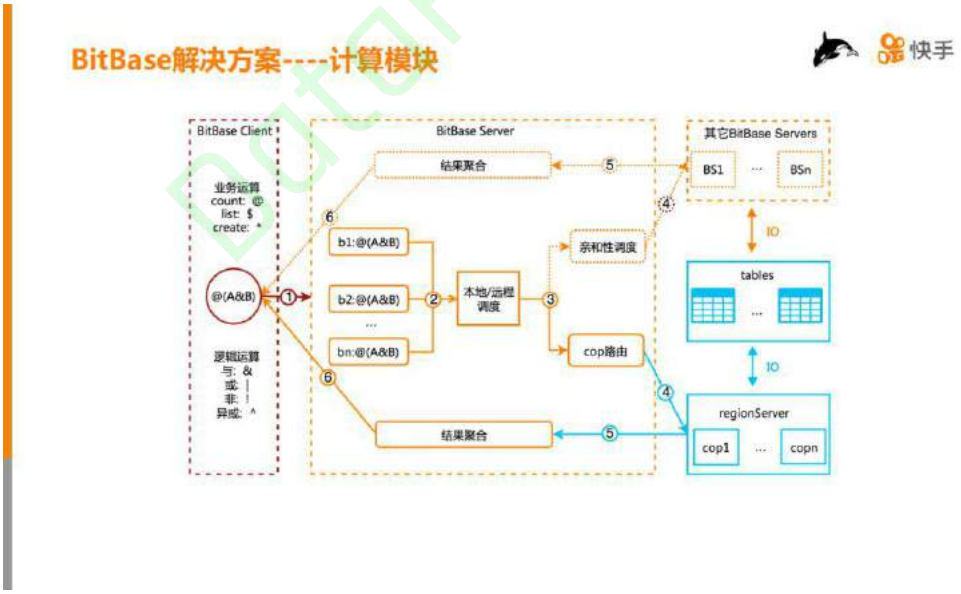
- Meta 信息唯一定位一个 bitmap，db 可以认为是 hive 中的 db，table 也可以认为是 hive 中的 table，event 表示维度（如:城市），eventv 表示维度值（如:bj），entity 表示 userId（也可能是 photoId），version 表示版本。

- BitmapData 从物理上讲是一个比特数组，把比特数组按照一定的大小进行切块：b1，b2，b3，...，bn，从而实现分块存储，分块计算。

最后把 bitmap 存在 HBase 的 3 张表中: 两张核心表和一张辅助表。

- BitmapMeta，保存 bitmap 的 meta 信息和一些 block 索引信息。
- BlockData， 直接保存 bitmap block 数据。
- BlockMeta，保存 block 的 meta 信息，起辅助作用。

4. 计算模块



一个完整的计算流程涉及到三个组件：BitBase Client、BitBase Server 和 HBase regionServer。

① BitBase Client 首先把业务的需求封装成计算表达式，然后把计算表达式发给 BitBase Server；

② BitBaseServe 接收到请求后，从 BitmapMeta 表中查询 Block 索引，然后根据索引将表达式切分为 n 个子表达式；

③ 如果所有 bitmap 的 db 相同，则走 coprocessor 路由，否则按照数据亲和性，将 block 计算分发到其它 bitbaseServer 中。

④ 根据第 3 步的调度策略，分两条不同的路径计算 block 表达式

⑤ BitBase Server 聚合 block 计算表达式的结果，然后返回给 BitBase Client。

两种计算方式的对比：

非本地计算，解决跨 db 计算的需求，它主要的瓶颈在于网卡和 GC。

本地计算，解决同 db 计算的需求，它主要的瓶颈在 CPU 和 GC 上。整体上看本地计算的性能比非本地计算的性能提高 3-5 倍，所以要尽量采用本地计算方式。

5. DeviceId 问题

在引入 Bitmap 数据模型之后，我们隐含的也引入了一个非常大的问题：无法支持 deviceId。要支持 deviceId，首先需要将 deviceId 转化为数字类型，并且转换之后的 DeviceIdIndex 必须要满足四个条件：

① 连续：deviceIdIndex 如果存在空洞，会降低压缩效率，同时 Block 数量会增加，计算复杂度相应增加，最终计算变慢；

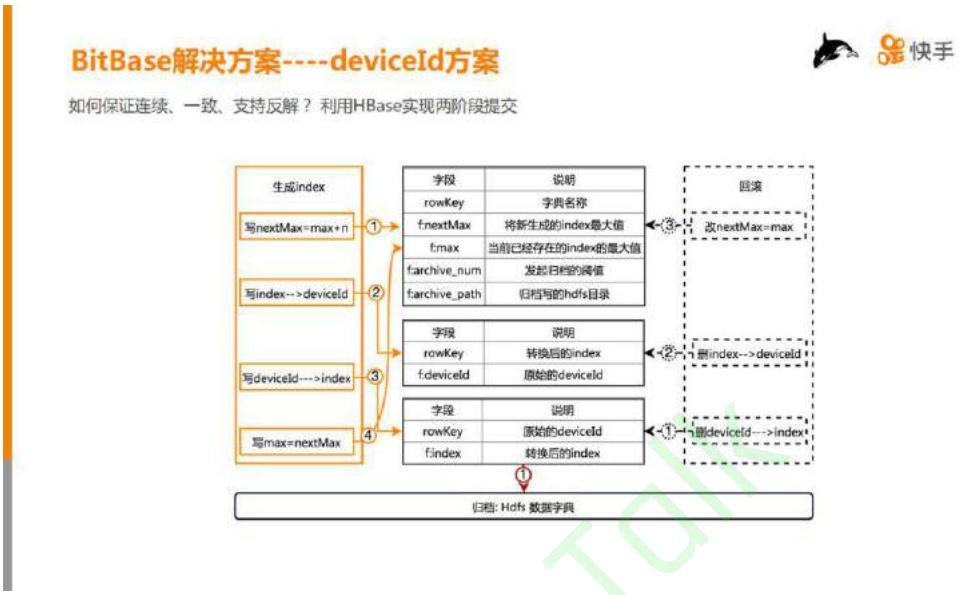
② 一致：deviceId 和 deviceIdIndex 必须是一一对应的，否则计算结果不准确；

③ 反解：根据 deviceIdIndex 能够准确、快速地反解成原始的 deviceId；

④ 转换快：在亿级数据规模下，deviceId 转化为 deviceIdIndex 的过程不能太长。

6. DeviceId 方案

连续、一致、支持反解：



如何保证连续、一致、支持反解？解决方案非常简单，利用 HBase 实现两阶段提交协议。如上图中间实线部分所示，定义 deviceId 到 deviceIdIndex 的映射为字典。第一张表存储字典的 meta 信息；第二张表存储 index 到 deviceId 的映射；第三张表存储 deviceId 到 index 的映射。

生成 Index 的过程。举例说明，假设我们已经生成了 1w 个 deviceId 映射，那么此时 f:max=1w，现在将新生成 1k 条映射：

① 将 f:nextMax=f:max+1k=1.1w；

② 写 Index 到 deviceId 的反向映射表，1k 条；

③ 写 deviceId 到 Index 的正向映射表，1k 条；

④ 把 f:max=f:nextMax=1.1w 更新到 meta 表，生成过程结束。

如果在生成过程中出现异常或服务器宕机，则执行回滚流程：

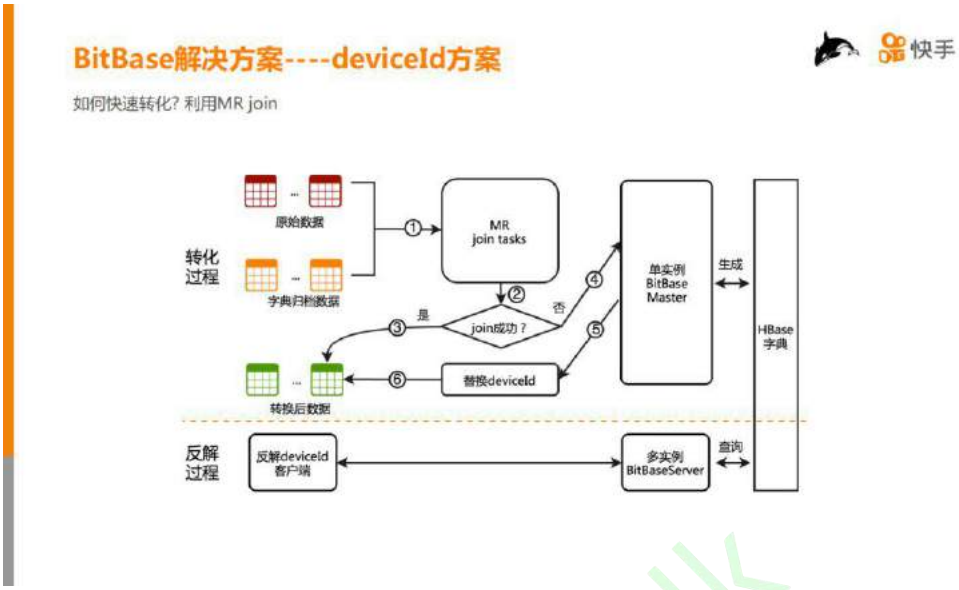
① 如果我们检测到 f:nextMax 不等于 f:max(f:nextMax>f:max) ,则从表 2 中查询 max 到 nextMax 的数据，从表 3 中删掉相应的 deviceId 到 index 的映射记录；

② 再删掉表 2 中相应的 index 到 deviceId 的记录；

③ 最后把 f:nextMax=f:max，从而实现数据 100%一致。

用 HBase 实现两阶段提交协议要求 index 生成流程和回滚流程一定是单线程的，从而出现性能瓶颈，所以 BitBase 设计了归档流程，以支持快速转换(见后面的描述)。Meta 表中有两个字段，如果发现新产生的数据大于 f:archive_num 就发起归档，把表 3 中的新数据直接写到 HDFS 中 archive_path 目录下。

快速转化：



这里我们用到了 MRjob 中的 Join：

① 同时输入原始数据和字典归档数据，在 MRjob 中根据 deviceId 做 join；

② 判断 deviceId 是否 join 成功；

③ 如果成功了，直接写 hdfs，这样就得到了转化后的数据；

④ 如果 join 失败，直接请求单实例 BitBase Master，BitBase Master 通过两阶段提交协议生成新的映射；

⑤ 然后返回给 join task 执行替换 deviceId；

⑥ 把转换后的数据写入 hdfs。

反解的过程很简单，直接多并发读取 HBase。

业务效果

如上图所示，第一个图是，2 维度，不同时间跨度计算留存的时间延迟，第 2 个图是 15 日留存在不同维度上的时延，时延并不会随着维度的增长而增长，原因是维度越多，表达式中可能不需要计算的 block 块也越多。

如上图所示，BitBase 可以应用在 app 分析，用户增长，广告 DMP，用户画像等多个业务场景中。

未来规划

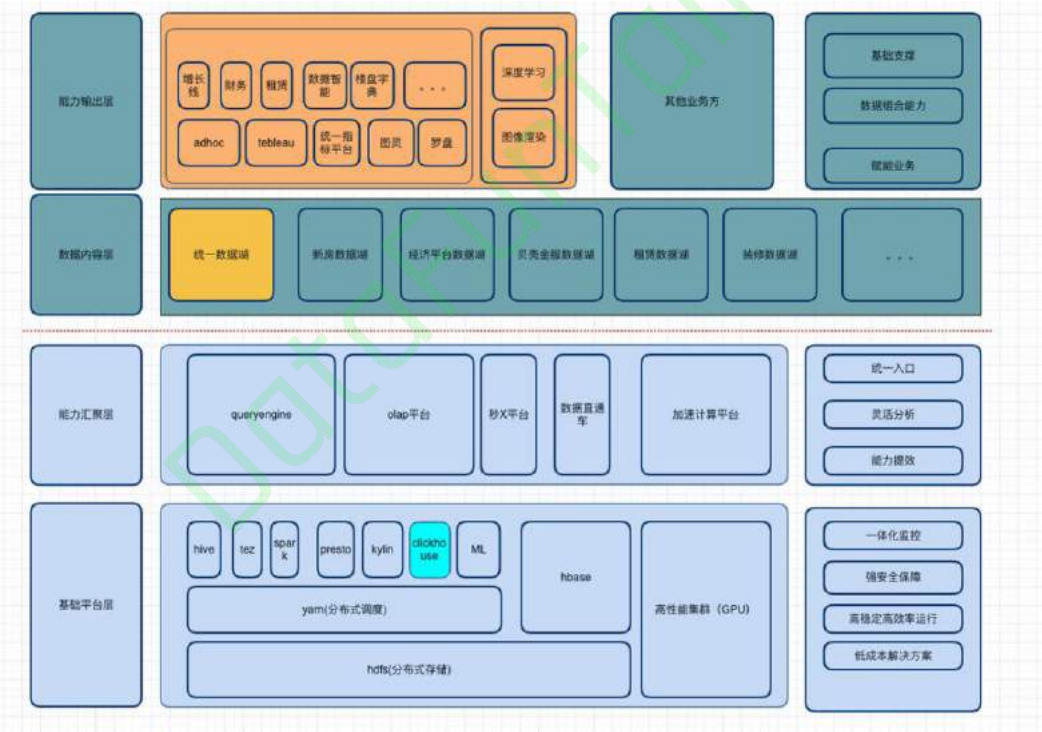
根据现在面临的业务场景，BitBase 后续会在多个方面做优化。支持实时聚合，在一些业务场景下，如运营效果监测，导入时效需要 <5min，BitBase 需要支持实时聚合；支持 SQL 查询，目前只支持 api 的接入方式，在一些简单场景下比较复杂；开源，希望通过开源，和大家一起挖掘 BitBase 的业务场景。

贝壳：流式数据的平台化实践与挑战

嘉宾：赵国贤　整理：Hoh Xil

今天为大家分享贝壳找房流式数据的平台化实践与挑战，具体介绍下如何建设流式数据平台来满足业务方的需求。

总体架构



贝壳找房大数据的整体架构，从下到上分为四层：

1. 基础平台层。这一层应用的都是比较常见的技术 :HDFS 分布式存储 ,yarn 分布式调度，以及 HBase 存储，另外还有一些计算引擎，如 hive、tez、

spark、presto、kylin、clickhouse、SparkML 等来满足各种各样的基础需求，同时还有高性能的计算机集群。

基础平台层的工作总结起来包括：

- 一体化监控
- 提供强安全保障
- 高稳定高效率运行
- 低成本解决方案

2. 能力汇聚层。大家知道每年都有大量的开源组件，那么如何让上层的业务方更好的应用这些组件，就是能力汇聚层要完成的工作：

Queryengine ,可以通过 queryengine 使用 hive、tez、spark、presto 来查询整个底层存储的数据。

Olap 平台 ,通过封装 kylin(预构建)、HBase、Phoenix、presto、clickhouse（流量分析）等引擎能力，打造统一的 oalp 平台来满足业务方不同的需求。

流式计算平台，这个是今天要分享的主题，包括流式计算产品天眼（原秒 X 平台）；然后是数据直通车，它是贝壳平台打造的一个数据接入平台，可以进行实时接入和离线接入。

加速计算平台，底层有高性能集群，上层就需要有加速计算平台，来满足业务方机器学习或者深度学习的需求。

能力汇聚层的工作总结起来包括：

- 统一入口
- 灵活分析
- 能力提效

更上层就是数据仓库做的事情：

3. 数据内容层。这层主要是做统一数据湖，新房数据湖等等一些数据治理的工作来构建数据仓库，以满足更上层业务方的需求。

4. 能力输出层。

DataFunTalk

Adhoc 即席查询，比如贝壳找房这边有些城市站，有运营市场，adhoc 可以直接写一些 sql，调用 queryengine，最终通过各种各样引擎的能力，来满足即席查询的需求。

我们还引进了 Tebleau 软件，可以用来做 BI 分析，它的底层也是调用 queryengine 查询数据内容层的数据。

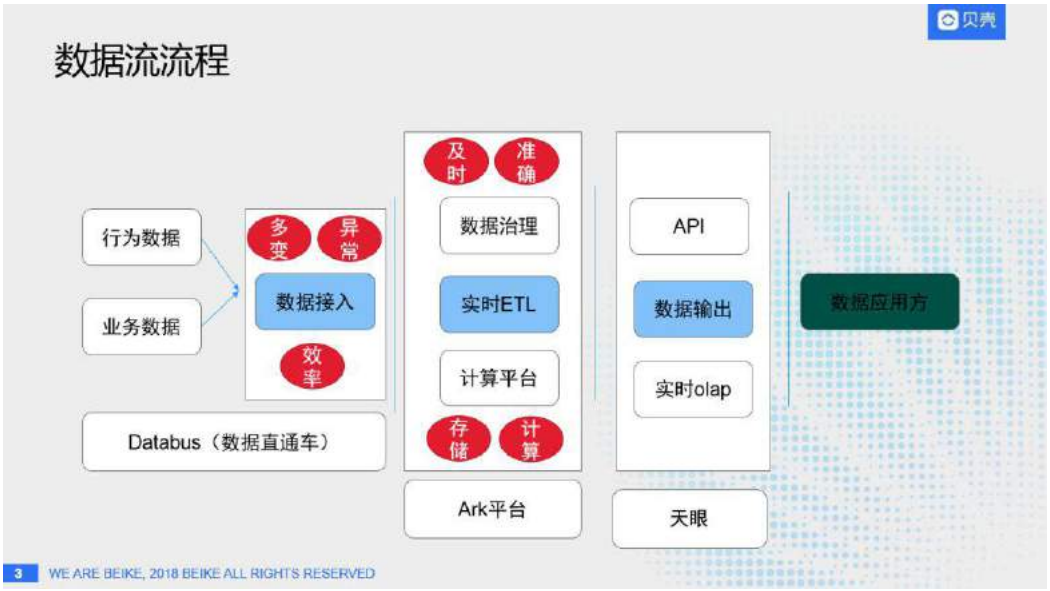
同时我们也构建了统一指标平台；图灵，满足经纪人的需求；罗盘，满足一些流量分析的需求。

能力输出层的工作总结起来包括：

- 基础支撑
- 数据组合能力
- 赋能业务

数据流流程

DataFunTalk



要理解流式数据，首先要知道数据流的流程：

1. 数据接入。为了做好数据分析，同时给用户画像、数据挖掘等提供数据物料，就需要做好数据接入层，一般情况下我们面临的问题主要有：

① 多变：数据来源有业务 DB、MySQL、Oracle、sql server、第三方存储 Redis 等多种数据源，怎么对多变的数据源做统一的、及时的接入。

② 异常：举个例子，某些业务方经常在半夜刷新数据，这对数据接入的实时性和准确性产生了很大的影响。

③ 效率：

A. 离线数据接入，如何更快的把数据接入进来。

B. 实时数据接入，如何准确的接入，并及时提供需求数据。

针对这些问题，贝壳找房的解决方案就是 **Databus**（数据直通车），通过数据直通车来解决上述的三个问题，把行为数据和业务数据及时、高效的接入计算平台层，来满足流式数据的计算和需求。

2. 实时 ETL。主要分为上层的数据治理和下层的计算平台。数据接入进来之后，采用什么样的形式存储，更好的做数据分析，而实时计算又采用什么样的方式存储，这里都是通过 Ark 平台来做实时处理。

3. 数据输出。如何把数据更好的提供给用户，这里主要会介绍日志流产品化的平台天眼。

挑战

流式数据平台面临的挑战：

1. 流式元数据管理。公司到底有哪些流式数据？比如 kafka 接了一批日志，MySQL 产生了一批 binlog，各种事务中也产生了一些流式数据。这些流式数据来自哪些业务方？数据类型是什么？有什么样的属性？以贝壳为例，是新房的数据？还是二手房的数据？还是经纪人的数据？还是客户的数据？里面数据的格式是什么样的？如何存储的？如何解析？都需要统一的流式元数据管理平台，来更好的理解并处理这些数据。

2. 流式数据处理平台。常见的 Flink、spark streaming 等，对于需求方来说，不可能总写相应的代码，来完成操作，这样成本比较高。如何让业务方尽可能简单的、可配置的进行流式处理操作，这里主要是通过 Ark 流处理平台完成，后面会重点介绍。

3. 流式数据应用产品。贝壳拥有多个场景，如日志分析、数据挖掘等，每个场景对流式数据都有不同的需求，这就需要我们针对不同需求做不同的产品。

流式计算平台及应用

1. 数据直通车

数据直通车的架构：

数据接入层分为：

① 流数据接入：贝壳的流数据包括，ngx 日志、dig 埋点、mysql-binlog、tidb-binlog、其他业务日志等，把这些流数据通过 rsyslog、tidb-binlog-drainer、canal 等接入 kafka。

② 存储接入：离线数据可能存在 Hive、Mysql、redis、HBase、TiDb 等多种数据存储源，通过 spark 离线任务、TiSpark 把数据接入到更上层的数据输出层。

流式数据接入之后，就可以做订阅，如何 kafka、redis、ES（检索操作）、HBase（大数据量 KV 操作）等，还可以做周期性同步，T+1 级别的数据，还有我们探索实现的准实时仓库，目前可以做到 5 分钟的延迟。

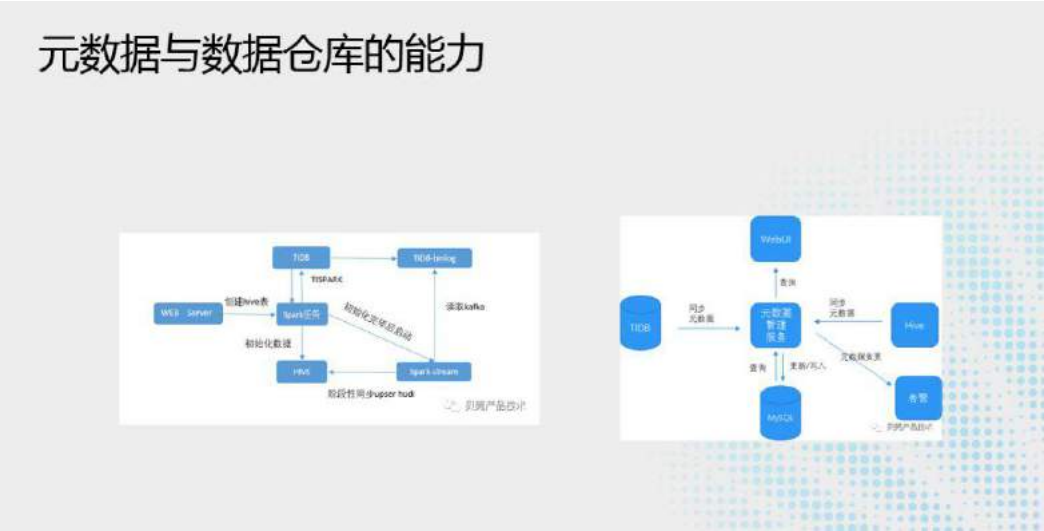
依赖于整个数据接入层、处理层、输出层的工作，可以满足：

① 建设完元数据之后，通过应用层就可以查询到所有元数据的信息，并且知道元数据的变更情况（如果不知道元数据的变更情况，那么数据可是不准确的，和真实的数据是对应不上的）。通过数据直通车，就可以满足我们的第一个挑战，完成了所有元数据的管理。可以及时、准确、高效的把数据接入到存储层，或者为数据处理层准备好数据物料。

② 流数据订阅

③ binlog 变更查询等。

元数据与数据仓库的能力：



简单说下元数据管理服务，见右图，主要通过 TiDB-binlog，获取业务数据之后，把数据直接同步到 mysql，变更之后如果能自动处理就自动处理，如果不能自动处理就会发出告警，由人为来操作，这样来保证业务数据是统一的，同时通过 WebUI 接口给用户来查看，也可以通过 API 接口给其他业务来调用。比如后面要讲的 Ark 平台就可以直接调用 API 来使用流式数据。

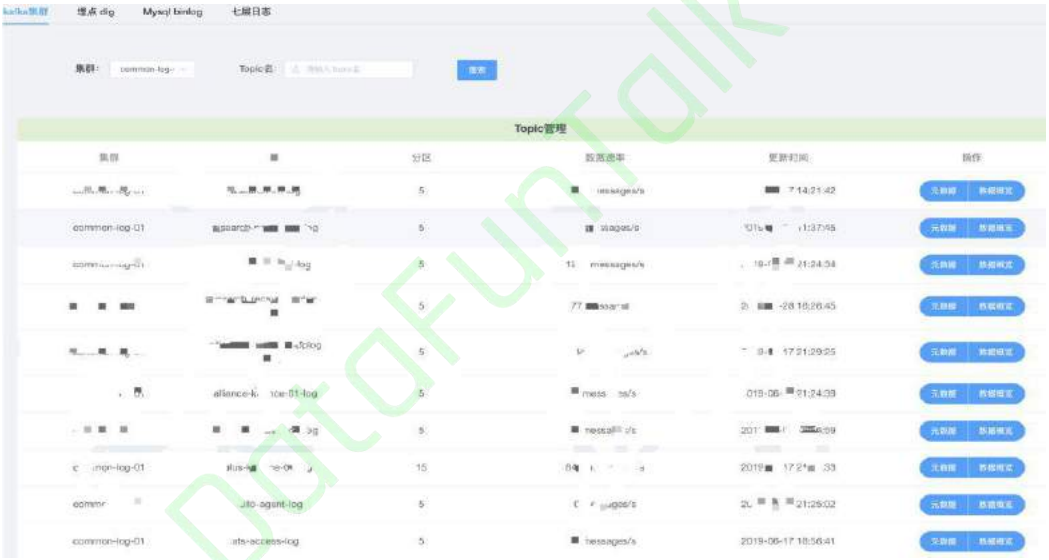
准实时数仓，见左图，引入 hudi 能力，通过 hudi 和 TiDB 可以准实时的把数据导入数据仓库之中，用户可以在 5 分钟延迟的情况下来满足对于业务数据的统计、监控、分析的需求。

产品化展示：

这是平台的首页，第一个图可以提供业务的执行情况，第二个图告诉我们数据的变更情况，第三个图是整个业务成功的情况。

左边栏：数据字典，元数据能力；实时查询，准实时数仓查询的能力；以及埋点管理，数据订阅管理、数据变化查询等。

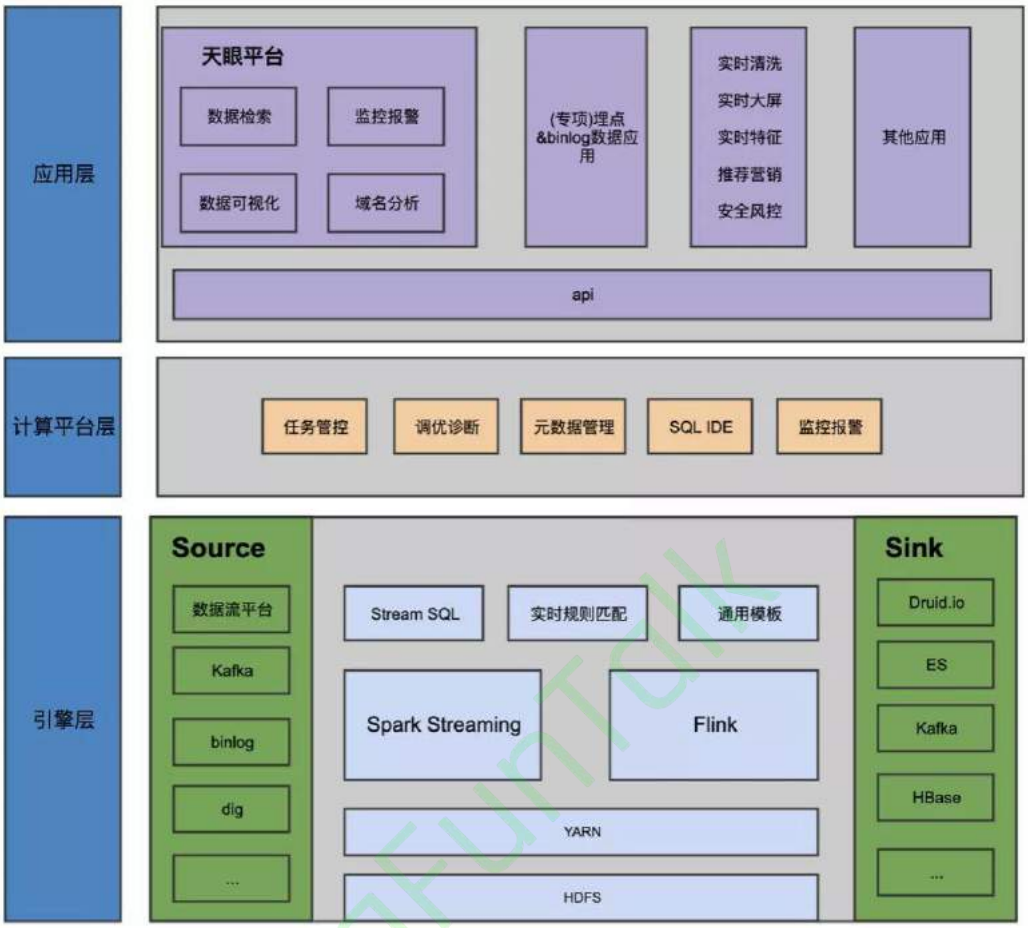
通过数据直通车，解决了数据接入和元数据的管理问题。



DataFunTalk

目前可以做到 kafka 集群、埋点 dig、Mysql binlog、七层日志的元数据管理。

2. Ark 流处理平台



Ark 整体架构：

引擎层：

① 把数据接入到 Source 源，如 kafka、binlog、dig 等。

② 数据接入之后，在引擎层进行处理，主要方式有三种：Stream SQL、实时规则匹配、通用模板，其底层依赖于 Spark Streaming 和 Flink。

③ 最终可以把数据输出到 Sink 中，如：Druid.io、ES、kafka、HBase 等，这时用户可以直接调用。

计算平台层：

通过封装的方式来构建计算平台，计算平台主要通过这样几件事儿来完成稳定性和通用性的需求，第一个是任务管控，需要管理好我们的任务，如何调度我们的任务，第二个是调优诊断，有些用户写的任务可能不是特别完善，我们可以自动判断他的任务是否需要优化，并给出优化建议，第三个可以引入 Databus 中的数据进行处理和使用，第四个是 SQL IDE，我们会尽可能多的生成 SQL，让用户尽可能写一些少的配置，来应用 Ark 平台，最后就是监控报警，监控整个任务的运行情况和延迟情况。

应用层：

通过 API 把我们 Ark 平台的能力输出出去：

① 数据清洗

② 实时大屏

③ 实时特征

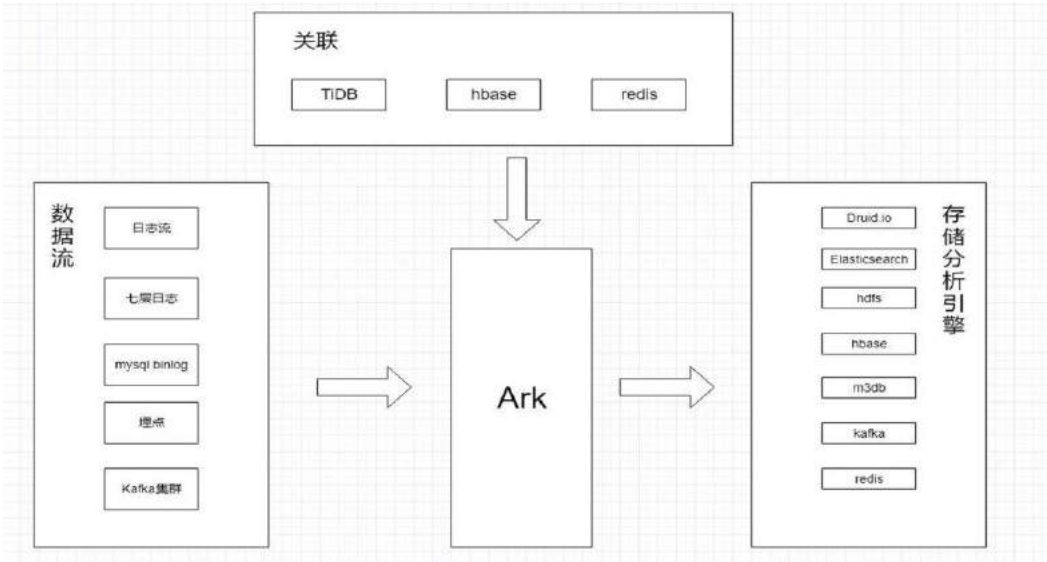
④ 推荐营销

⑤ 安全风控

还有其他的一些应用，以及后面会讲到的天眼平台，通过对日志的流式处理统一化及产品化满足日志的各种诉求。

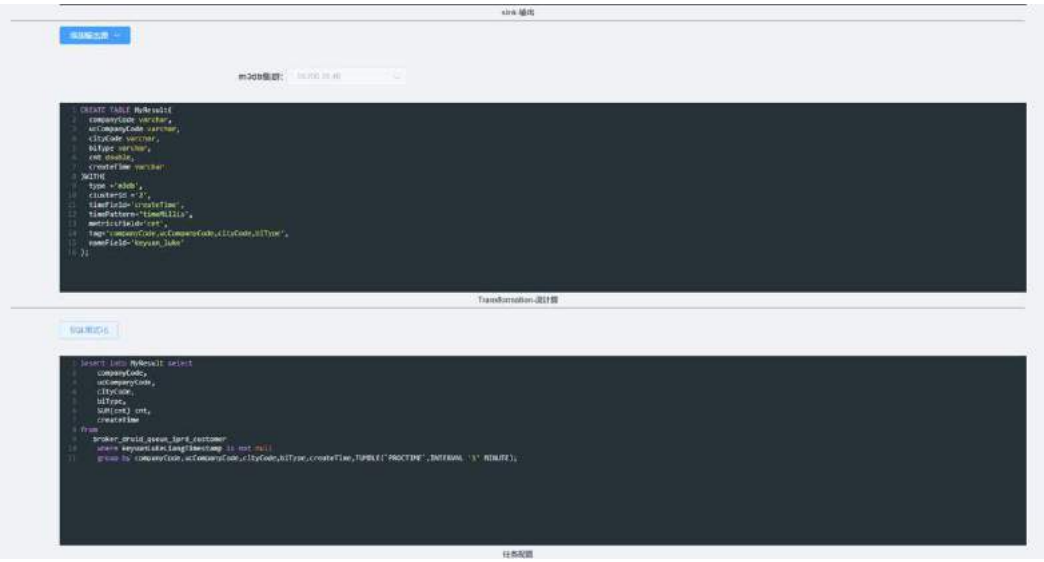
这就是整个 Ark 流处理平台提供的能力。

Ark 能力接口：



数据流（包括日志流、七层日志、mysql、埋点、kafka 集群等）通过 Ark 平台可以把数据清洗到多个分析引擎中，并且可以通过 TiDB、HBase、redis 做关联，根据不同的需求把数据关联到不同的存储分析引擎中。

产品化展示：



这个是 SQL IDE 的能力，90%的 SQL 都是自动生成的，大家只需要配置 m3db 地址，处理的类型等，就可以直接应用 Ark 流处理平台。这样就降低了很大的代价，因为 sql 的接入度是最强的。

这个是 Ark 流处理平台的首页，我们简单介绍一下产品的能力特点：

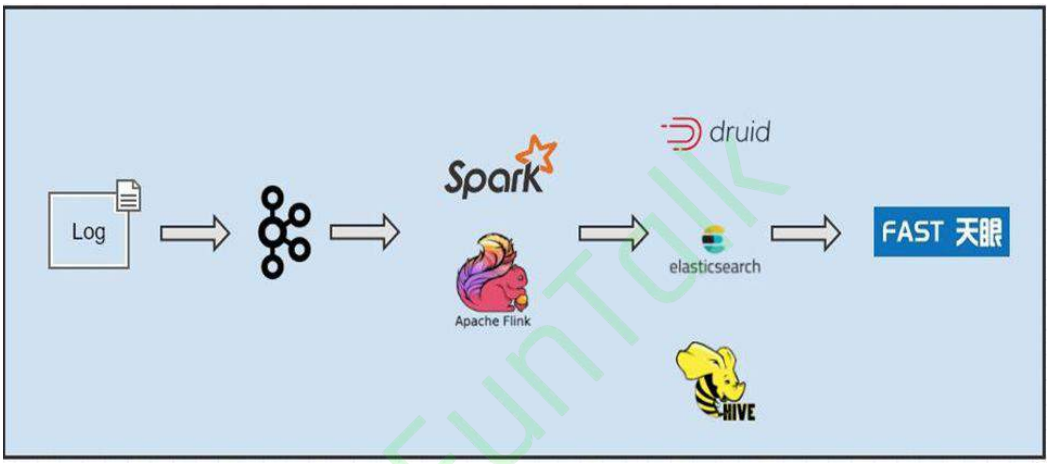
丰富的应用表达方式

计算结果实时可视化

接入全量流数据源

资源管理 & 调优诊断

3. 天眼



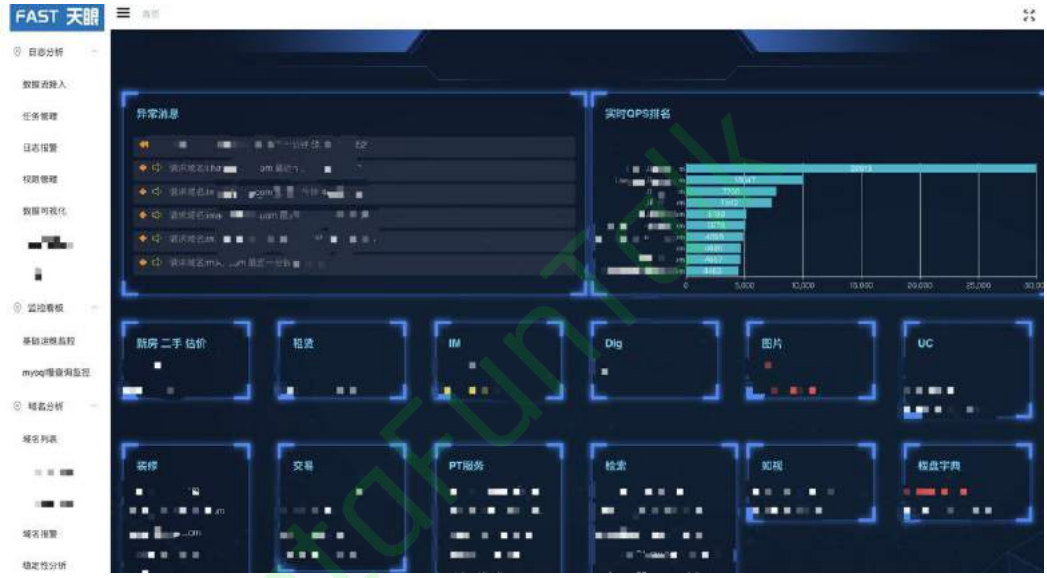
传统的方式，日志接入到 kafka，通过 spark streaming 或者 Apache flink，解析到 druid/es/hive 中来做分析，每次业务方都是按需来做这些事情，自己做数据清洗，自己在做应用，非常繁琐耗时，为了解决这个问题，我们把这些步骤集成到了一个产品天眼中。

天眼架构：

数据采集：所有的业务日志、访问日志、设备日志以及第三方日志，Metrics 上报，SDK 埋点数据，Agent 数据，都可以通过 Databus 接入到数据处理

层，然后再按需放入 Druid.io、HDFS、ES、Hive、Mysql 中，再通过查询引擎，可以做到检索、统计报表、可视化、数据分析、监控报警，最终根据数据采集、数据处理、数据服务三层的能力抽象出来多个数据应用，天眼可以做可视化分析、全链路监控、业务监控、故障管理，以及常见的数据管理、检索、报警，和开放的 API，让日志可以做到多种方式的应用到不同的场景。

产品化展示：



这是天眼的界面，包括实时 QPS 排名，以及多个业务方的域名，点击之后可以看到对应域名的整个实时大盘。左边栏为天眼能力的列表，通过我们的数据直通车、Ark 平台，对数据日志处理做了一站式的集成和能力的输出。

总结

① 做流数据处理，一定要先把流数据字典能力做出来，只有一个清晰的、完整的、可视的流数据字典能力才能让流数据应用的更好，方便用户的查询。

② 丰富的流处理能力，通过产品化的封装，比如 Spark streaming 和 Flink 有多种流处理能力，通过 Ark 封装，用户可以更简易、自主的配置。

③ 多场景适应能力，根据各种场景，把基础层提供的数据能力，做一个产品化的封装，满足不同场景应用的能力。除了刚刚讲的天眼日志分析外，流式处理还有各种的应用，如 AI 对实时的数据挖掘，实时的用户画像等。

今天主要为大家分享了贝壳找房的三个平台，以及流式数据处理的挑战和一些实战的经验。谢谢大家。

DataFunTalk

Apache Beam 架构原理及应用实践

嘉宾：张海涛　编辑：Hoh Xil

导读：大家好，很荣幸跟大家分享 Apache Beam 架构原理及应用实践。讲这门课之前大家可以想想，从进入 IT 行业以来，不停的搬运数据，不管职务为前端，还是后台服务器端开发。随着这两年科技的发展，各种数据库，数据源，应运而生，大数据组件，框架也是千变万化，从 Hadoop 到现在的 Spark、Flink，数据库从先前的 oracle、MySQL 到现在的 NOSQL，不断延伸。那么有没有统一的框架，统一的数据源搬砖工具呢？

带着这样的疑问，开始我们今天的分享，首先是内容概要：

Apache Beam 是什么？

Apache Beam 的优势

Apache Beam 的架构设计

Apache Beam 的核心组件刨析

AIoT PB 级实时数据,怎么构建自己的“AI 微服务”？

Apache Beam 是什么？

1. Apache Beam 的前世今生

大数据起源于 Google 2003 年发布的三篇论文 GoogleFS、MapReduce、BigTable 史称三驾马车，可惜 Google 在发布论文后并没有公布其源码，但是 Apache 开源社区蓬勃发展，先后出现了 Hadoop，Spark，Apache Flink 等产品，而 Google 内部则使用着闭源的 BigTable、Spanner、Millwheel。这次 Google 没有发一篇论文后便销声匿迹，2016 年 2 月 Google 宣布 Google DataFlow 贡献给 Apache 基金会孵化，成为 Apache 的一个顶级开源项目。然后就出现了 Apache Beam，这次不它不是发论文发出来的，而是谷歌开源出来的。2017 年 5 月 17 日 发布了第一个稳定版本 2.0。

2. Apache Beam 的定义

Apache Beam 的定义如上图，其定位是做一个统一前后端的模型。其中，管道处理和逻辑处理是自己的，数据源和执行引擎则来自第三方。那么，Apache Beam 有哪些好处呢？

Apache Beam 的优势

1. 统一性

① 统一数据源，现在已经接入的 java 语言的数据源有 34 种，正在接入的有 7 种。Python 的 13 种。这是部分的数据源 logo，还有一些未写上的，以及正在集成的数据源。基本涵盖了整个 IT 界每个时代的数据源，数据库。

② 统一编程模型，Beam 统一了流和批，抽象出统一的 API 接口。

③ 统一大数据引擎，现在支持性最好的是 flink，spark，dataflow 还有其它的大数据引擎接入进来。

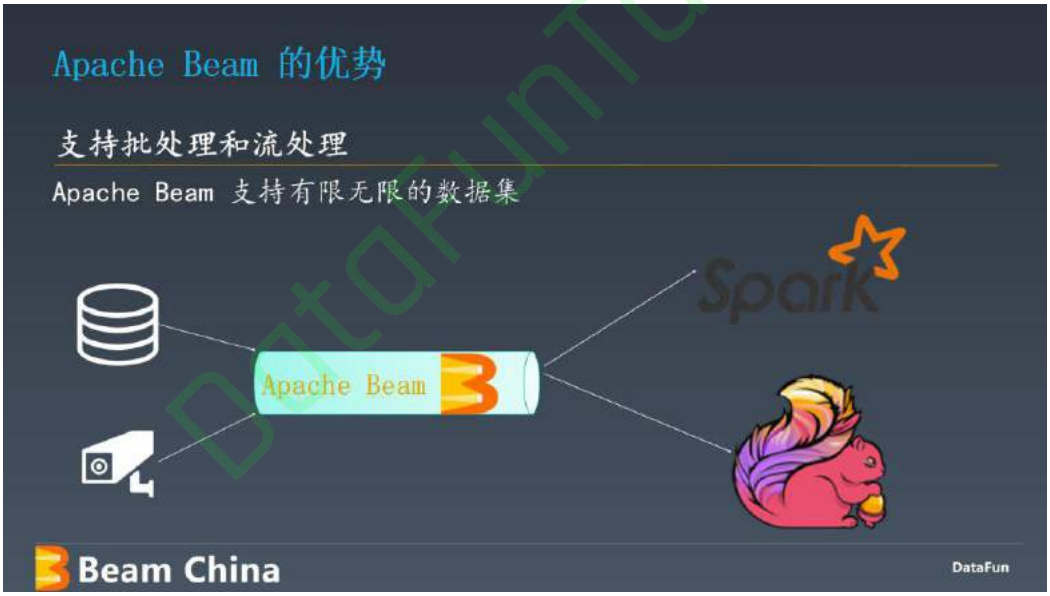
2. 可移植性

Beam 的 jar 包程序可以跨平台运行，包括 Flink、Spark 等。

3. 可扩展性

很多时候，随着业务需求的不断变化，用户的需要也随之变化，原来 Apache Beam 的功能可能需要进行扩展。程序员就会根据不同的需求扩展出新的技术需求，例如我想用 spark 新特性，能不能重写一下 sparkrunner 换个版本。我想重写一下 kafkaIO 可以吗？对于数据的编码，我可以自定义吗？最后干脆我感觉 Pulsar 技术不错，我想自己写个 SDKIO，集成进去可以不？答案都是可以的。Apache Beam 是具有可扩展性的，零部件都可以重塑。

4. 支持批处理和流处理



如果在 AIoT 行业，开发过程中，我们可能经常碰到两种数据：

- 摄像头等传感器的实时报警信息
- 不同数据库的数据，进行一起处理

Beam 对这两种数据是同时支持的。

5. 支持多语言开发

此外 Beam 支持 java，Python，go，Scala 语言，大家可以利用自己擅长的语言开发自己的 Beam 程序。

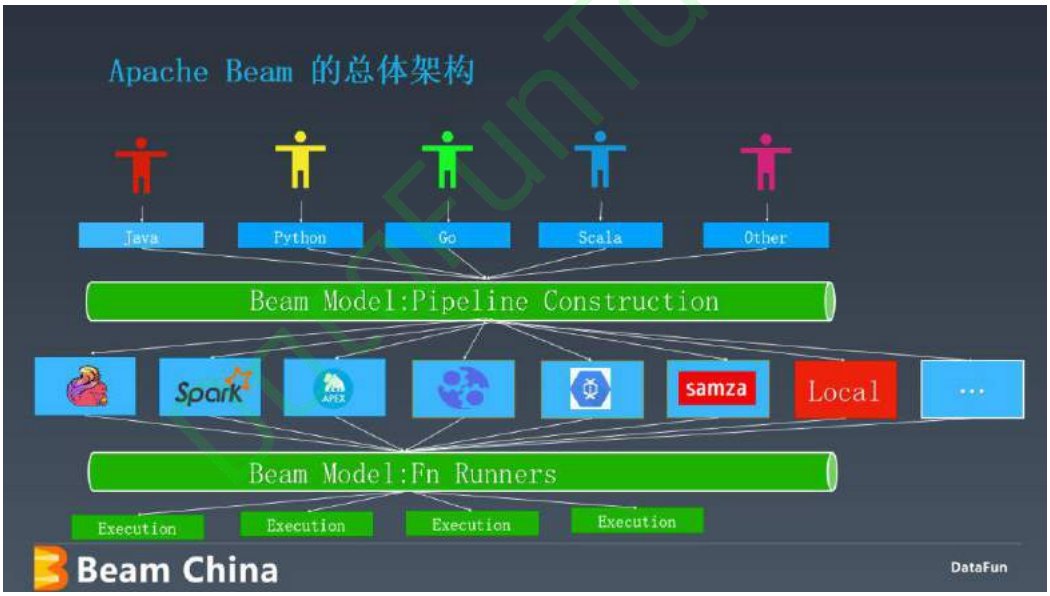
6. DAG 高度抽象

DAG，中文名“有向无环图”。“有向”指的是有方向，准确的说应该是同一个方向，“无环”则指够不成闭环。如果做一些去重、统计、分组等，开发人员不用再做 Map Reduce ，Beam 已经封装提供了相应的高级操作。

Apache Beam 的架构设计

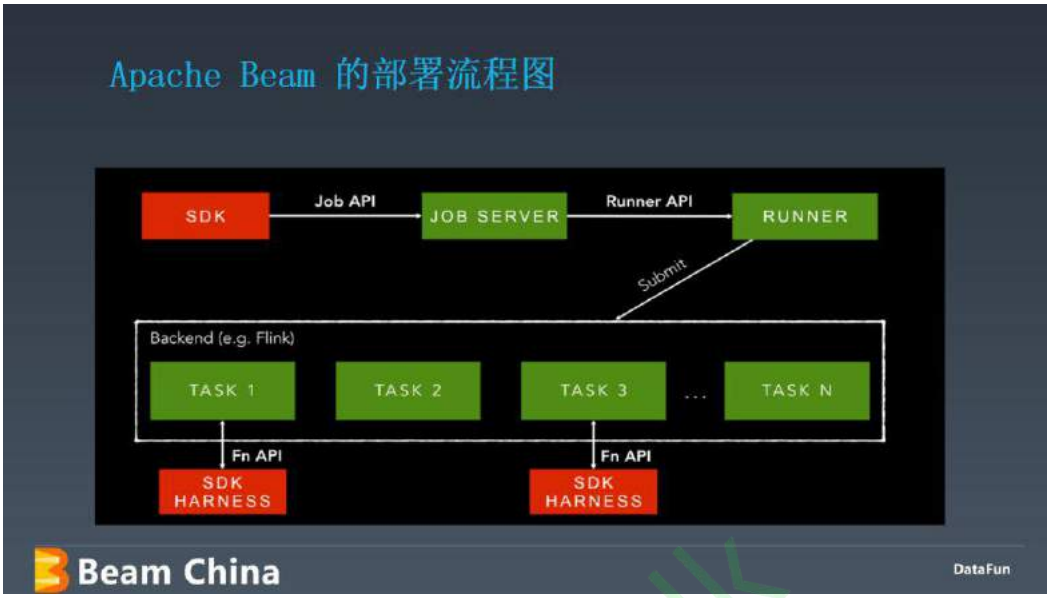
我们接下来看一下 Beam 架构是怎样的：

1. Apache Beam 的总体架构



Apache Beam 的总体架构是这样的，上面有各种语言，编写了不同的 SDKs，Beam 通过连接这些 SDK 的数据源进行管道的逻辑操作，最后发布到大数据引擎上去执行。需要注意的是，Local 虽然是一个 runner 但是不能用于生产上，它是用于调试/开发使用的。

2. Apache Beam 的部署流程图



让我们一起看下 Apache Beam 总体的部署流程。首先我们去构建这个 Beam jobAPI .jar 通过 job 服务器以及设置大数据执行平台，最后提交 flink 或 spark 的任务集群去执行任务。

Apache Beam 的核心组件刨析

1. SDks+Pipeline+Runners （前后端分离）

如上图，前端是不同语言的 SDKs，读取数据写入管道， 最后用这些大数据引擎去运行。可以发现完整的 beam 程序由 SDks+Pipeline+Runners 构成的。

2. 什么是 SDK？

DataFunTalk

什么是 SDK，就是一个编写 beam 管道构成的一部分，一个客户端或一个类库组件也可以，最后提交到大数据运行平台上。

3. Beam 版本和 Kafka-clients 依赖情况表

我们以 kafka 为例，看一下 Kafka-client 对版本的依赖情况，从图中可以看出 beam 2.6.0 版本的 api 改变基本是稳定的。当然，现在用的比较多的 2.4、2.5 版本。吐个槽，2.6 版本之前的兼容性问题，上个版本还有这个类或方法，下一个版本就没有了，兼容性不是很好。

4. SDK beam-sdks-java-io-kafka 读取源码剖析

```
pipeline.apply(KafkaIO.<Long, String>read()
        .withBootstrapServers("broker_1:9092,broker_2:9092")//
        .withTopic("my_topic")     // use withTopics(List<String>)
to read from multiple topics.
        .withKeyDeserializer(LongDeserializer.class)
       .withValueDeserializer(StringDeserializer.class)
```

```
        .updateConsumerProperties(ImmutableMap.of("group.id",
 "my_beam_app_1"))

         // set event times and watermark based on 'LogAppendTime'.
To provide   a custom
        // policy see withTimestampPolicyFactory().
withProcessingTime() is   the default.
        // Use withCreateTime() with topics that have 'CreateTime'
timestamps.
         .withLogAppendTime()
        // restrict reader to committed messages on Kafka (see
method   documentation).
         .withReadCommitted()
         // offset consumed by the pipeline can be committed back.
         .commitOffsetsInFinalize()
         // finally, if you don't need Kafka metadata, you can drop
it.g
        .withoutMetadata() // PCollection<KV<Long, String>>
      )
      .apply(Values.<String>create()) // PCollection<String>
```

```
PCollection<KV<Long, String>> kvColl = ...;
kvColl.apply(KafkaIO.<Long, String>write()
      .withBootstrapServers("broker_1:9092,broker_2:9092")
      .withTopic("results")
      .withKeySerializer(LongSerializer.class)
      .withValueSerializer(StringSerializer.class)
```

```
    .updateProducerProperties(ImmutableMap.of("compression.type", "gzip"))
    // You set publish timestamp for the Kafka records.
    .withInputTimestamp() // element timestamp is used while publishing to Kafka
    // or you can also set a custom timestamp with a function.
    .withPublishTimestampFunction((elem, elemTs) -> ...)
    // Optionally enable exactly-once sink (on supported runners). See JavaDoc for withEOS().
    .withEOS(20, "eos-sink-group-id");
```

① 指定 KafkaIO 的模型，从源码中不难看出这个地方的 KafkaIO<K,V> 类型是 Long 和 String 类型，也可以换成其他类型。

pipeline.apply(KafkaIO.<Long, String>read() pipeline.apply(KafkaIO.<Long, String>read()

② 设置 Kafka 集群的集群地址。

.withBootstrapServers("broker_1:9092,broker_2:9092")

③ 设置 Kafka 的主题类型，源码中使用了单个主题类型，如果是多个主题类型则用 withTopics(List) 方法进行设置。设置情况基本跟 Kafka 原生是一样的。

.withTopic("my_topic") // use withTopics(List<String>) to read from multiple topics.

④ 设置序列化类型。Apache Beam KafkaIO 在序列化的时候做了很大的简化，例如原生 Kafka 可能要通过 Properties 类去设置 ,还要加上很长一段 jar 包的名字。

Beam KafkaIO 的写法：

```
.withKeyDeserializer(LongDeserializer.class)

.withValueDeserializer(StringDeserializer.class)
```

原生 Kafka 的设置：

```
Properties props = new Properties();

props.put("key.deserializer","org.apache.kafka.common.serialization.ByteArrayDeserializer");

props.put("value.deserializer","org.apache.kafka.common.serialization.ByteArrayDeserializer");
```

⑤ 设置 Kafka 的消费者属性，这个地方还可以设置其他的属性。源码中是针对消费分组进行设置。

```
.updateConsumerProperties(ImmutableMap.of("group.id", my_beam_app_1"))
```

⑥ 设置 Kafka 吞吐量的时间戳，可以是默认的，也可以自定义。

```
.withLogAppendTime()
```

⑦ 相当于 Kafka 中 "isolation.level"，"read_committed"，指定 KafkaConsumer 只应读取非事务性消息，或从其输入主题中提交事务性消息。流处理应用程序通常在多个读取处理写入阶段处理其数据，每个阶段使用前一阶段的输出作为其输入。通过指定 read_committed 模式，我们可以在所有阶段完成一次处理。针对 "Exactly-once" 语义，支持 Kafka 0.11 版本。

```
.withReadCommitted()
```

⑧ 设置 Kafka 是否自动提交属性 "AUTO_COMMIT"，默认为自动提交，使用 Beam 的方法来设置。

```
set CommitOffsetsInFinalizeEnabled(boolean commitOffsetInFinalize)
```

```
.commitOffsetsInFinalize()
```

⑨ 设置是否返回 Kafka 的其他数据，例如 offset 信息和分区信息，不用可以去掉。

```
.withoutMetadata() // PCollection<KV<Long, String>>
```

⑩ 设置只返回 values 值，不用返回 key。例如 PCollection，而不是 PCollection<Long，String>。

```
.apply(Values.<String>create()) // PCollection<String>
```

在写入 Kafka 时完全一次性地提供语义，这使得应用程序能够在 Beam 管道中的一次性语义之上提供端到端的一次性保证。它确保写入接收器的记录仅在 Kafka 上提交一次，即使在管道执行期间重试某些处理也是如此。重试通常在应用程序重新启动时发生（如在故障恢复中）或者在重新分配任务时（如在自动缩放事件中）。Flink runner 通常为流水线的结果提供精确一次的语义，但不提供变换中用户代码的副作用。如果诸如 Kafka 接收器之类的转换写入外部系统，则这些写入可能会多次发生。

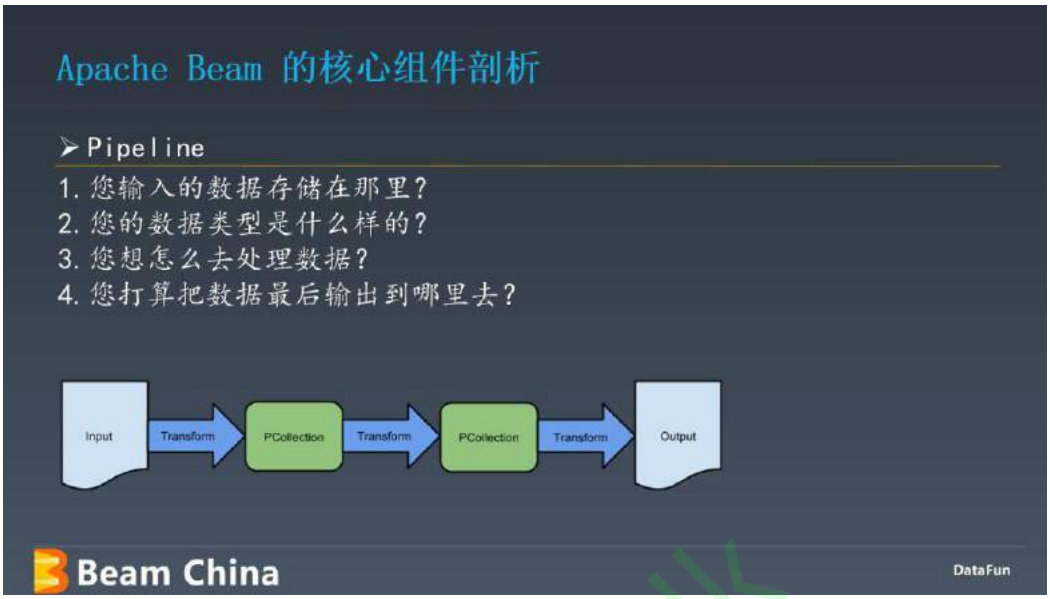
在此处启用 EOS 时，接收器转换将兼容的 Beam Runners 中的检查点语义与 Kafka 中的事务联系起来，以确保只写入一次记录。由于实现依赖于 runners checkpoint 语义，因此并非所有 runners 都兼容。Beam 中 FlinkRunner 针对 Kafka 0.11+ 版本才支持，然而 Dataflow runner 和 Spark runner 如果操作 kafkaIO 是完全支持的。

关于性能的注意事项：

"Exactly-once" 在接收初始消息的时候，除了将原来的数据进行格式化转换外，还经历了 2 个序列化 - 反序列化循环。根据序列化的数量和成本，CPU 可能会

涨的很明显。通过写入二进制格式数据（即在写入 Kafka 接收器之前将数据序列化为二进制数据）可以降低 CPU 成本。

5. Pipeline



您输入的数据存储在哪里？

首先要确定你要构造几条数据源，在 Beam 可以构建多条，构建之前可以选择自己的 SDK 的 IO。

您的数据类型是什么样的？

Beam 提供的是键值对的数据类型，你的数据可能是日志文本，格式化设备事件，数据库的行，所以在 PCollection 就应该确定数据集的类型。

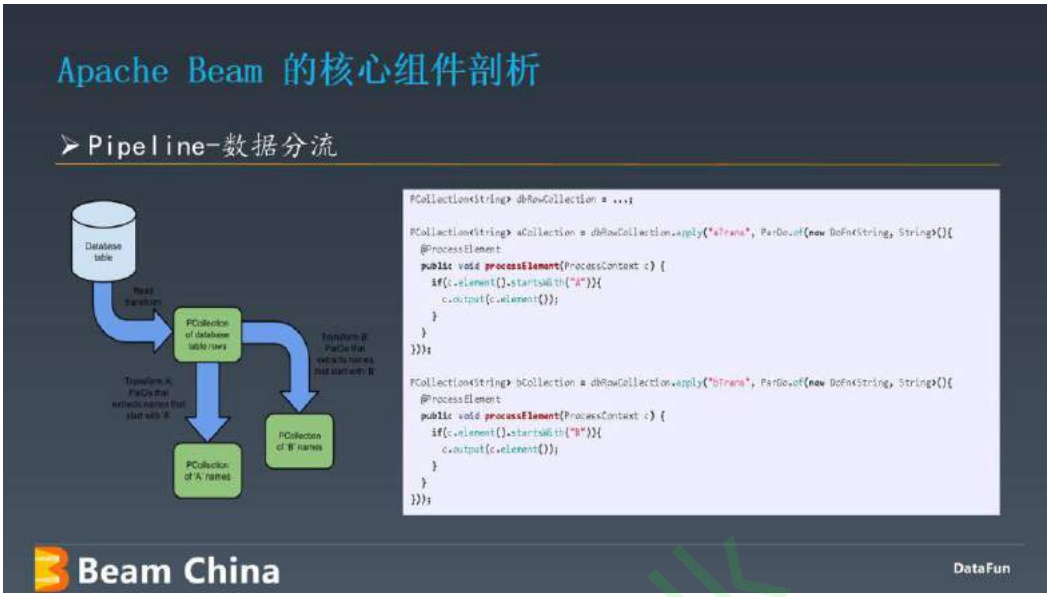
您想怎么去处理数据？

DataFunTalk

对数据进行转换，过滤处理，窗口计算，SQL 处理等。在管道中提供了通用的 ParDo 转换类，算子计算以及 BeamSQL 等操作。

您打算把数据最后输出到哪里去？

在管道末尾进行 Write 操作，把数据最后写入您自己想存放或最后流向的地方。



重要的是要理解变换不消耗 PCollections；相反，他们会考虑 a 的每个元素 PCollection 并创建一个新 PCollection 的输出。这样，您可以对不同的元素执行不同的操作 PCollection。这里是出现了两条管，例如输入 AR，AI，VAR，BT，BMP。

例如不同的数据源，有数据库，文件，以及缓存等输入进行合并。

一种是收费的拓蓝公司出品叫 Talend Big Data Studio，有没有免费的呢？

分享平台：DataFun 公众号：DataFunTalk

DataFunTalk

有的，它叫 kettle-beam。例如不同的数据源，有数据库，文件，以及缓存等输入进行合并。大家可以去 github 去看一下插件相应的安装及使用说明。从图中可以看出大部分 beam 的输入输出现在都是支持的。

https://github.com/mattcasters/kettle-beam

6. Runners

我们在看一下运行平台，这是运行平台支持度的截图。例如不同的数据源，有数据库，文件，以及缓存等输入进行合并。

Runners 在 Beam Model 模型中有 4 个支持的维度：

What，如何对数据进行计算？例如，机器学习中训练学习模型可以用 Sum 或者 Join 等。在 Beam SDK 中由 Pipeline 中的操作符指定。

Where，数据在什么范围中计算？例如，基于 Process-Time 的时间窗口、基于 Event-Time 的时间窗口、滑动窗口等等。在 Beam SDK 中由 Pipeline 的窗口指定。

When ,何时输出计算结果？例如 ,在 1 小时的 Event-Time 时间窗口中 ,每隔 1 分钟将当前窗口计算结果输出。在 Beam SDK 中由 Pipeline 的 Watermark 和触发器指定。

How，迟到数据如何处理？例如，将迟到数据计算增量结果输出，或是将迟到数据计算结果和窗口内数据计算结果合并成全量结果输出。在 Beam SDK 中由 Accumulation 指定。

① What

对数据如果处理，计算。分组的矩阵图，提到这里说一下，这些运行平台已经集成到 Beam，只是没有更新到官方首页而已。以及或者是官方不打算主推的，就没有写上去。

② Where

窗口处理矩阵能力图，大家从图中可以看出很多都是全部支持的。

③ When

对于事件处理，流计算引擎 Apache Flink，Google Cloud ，Dataflow 以及 Jstorm 都支持性比较好。

④ How

最后是对迟到数据的数据处理能力矩阵图。

7. FlinkRunner Beam

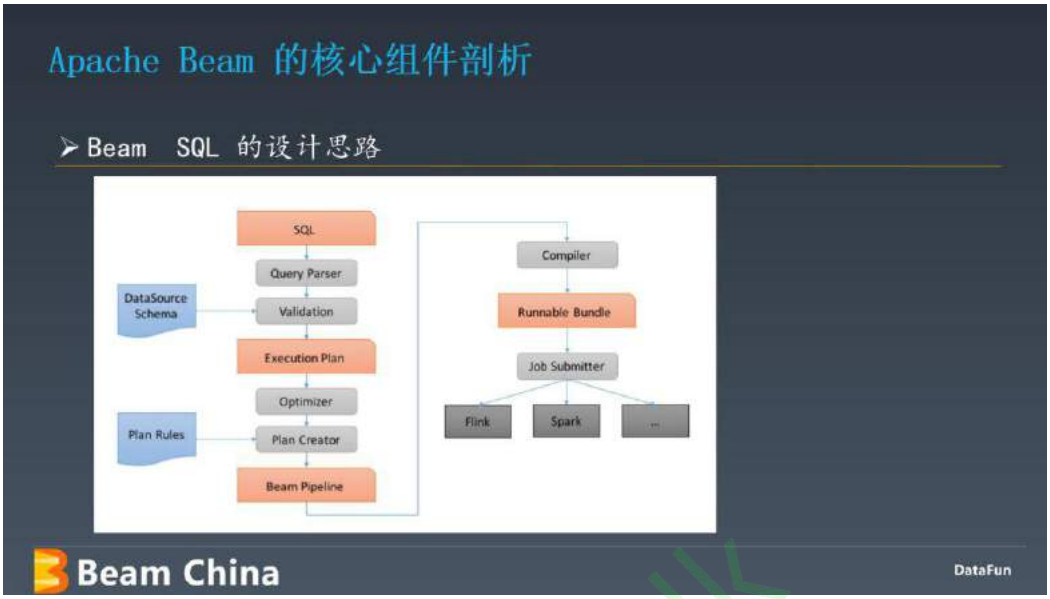
我们以最近两年最火的 Apache Flink 为例子，帮大家解析一下 beam 集成情况。

大家可以从图中看出，flink 集成情况。

然后看一下，FlinkRunner 具体解析了哪些参数，以及代码中怎样设置。

8. Beam SQL

Apache Calcite 是一种保准 SQL 的解析器，用于大数据处理和一些流增强功能，基于它做 SQL 引擎的有很多，例如 spark，Cassandra，druid 和我们的 Beam。

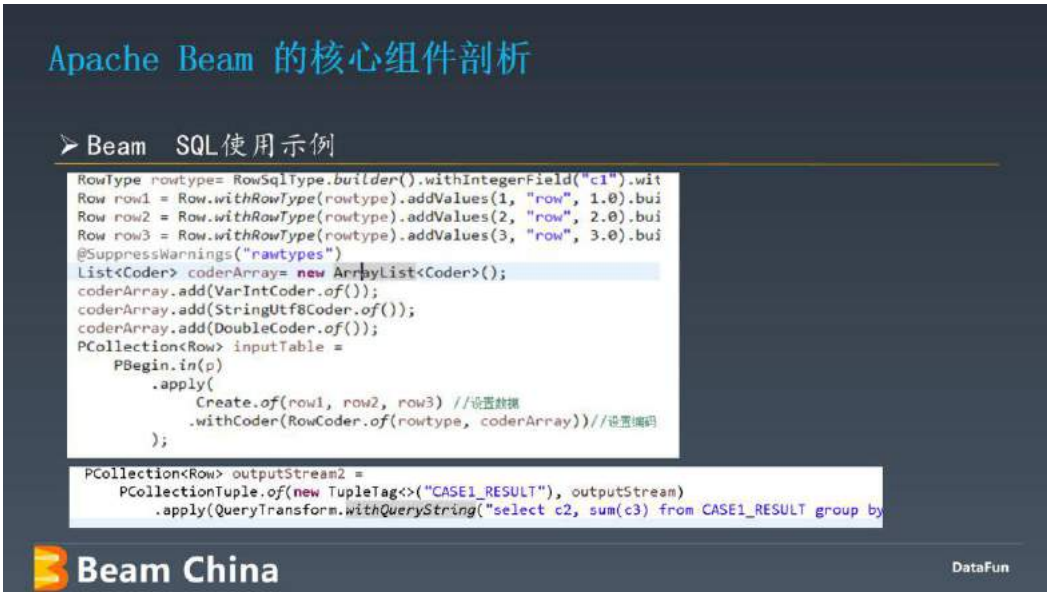


我们看一下 Beam SQL 的设计思路：首先是我们写的 SQL 语句，进行查询解析，验证来源的类型，数据格式，建一个执行计划，然后通过优化，设计计划规则或逻辑，封装在 Beam 管道中，进行编译器编译，最后提交 job 到运行平台执行。

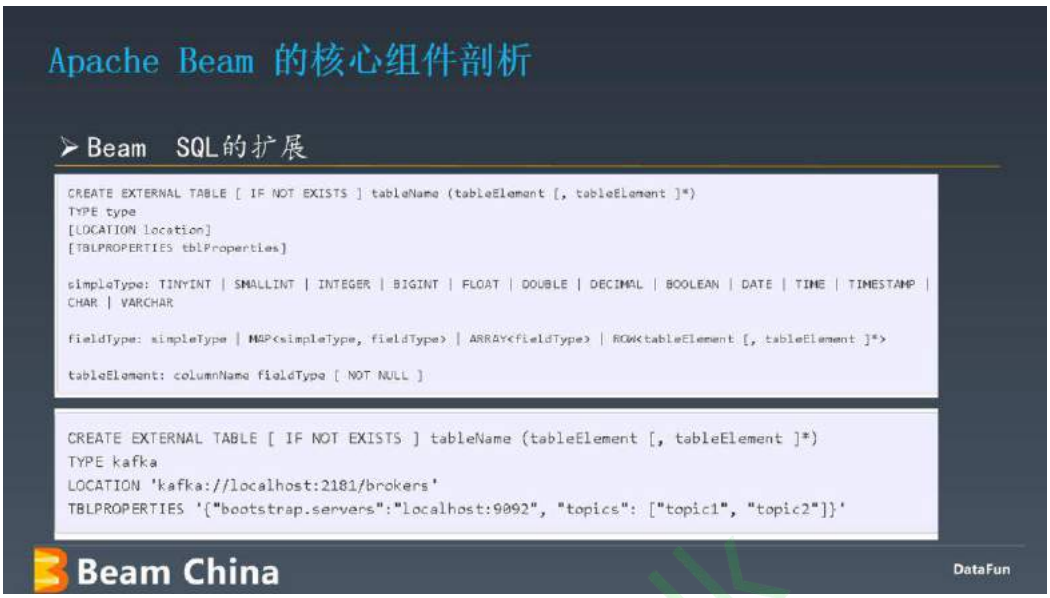
表中是 beam SQL 和 Calcite 的类型支持度，是把 Calcite 进行映射。

Beam SQL 和 Apache Calcite 函数的支持度。里面有一些现在不支持的，需要大家做的时候多多关注，特别是架构师设计时候。

DataFunTalk



从图中可以看出，首先要设置好数据类型，在设置数据，最后填充到管道数据集，最后做 SQL 的操作。其实这样写还是不方便的。有没有很好的解决方式，有。大家继续往下看...



Beam SQL 的扩展。Beam SQL 的 CREATE EXTERNAL TABLE 语句注册一个映射到外部存储系统的虚拟表 。对于某些存储系统，CREATE EXTERNAL TABLE 在写入发生之前不会创建物理表。物理表存在后，您可以使用访问表 SELECT，JOIN 和 INSERT INTO 语句。通过虚拟表，可以动态的操作数据，最后写入到数据库就可以了。这块可以做成视图抽象的。

Create 创建一个动态表，tableName 后面是列名。TYPE 是数据来源的类型，限制支持 bigquery，pubsub，kafka，text 等。Location 下面为表的数据类型配置， 这里以 kafka 为例。

AIoT PB 级实时数据，怎么构建自己的“AI 微服务”？

在 AIoT 里面，实时性数据比较大，例如视频分析，视频挖掘，合规检测，语音分析等等。130W 路的摄像头每秒写入 300 多 G 的视频，一天就是 25PB，有人说可以晚上用批方式上数据，其实 AIoT 场景跟其他的场景是不一样的，例如做智能儿童手表，我们晚上上报数据的频度可以变低，白天儿童上学放学路上可以正常上报数据。AIoT 场景下摄像头 24 小时监控的，并且宽带主杆线都换成千兆光线，其实也支持不了每秒 300G 的实时写入。我们是怎么处理呢？

首先在设计架构方案的时候，相信很多架构师都会这样想，不想第一个去吃螃蟹，因为稳定性 ,安全性 ,及不确定性原因会导致整个项目的成败。那我们看一下 Beam 有哪些大厂在使用。

知道他们使用 Beam ，咱们了解一下他们用 Beam 做了什么？例如：

- 使用 Apache Beam 进行大规模流分析
- 使用 Apache Beam 运行定量分析
- 使用 Apache Beam 构建大数据管道
- 从迁移到 Apache Beam 进行地理数据可视化
- 使用 Apache Beam & tf.Transform 对 TensorFlow 管道进行预处理

- 卫星图像的土地利用分类
- 智慧城市大数据集成
- 平安城市及质量实时风控
- 电商平台双十一活动实时数据处理

国外的可以从官方网站上找到案例的原文，国内可以从新闻或者官方网站找到相应的案例。

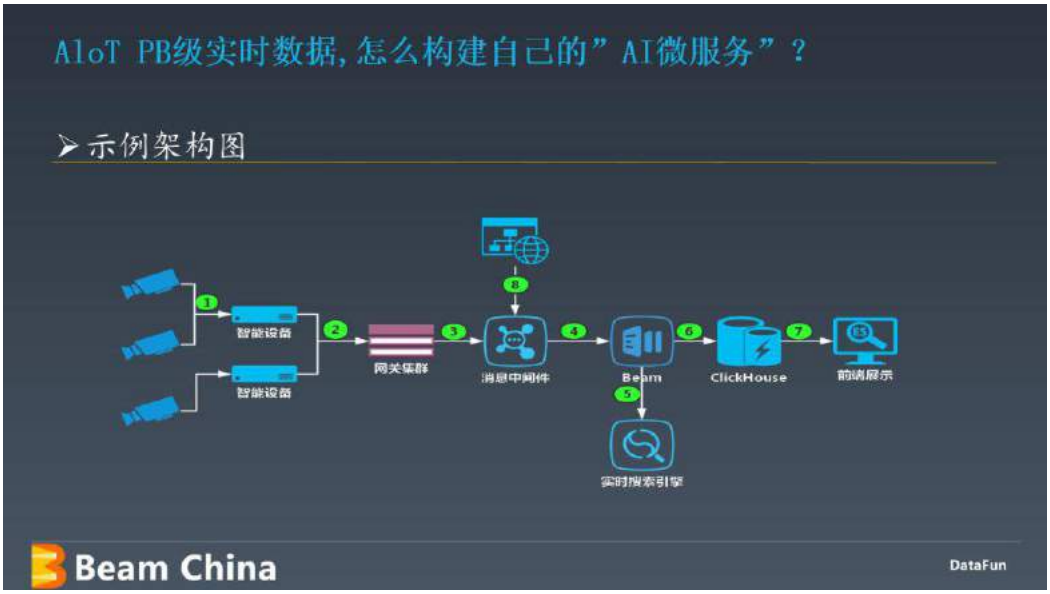
在 AIoT 场景下我们为什么会选择 Beam 呢？

- 数据源可以适配，因为平安城市，雪亮工程数据源千奇百怪。
- 能够进行数据多样处理，连接，过滤，合并，拆分。
- 具有清洗脏数据功能，例如警情去重误报警，合规检测等。
- 具有大数据集群虚拟化部署功能，可扩展性，伸缩性。
- 具有实时处理和离线处理能力。

1. 案列系统架构图

这是案例的总架构图，底层是 Beam SDK，上层是抽象封装的输入输出组件，以及清洗组件，类型管理，第三方 SDK，在往上层是组件配置管理，及版本控制，最上层是 jar 可视化配置，以及 SQL 可视化，最后把 jar 通过运维一体化平台提交给执行引擎集群，当然这里有个解析器，是我们自己开发的。

2. 示例架构图



以下为示例架构图：

① 摄像头以及 AI 智能设备产生的报警以及抓取的信息上报到后端智能设备。

② 智能设备产生的 AI 分析结果进行通过网关集群进行传输，注意网关集群地方要做流控及雪崩控制。

③ 消息通过网关集群发送到消息中间件。注意：这边这个规则下发是针对前段的数据进行 ETL 清洗的清洗规则的下发。

④ Beam 集群接收下发规则的更新，并且根据规则进行数据清洗。

⑤ 对于文档性的数据我们实时存储到实时搜索引擎。

⑥ 需要复杂查询，统计以及报表的数据存储到 ClickHouse。

⑦ 进行 BI 套件的展示以及前端大屏幕的展示。

3. 示例代码

```
public static void main(String[] args) {

// 创建管道工厂
PipelineOptions options = PipelineOptionsFactory.create();
// 显式指定 PipelineRunner: FlinkRunner 必须指定如果不制定则为本地
options.setRunner(FlinkRunner.class);
Pipeline pipeline = Pipeline.create(options);// 设置相关管道
// 这里 kV 后说明 kafka 中的 key 和 value 均为 String 类型
PCollection<KafkaRecord<String, String>> lines =
 pipeline.apply(KafkaIO.<String,String>read().withBootstrapServers("101.*.*.77:9092")// 必需设置
.withTopic("TopicAlarm")// 必需. 设置要读取的 kafka 的 topic 名称 .withKeyDeserializer(StringDeser
.updateConsumerProperties(ImmutableMap.<String, Object>of("auto.offset.reset", "earliest")));//
// 设置 Schema 的的字段名称和类型
final Schema type = Schema.of(
Schema.Field.of("alarmid", FieldType.STRING), Schema.Field.of("alarmTitle", FieldType.STRING),
Schema.Field.of("deviceModel", FieldType.STRING), Schema.Field.of("alarmSource", FieldType.INT3
        // 从 kafka 中读出的数据转换成 AlarmTable 实体对象
PCollection<AlarmTable> kafkadata = lines.apply("Remove Kafka Metadata", ParDo.of(new DoFn<Kafk
private static final long serialVersionUID = 1L;
@ProcessElement
public void processElement(ProcessContext ctx) {
Gson gon = new Gson();
```

核心示例代码，首先创建管道工厂，然后显示设置执行引擎，根据 SDKIO 进行读取 kafka 的消息。

```
public void processElement(ProcessContext ctx) {
Gson gon = new Gson();
AlarmTable modelTable = null;
try {// 进行序列号代码
modelTable = gon.fromJson(ctx.element().getKV().getValue(),AlarmTable.class);
} catch (Exception e) {
System.out.print("json 序列化出现问题，" + e);
}
ctx.output(modelTable);// 回传实体
}
}));
// 备份写入 Elasticsearch
String[] addresses = { "http://101.*.*.77:9200/" };
PCollection<String> jsonCollection=kafkadata
.setCoder(AvroCoder.of(AlarmTable.class))
.apply("covert json", ParDo.of(new DoFn<AlarmTable, String>() {
private static final long serialVersionUID = 1L;
@ProcessElement
public void processElement(ProcessContext ctx) {
Gson gon = new Gson();
String  jString="";
try {// 进行序列号代码
```


DataFunTalk

序列化消息，写入 es 进行备份，因为 es 数据是 json 的写入的时候首先要考虑转换成 json 类型。这个地方我设置了一个编码，实体类的编码类型为 AvroCoder ，编码类型是每个管道都要设置的。

```
try {// 进行序列号代码
jString = gon.toJson(ctx.element());
System.out.print(" 序列化后的数据：" + jString);
} catch (Exception e) {
System.out.print("json 序列化出现问题：" + e);
}
ctx.output(jString);// 回传实体
}
}));
// 所有的 Beam 数据写入 ES 的数据统一转换成 json 才可以正常插入
jsonCollection.apply( ElasticsearchIO.write().withConnectionConfiguration(ElasticsearchIO.Conne
));

PCollection<Row> modelPCollection = kafkadata
//.setCoder(AvroCoder.of(AlarmTable.class))// 如果上面设置下面就不用设置
.apply(ParDo.of(new DoFn<AlarmTable, Row>() {// 实体转换成 Row
private static final long serialVersionUID = 1L;
@ProcessElement
   public void processElement(ProcessContext c) {
AlarmTable modelTable = c.element();
System.out.print(modelTable.getAlarmMsg());
Row alarmRow = Row.withSchema(type)
    .addValues(modelTable.getAlarmid(),modelTable.getAlarmTitle(), modelTable.getDeviceModel(),
c.output(alarmRow);
}
```

把 kafka 的数据转换成 row 类型，这里就是运用了管道设计中的流分支处理。

```
}));
      // 写入 ClickHouse
modelPCollection.setRowSchema(type).apply(
ClickHouseIO.<Row>write("jdbc:clickhouse://101.201.56.77:8123/Alarm", "AlarmTable").withMaxRetri
.withMaxInsertBlockSize(5) // 添加最大块的大小
.withInitialBackoff(Duration.standardSeconds(5))
.withInsertDeduplicate(true) // 重复数据是否删除
.withInsertDistributedSync(false));
pipeline.run().waitUntilFinish();
}
```

最后一步是写入咱们的 clickhouse，大家可能对 clickhouse 不是很了解，这是俄罗斯的一家高科技公司研发的。查询速度非常快 ,比 Hive 快 279 倍 ,比 MySQL 快 801 倍的神器。

4. 示例效果展示

以下为写入 es 的效果。这个字段写入时候自动创建。

今天的分享就到这里，谢谢大家。

阿里小蜜新一代智能对话开发平台技术解析

嘉宾：李维　编辑：杨辉之

导读：本次分享的内容是“新一代智能对话开发平台”（Dialog Studio），中文名叫对话工厂，是小蜜家族中的一个做多轮任务型对话开发产品，目标是以平台的方式赋能更多开发者，让所有的对话者能创造自己的多轮对话机器人。大概分为四个部分，主要包括：

- 为什么做平台？
- “新一代”对话开发平台
- “一站式”对话开发平台
- 业务应用

1. 为什么做平台

首先给大家举个例子：

机器人：您好，这里是 114 移车热线，您的车挡住了出行，请您移下车好吗？

车主：哦，哪个车子啊？

DataFunTalk

机器人：车牌是浙 A，尾号是 29，请问是您的车吗？

车主：啊，什么情况？

机器人：这手机登记的是您的号码，您能去看一下吗？

车主：什么号码，我，你再说一遍

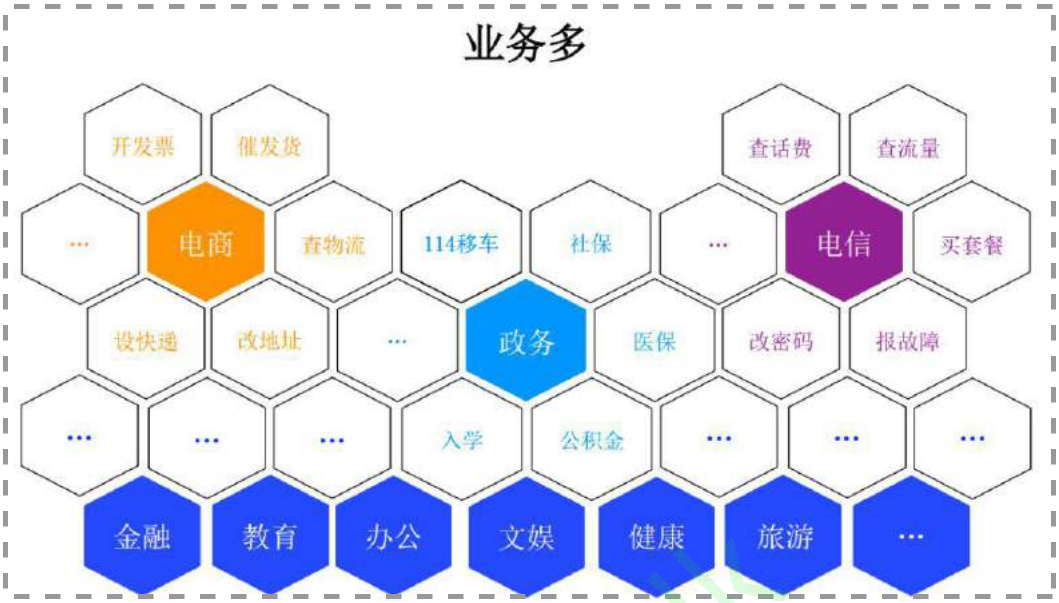
机器人：车牌是浙 A，尾号是 29，请问是您的车吗？

车主：哦，是我的，我打个电话

机器人：那请您移一下车好吗？

车主：嗯

这是一个政务场景上 114 移车的 case，这个场景是用我们“对话工厂”开发的机器人实现的。114 移车 case 只是一个我们真实生活中很小的一个点，还有很多的业务是需要对话机器人来帮助人类解决各种各样的问题，或者减少人类客服成本。比如说在政务的其他领域，我们希望机器人能帮助人来查社保、查医保、查公积金等；在电商领域，我们希望机器人能帮助做售后的客服，帮助消费者来开发票、催发货、查物流等；在电信领域，我们希望机器人能引导用户查话费、查流量等。



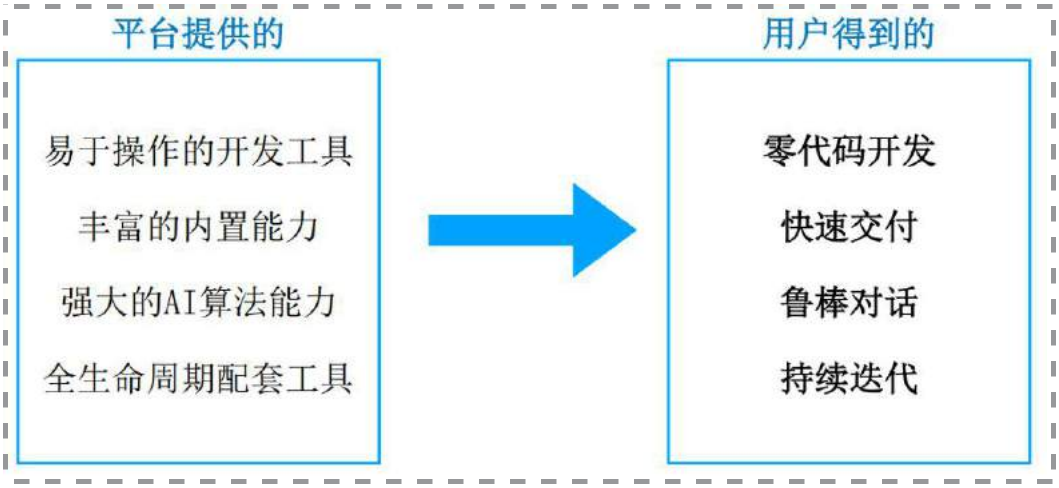
但是创造一个对话机器人的门槛是非常高的，做一个 demo 很简单，但能解决实际问题的机器人很难。假设我们把开发对话机器人的整个过程称为一个完整的生命周期，在每一步都会遇到很多困难，在开发阶段，冷启动很困难，根本不知道用户的真实表述会是什么样的，假设有了话术后，自然语言理解和对话管理也很难，用户可能根本不按预设的方式进行表述，方式千变万化，普通方式无法理解。在调试测试阶段，也面临着调试对话难、测试覆盖难和人工评测难的问题，是一件繁琐、费时费力的工作。在发布阶段，版本控制，代码回滚，线上部署也需要考虑怎么做到自动化和一键化。到了闭环阶段，线上状态监控、可视化数据分析、数据自动标注，机器人持续优化迭代也是一个难题。所以在整个生命周期中，如果以一个独立开发者的身份来做对话机器人是一件很难的事情，门槛很高。

基于以上两点：

1、需要对话机器人的业务很多；

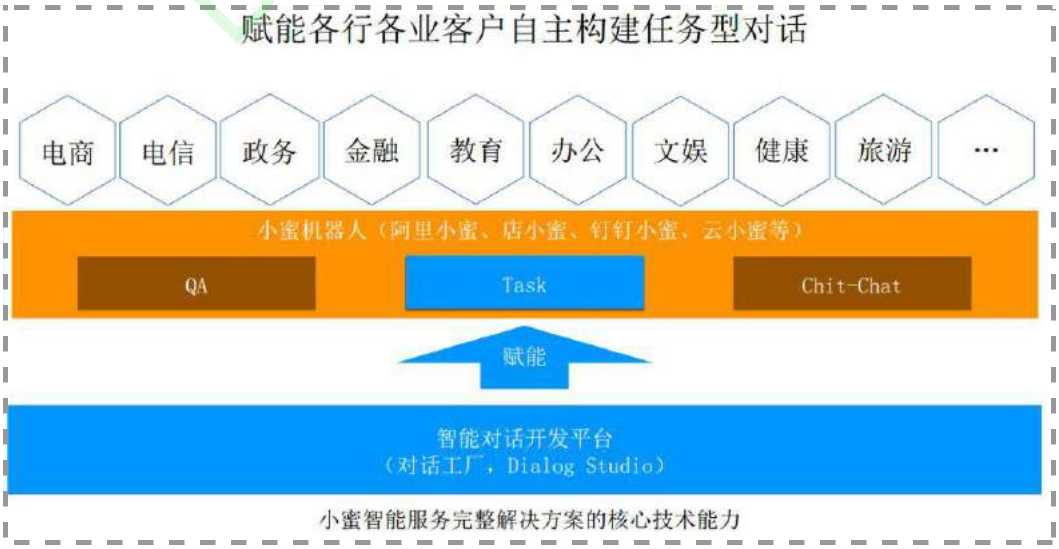
2、独立开发者的开发成本又很高。

我们就想做一套平台产品来赋能开发者。如果我们做一个平台能够提供一些非常易于操作的开发工具，有丰富的内置能力，有强大的 AI 算法能力，以及全生命周期的配套工具，那么我们这些独立的开发者或者企业就能够做到零代码开发，快速交付具有鲁棒对话的机器人，并且该机器人可以在线上进行持续迭代优化。



平台的好处

基于这样的想法，我们打造了对话工厂（Dialog Studio）这个产品，属于小蜜机器人解决方案中的一个核心基础能力，我们已经嵌入到小蜜家族（阿里小蜜，店小蜜，钉钉小蜜，云小蜜等）的各个平台中，支持了包括阿里电商，电信，政务，金融，教育等领域的各个场景的对话机器人业务。这就是我们为什么要做一个平台型对话机器人产品的初衷。



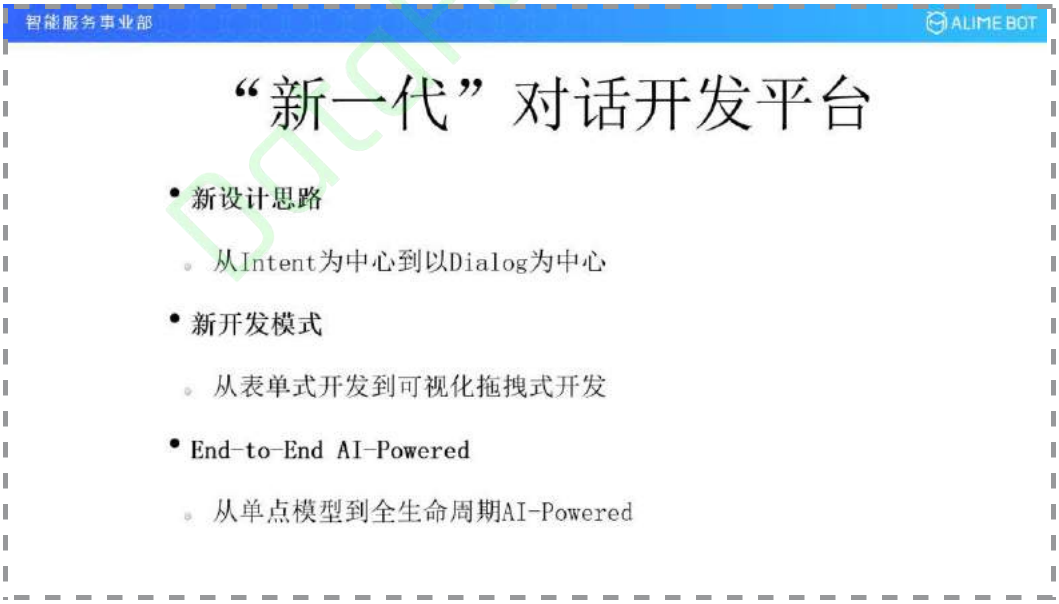
2. “新一代”对话开发平台

我们把这个平台定义为“新一代”对话开发平台，为啥叫“新一代”，到底它新在哪。我们大概总结了三点：

第一个是新的设计思路：将原来以 Intent 为中心的设计思路转变为以 Dialog 为中心的设计思路。

第二个是新的开发模式：从原先表单开发方式转变为可视化拖拽式开发。

第三个是 End-to-End AI-Powered：从原来只提供单点的 NLU、DM 模型 AI 赋能方式转变为 End-to-End AI-Powered 平台，在整个生命周期的各个阶段都会用 AI 来赋能开发者。



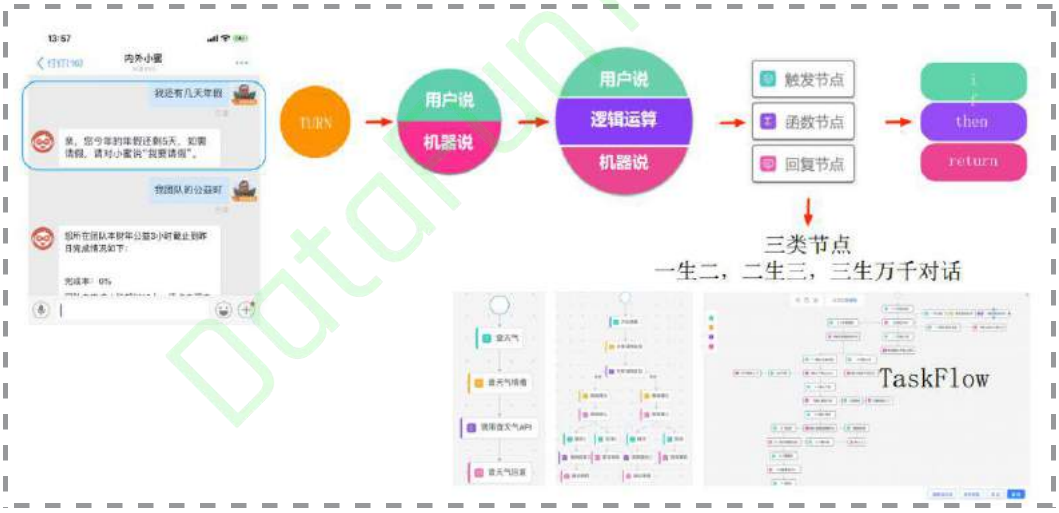
下面针对这三点进行具体介绍：

2.1 从 Intent 为中心到以 Dialog 为中心

以 Intent 为中心的方式每次要创建单个的意图，如果遇到一个复杂场景，需要创建很多个意图堆积在一起，每次开发只能看到当前正在创建的意图，因为该技术起源于原来的 Slot Filling 方式，只能解决简单的，Slot Filling 这种任务形式对话，任务场景比较受限。而以 Dialog 为中心的设计思路，把人机对话过程通过图的方式展现出来，对话步骤用图上的节点进行抽象，开发者在设计对话流的时候，这种方式能提供一个全局的视野，而且图的节点抽象比较底层，可以进行各种任务的配置，适用场景比较广泛。

从以 Intent 为中心到以 Dialog 为中心

接下来我们具体介绍下对这些对话怎么抽象出图和节点的概念，可以从简单的一轮对话场景来说，一般的情况下一个人和机器的对话场景可能是人说一句话，机器说一句话。在这个例子当中，人问机器人：我还有几天年假，然后机器人回复：亲，您今年的年假还剩 5 天，等等。这一轮对话可以抽象成两个元素，一个元素叫用户说，一个元素叫机器人说。但是在好多产品中，用户的答案不是简单的静态答案，直接能回答的，需要机器人有一个思考过程，会进行一些逻辑的判断，比如说查看服务，做一些逻辑分支判断，这个时候我们在前面两个元素基础上增加一个逻辑运算元素，把这三个元素抽象定义为“对话工厂”画布上三种节点类型，分别为“触发节点”、“函数节点”和“回复节点”，从代码的角度来看就是“if”、“then”和“return”。所以整个抽象过程就是用这三类节点，一生二，二生三，三生万千对话，可以配置出各种业务场景，可以是简单的线性场景和复杂的、带各种分支的场景。把这种由节点定义的图称为 TaskFlow，它定义了一个对话机器人的所有任务。



重新对 Dialog 进行抽象

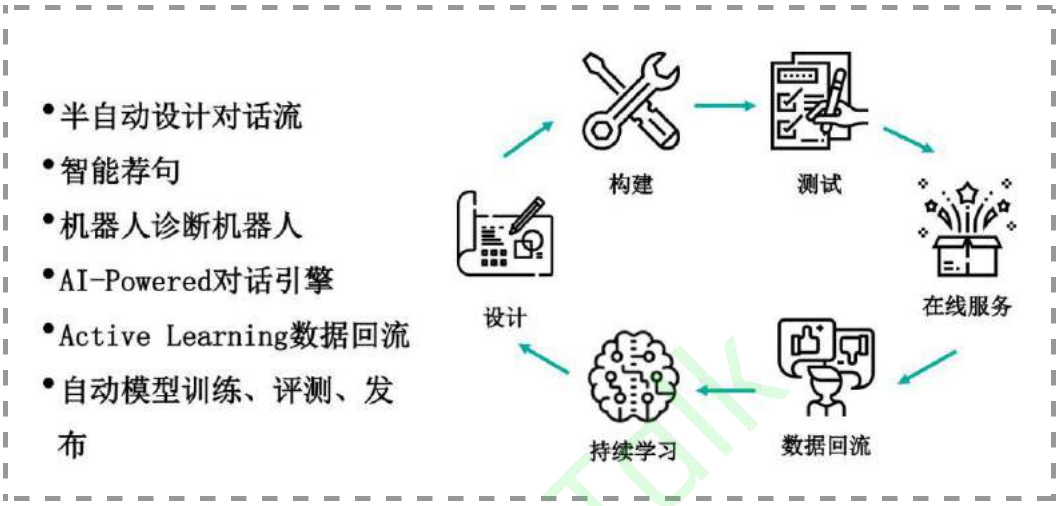
2.2 从表单式开发到可视化拖拽式开发

在开发模式上将原来的表单式开发方式变成了可视化拖拽式开发方式。原来表单式开发方式以 Intent 为中心，所以对于开发者来说更像做一道一道表单填空题，只能单点控制，整个对话流程非常不直观，所有 Intent 压缩在一个表单，填写复杂。在可视化拖拽方式中，整个对话流过程的每一个节点都可以通过简单拖拽方式进行完整描述，拥有全局视野，整体可控。

从表单式开发到可视化拖拽式开发

2.3 从单点模型到全生命周期 AI-Powered

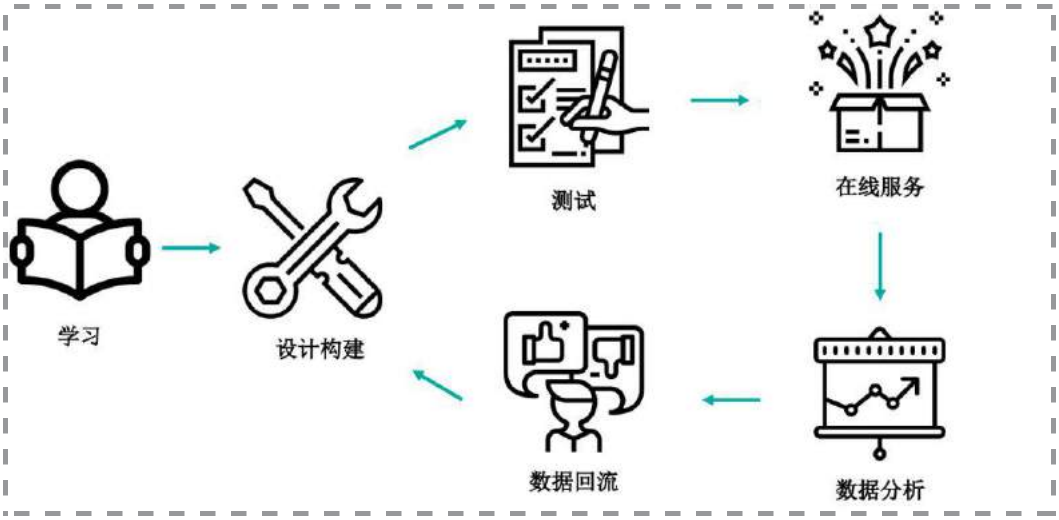
第三点是从原来单点 NLU、DM 模型到全生命周期 AI-Powered，在对话机器人开发的各个阶段都利用了 AI 算法能力赋能开发者，加速开发过程和降低开发成本。在设计阶段尝试了半自动对话流设计辅助，让开发者从冷启动阶段就能够设置出第一版的对话流。在对话流的构建阶段，我们推出了智能荐句功能，也就是说在编写用户话术的时候，机器人可以进行话术推荐和联想。在测试阶段，我们推出了机器人诊断机器人功能，可以大大减少测试的工作量，增速测试过程。在在线服务阶段，会有全套的 AI-Powered 对话引擎，包括 NLU、DM 等算法。在数据回流阶段，通过 Active Learning 将日志数据进行自动标注，用于后续模型迭代训练。在持续学习阶段，我们构建了一套完整自动模型训练、评测、发布 Pipeline，自动提升线上机器人效果。



End-to-End AI-powered 平台

3.“一站式”对话开发平台

接下来我们聊一聊为什么叫“一站式”对话开发平台，因为我们是对对话机器人开发的整个生命周期进行赋能，不是单点开发过程。在“一站式”的对话平台我们提供了哪些能力来赋能开发者呢？假设把开发对话机器人的完整生命周期简化为以下几个阶段，首先需要学习相关知识；然后基于一定的知识储备在平台上设计构建机器人；接着对机器人测试和调试；然后就可以将其在线发布，提供对话服务；Serving 过程中可以对线上数据进行可视化分析；同时可以通过数据回流划分 bad case 和 good case，进行线上持续学习迭代。



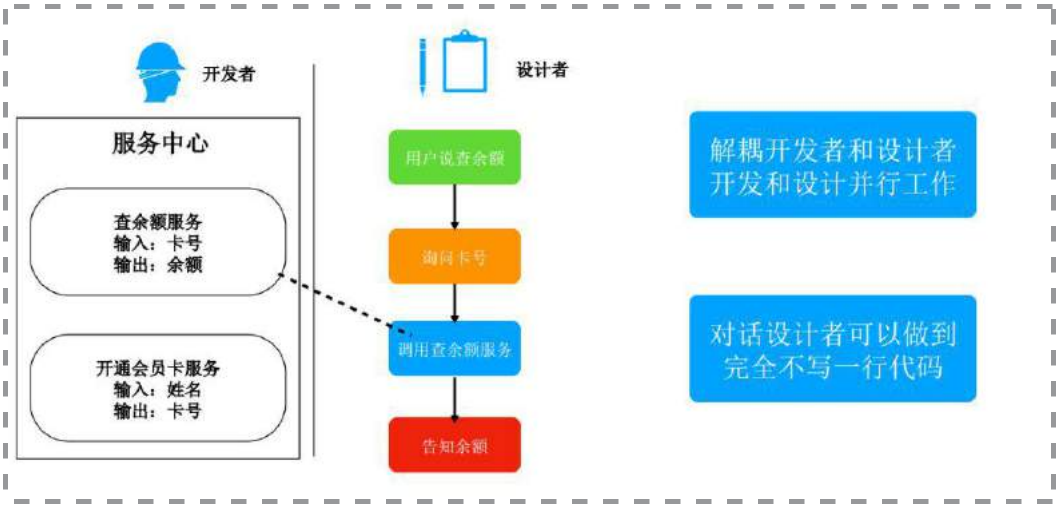
开发对话机器人的完整生命周期

在上面几个阶段我们都提供了强有力的辅助和能力对开发者赋能。在设计构建阶段提供了 AI 智能荐句能力，用户只需要输入种子话术，平台就会自动推荐相似问法，能节省人工造句和打字时间。

构建对话流动态图时只需要拖拽式操作，在画布中将节点和边连在一起，可视化编程，对话逻辑一目了然，且条件分支简单配置，可以根据业务灵活定制。在该阶段通过服务注册中心的方式也解耦了开发者和设计者，开发者将服务注册到中心，设计者从中心发现并使用服务，他们基于商定好的协议沟通，开发和设计可以并行工作，使对话设计者可以做到完全不写一行代码，开发者专注于功能服务开发。

设计构建阶段：AI 辅助加速开发

设计构建阶段：拖拽式操作



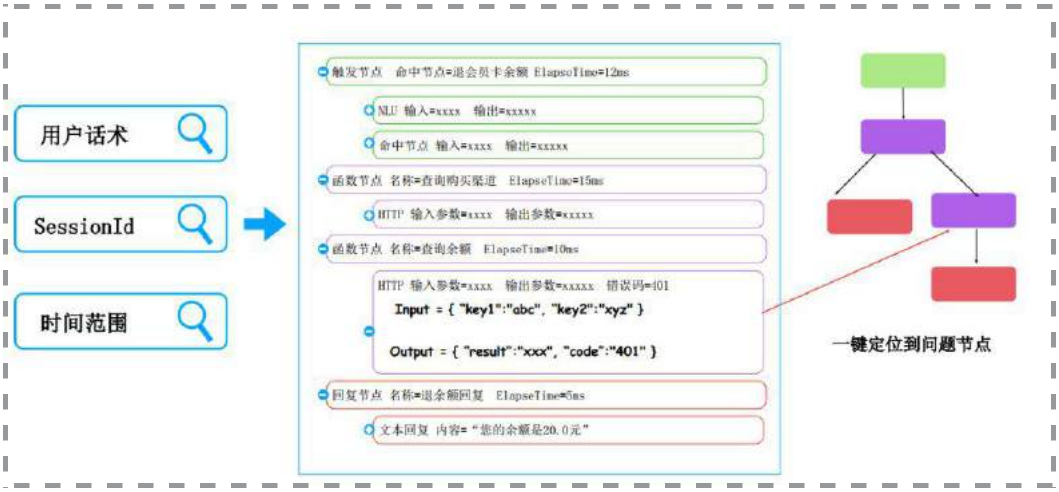
设计构建阶段：Code Free 开发

在对话流测试阶段，平台能自动的对节点、分支和连线进行合法性校验，类似 coding 时 IDE 做的语法检查和静态编译检查，可以一键定位静态问题位置。在动态测试时，可以使用机器人诊断机器人，智能构建测试 case，全自动批量测试，最多可以同时跑几千个 case，并可以在几秒内完成，并生成测试报告一键定位动态问题进行修复。在遇到线上 bug 时，可以利用平台提供的链路排查工具，在这个工具里，只需要输入用户当时的话术、SessionId 或时间范围，会以 Stack Trace 的方式把错误呈现出来，比如某个节点的输入输出是啥，是否报错，可以一键定位问题节点。

测试阶段：自动校验合法性

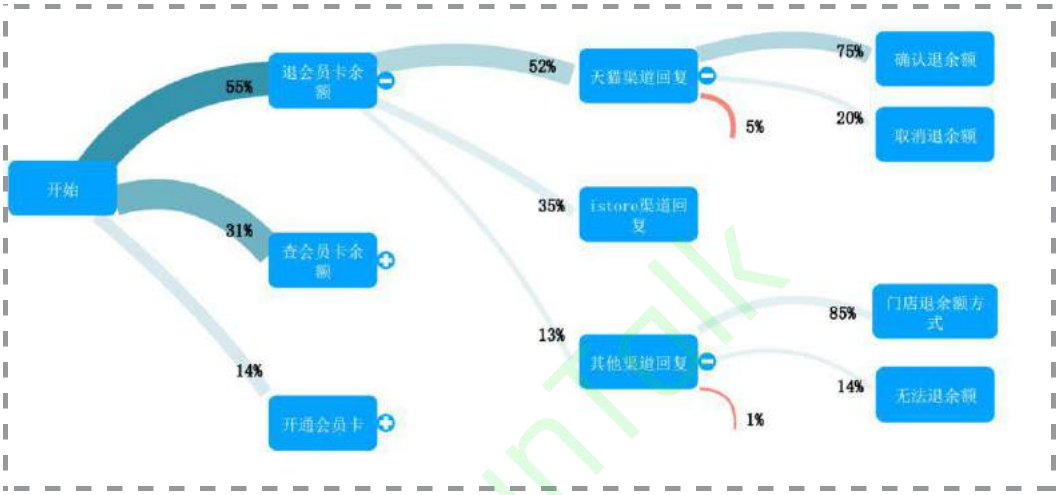
测试阶段：机器人诊断机器人

DataFunTalk



测试阶段：链路排查，方便定位问题

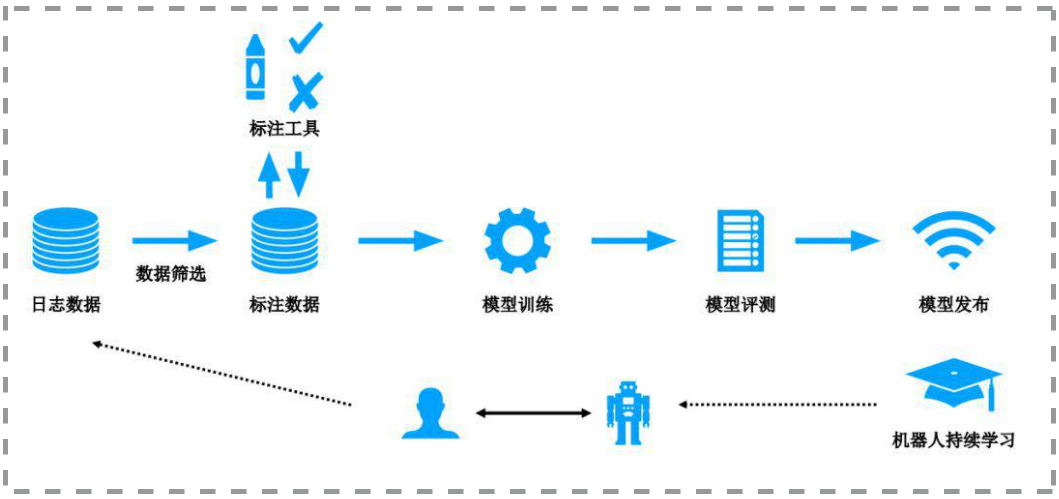
针对可视化报表，平台也提供了桑吉图、饼状图、表格等图表以非常直观的方式展示给用户对话机器人的运行情况，对话效果一目了然。



智能分析报表，对话效果一目了然

DataFunTalk

当线上机器人发现一些 bad case 或者积累足够多的线上数据的时候，平台也提供了一套全自动的模型持续学习的 Pipeline ,从人机交互日志数据回流、数据筛选、标注数据、模型训练、模型评测到模型发布的一个完整闭环。



数据回流闭环，持续迭代优化机器人

4. 业务应用

最后介绍对话工厂在真实场景的应用，对话工厂接入了阿里小蜜中的各个生态，在阿里内部的事业部场景中，比如手淘小蜜、优酷小蜜、河马小蜜、飞猪小蜜等，在商家和企业的业务场景中，比如店小蜜和钉钉小蜜，在阿里经济体外部业务场景中，比如云小蜜，他们客户端的多轮任务型对话都是通过对话工厂提供支持和赋能的。

构建端到端的联邦学习 Pipeline 生产服务

嘉宾：曾纪策　编辑：Hoh Xil

导读：本次分享的主题为构建端到端的联邦学习 Pipeline 生产服务。联邦学习的优势在于能够保证参与各方在数据不出本地，保持数据独立性的情况下，多方共建模型，共同提升机器学习效果。联邦机制下，安全隐私有了优势，但技术上也会面临更多挑战。作为一个工业级的框架，端到端的联邦学习 Pipeline 致力于完成高弹性、高性能的联邦学习任务，主要包括建模、训练、模型管理、生产发布和在线推理几个方面。本次将为大家分享如何灵活调度管理复杂的联邦学习任务、可视化联邦建模的实现以及在线联邦推理服务的思考与实践，解决实验性机器学习到实际生产应用落地的难点。

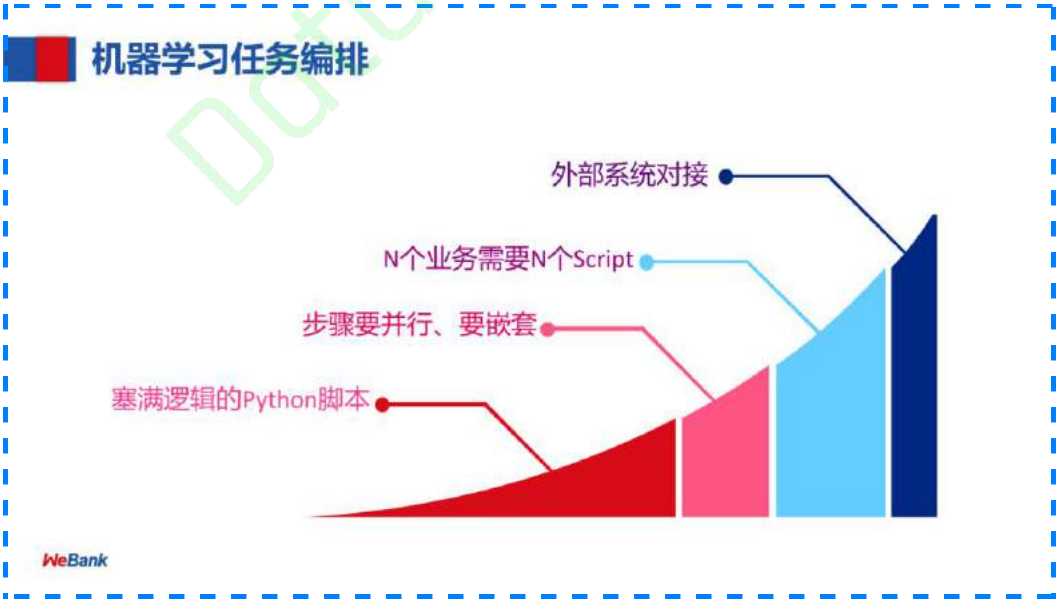
主要分享 4 个方面：

- 背景介绍
- 高弹性联邦学习 Pipeline 调度
- 联邦学习任务可视化
- 高性能联邦学习在线推理服务

DataFunTalk

我们在日常建模过程中，会遇到哪些需求和痛点？

1. 机器学习任务编排



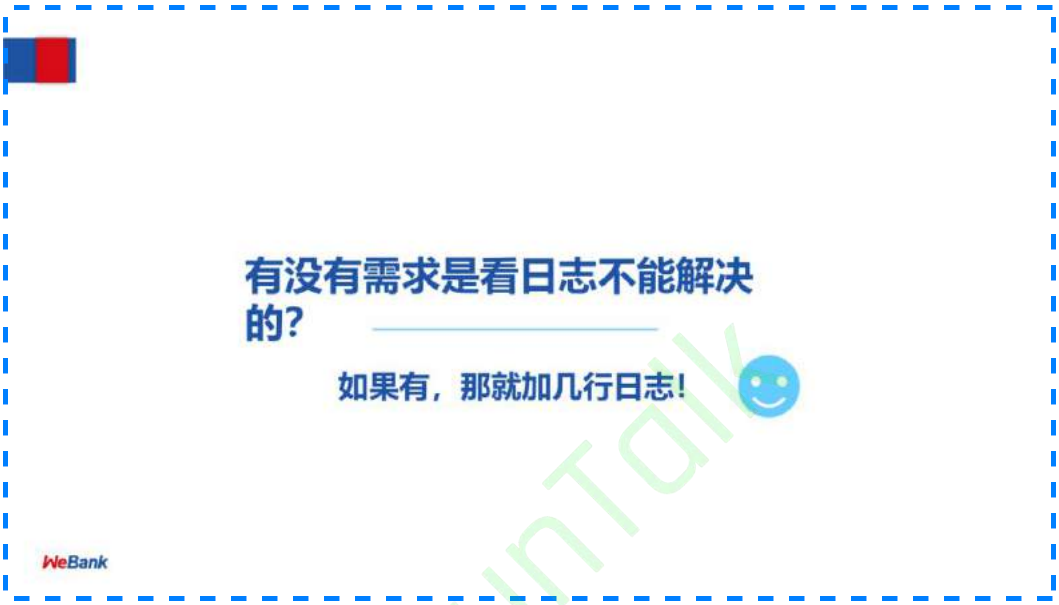
我们是如何编排机器学习任务的：

- 塞满逻辑的 Python 脚本。把机器学习的一些步骤，如特征工程、预处理、验证等，写成一个符合逻辑的 Python 脚本。
- 步骤要并行、要嵌套。通过多线程、多进程等手段来完成。
- N 个业务需要 N 个 Script。生产中我们有非常多的业务，每个业务采用的算法也不同，所以会有多个 Script。
- 外部系统对接。一般其他平台在发起自动化机器学习平台任务时，系统对接上是非常困难的，毕竟还没有那么智能，那么如何更好的写好 Python 脚本呢？

2. 机器学习任务状况观察

当我们千辛万苦的把任务运行起来，还需要不断观察任务的运行情况，运行到哪一步？每一步的数据输出是什么样的？指标输出是什么样的？Loss、AUC 等指标的趋势？然后，任务跑完了吗？

我们该如何观察到各种趋势，来尽可能做参数的调整等操作。这里要大家可以想想，平时是如何观察机器学习任务状态的呢？



如果是我，可能就会看日志。那有没有需求是看日志不能解决的？如果有，那就加几行日志！^_^

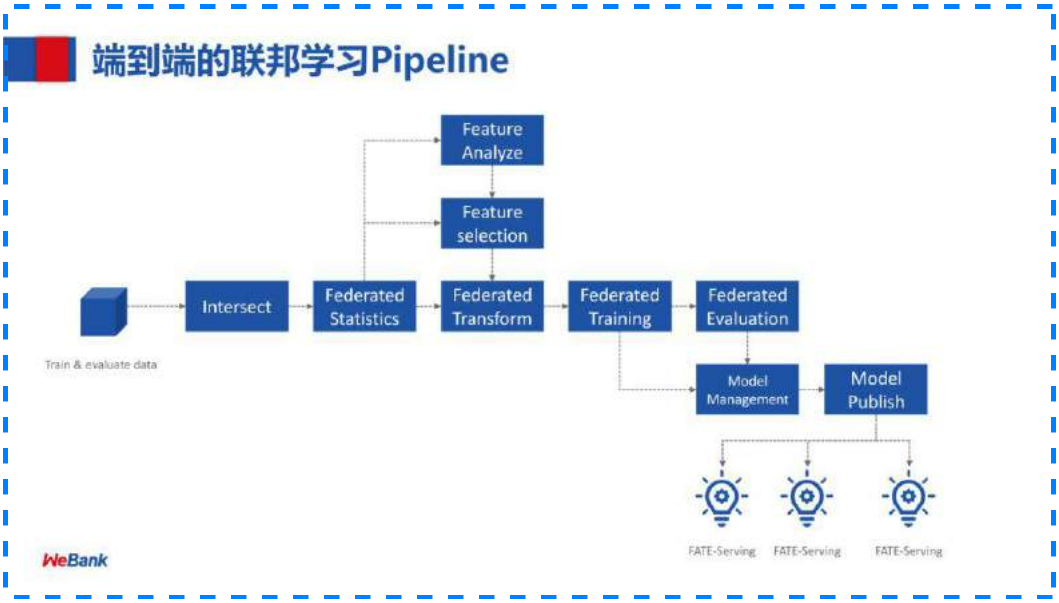
3. 联邦学习任务协同建模

刚刚介绍的都是一般机器学习任务中会遇到的挑战，在联邦机器学习下，因为涉及多方协同，会遇到更多的挑战：

- 启动多方任务。如需要跟多方协调启动任务，时间难同步。
- 各方的运行状况。需要通过通信工具，同步任务进度。
- 多方日志、排查问题。如日志是散落的，各个操作系统遇到的问题也不一样。
- 多方任务管理。可能非联邦学习的建模只需要 1 个人，而联邦机制下需要 3 个人或者更多。

接下来分享下 FATE 是如何尝试解决这样的问题，以及当前的方案和 Pipeline 调度。

1. 端到端的联邦学习 Pipeline

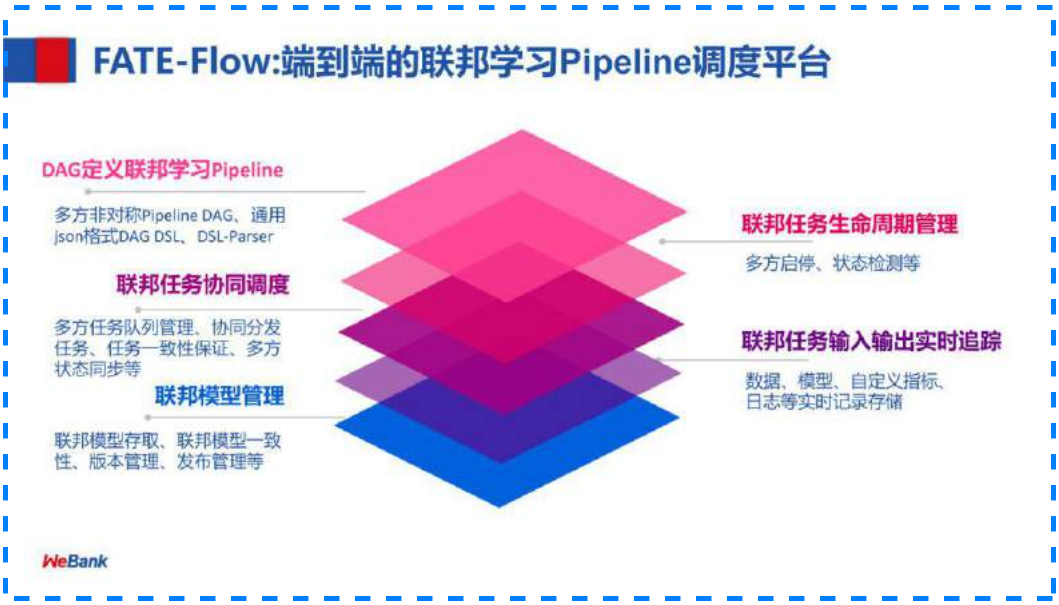


这是一个比较典型的纵向联邦学习 Pipeline 的例子：

交集（intersect）-> 联邦统计 -> 联邦特征处理 -> Training -> 验证 -> 模型保存 -> 模型发布（发布到线上服务 FATE-serving，后面会介绍）

2. FATE-Flow：端到端的联邦学习 Pipeline 调度平台

① FATE-Flow 功能



FATE-Flow 是我们自研的联邦学习调度平台，主要有 5 项功能：

- 用 DAG 定义联邦学习 Pipeline。DAG 具有比较灵活、弹性的特点，是业界工作流调度系统常用的方式；在联邦学习场景下，DAG 是有难点的，由于协作的机制，我们的 Pipeline 是非对称的；我们在设计 DSL 时，考

虑更多系统化对接的情况，所以我们采用了 json 格式的 DAG DSL，然后再采用 DSL-Parser 进行解析。

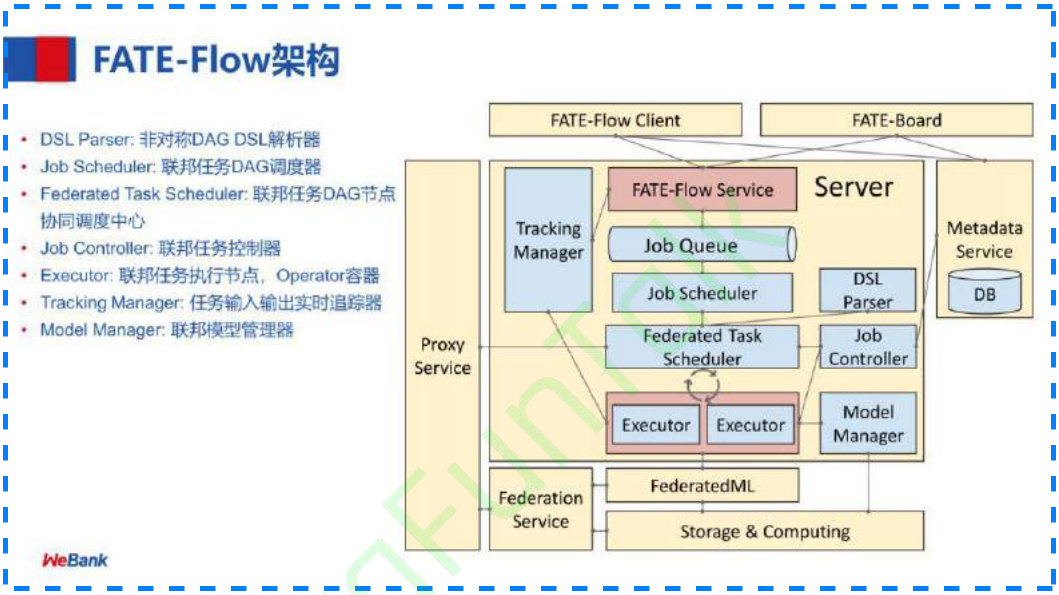
- 联邦任务生命周期管理。对应刚刚所说的痛点，对联邦任务生命周期进行管理，如多方启停、状态检测。

- 联邦任务协同调度。包括：多方任务队列、分发任务、状态同步等协同调度。

- 联邦任务输入输出实时追踪。这个功能比较普遍，一般的调度平台都会有，包括：数据、模型、自定义指标、日志等的实时记录存储。

- 联邦模型管理。联邦学习模型的管理存在一个很大的问题，尤其在纵向联邦学习场景下，就是如何保证多方模型的一致性。

② FATE-Flow 架构



FATE-Flow 架构：

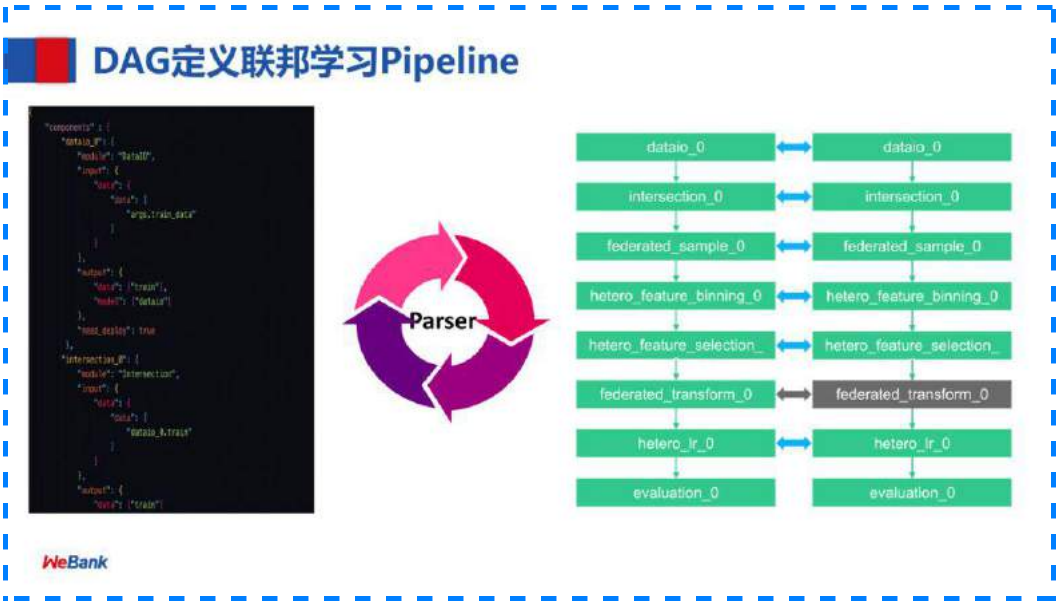
- DSL Parser：是调度的核心，通过 DSL parser 可以拿到上下游关系、依赖等。
- Job Scheduler：是 DAG 层面的调度，把 DAG 作为一个 Job，把 DAG 里面的节点 run 起来，就称为一个 task。
- Federated Task Scheduler：最小调度粒度就是 task，需要调度多方运行同一个组件但参数算法不同的 task，结束后，继续调度下一个组件，这里就会涉及到协同调度的问题。
- Job Controller：联邦任务控制器

- Executor：联邦任务执行节点，支持不同的 Operator 容器，现在支持 Python 和 Script 的 Operator。Executor，在我们目前的应用中拉起 FederatedML 定义的一些组件，如 data io 数据输入输出，特征选择等模块，每次调起一个组件去 run，然后，这些组件会调用基础架构的 API，如 Storage 和 Federation Service（API 的抽象），再经过 Proxy 就可以和对端的 FATE-Flow 进行协同调度。

- Tracking Manager：任务输入输出的实时追踪，包括每个 task 输出的 data 和 model。

- Model Manager：联邦模型管理器

3. DAG 定义联邦学习 Pipeline



左边为我们的 DSL，它的结构比较简单，我们可以定义一串 Component，通过 Parser 解析出 DAG 图（如右图，可以清晰地看到整个算法流程的架构）。

构建 DSL 只需要三步：

① Module :模型组件 ,FATE 当前支持 11 个模型组件 ,基本满足当前 FATE 所支持的所有算法。

② Input：

data：数据输入

model：模型输入

isometric_model：异构模型，当前只用于 Feature Selection

③ Output：

data：输出输出

model：模型输出

可参考下面的例子：

```
"hetero_lr_0": {
    "module": "HeteroLR",
    "input": {
        "data": {
            "train_data": ["hetero_feature_selection_0.train"]
        }
    },
    "output": {
        "data": ["train"],
        "model": ["hetero_lr"]
    }
},
```

构建 DSL 示例

DSL 怎么工作的呢？它是一个非常酷的模块，就像人类的大脑，它是 FATE-Flow 的中心。

组件初始化：

① 根据 DSL 定义和任务配置，解析每个 Component 运行参数

② 分析 DSL 定义 data、model 输入输出，提取依赖关系

DAG 图：

① 构建依赖关系邻接表

② 拓扑排序进行 DAG 依赖检测，因为用户定义的 DSL 不一定是有效的

DataFunTalk

调度协作：

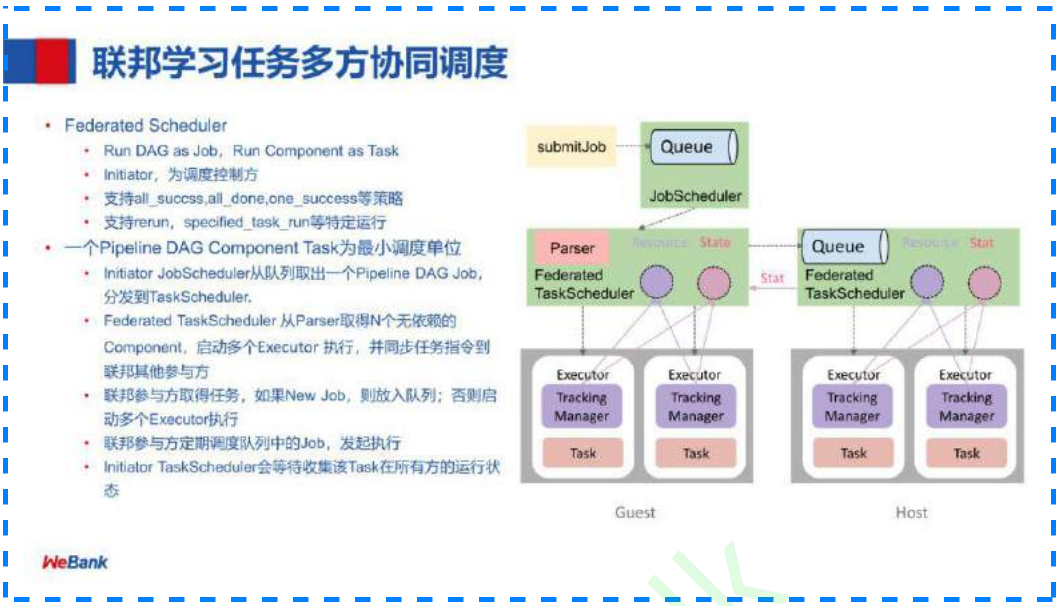
① 实时输出 Component 无依赖上下游

② Component 依赖度自动递减

预测 DSL 推导：

剔除预测阶段无用 Component 数据，模型依赖传递，推导预测 DSL

4. 联邦学习任务多方协同调度



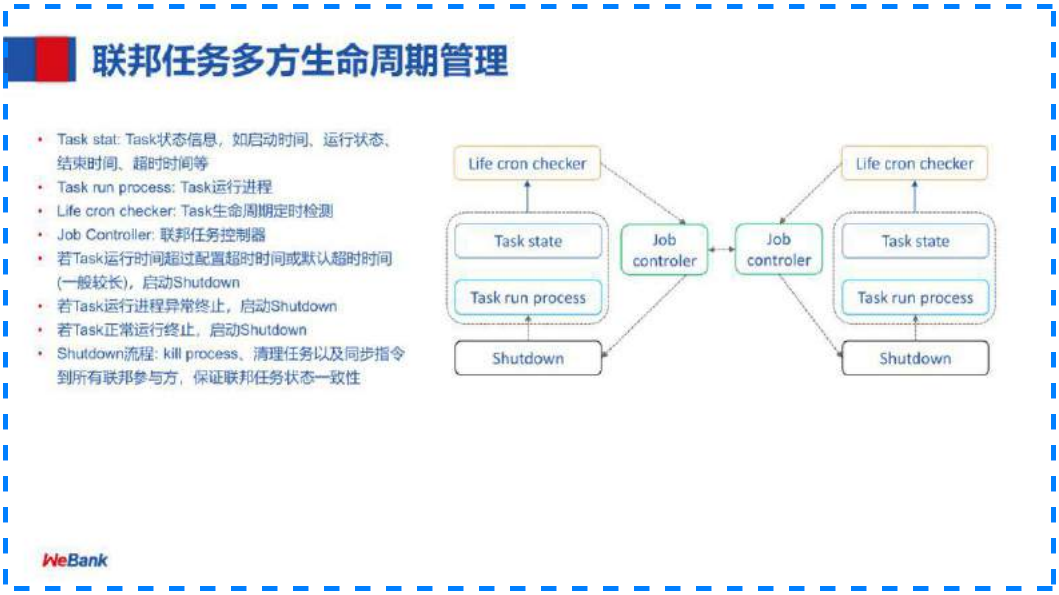
联邦学习任务多方协同调度的流程：

DataFunTalk

首先，是以任务提交的一种方式，提交任务到 Queue，然后 JobScheduler 会把这个任务拿出来给到 Federated TaskScheduler 调度，Federated TaskScheduler 通过 Parser 取得下游 N 个无依赖的 Component，再调度 Executor（由两部分组成：Tracking Manager 和 Task）执行，同时这个任务会分发到联邦学习的各个参与方 Host。联邦参与方取得任务，如果是 New Job，则放入队列（参与方会定期调度队列中的 Job），否则启动多个 Executor 执行，Executor 在 run 的过程中，会利用 Federation API 进行联邦学习中的参数交互，对一个联邦学习任务，每一方的 Job id 是保持一致的，每跑一个 Component，它的 Task id 也是一致的。每个 Task 跑完 Initiator TaskScheduler 会收集各方的状态，进行下一步的调度。对于下一步的调度策略我们支持：all_succss，all_done，one_succss 等策略。由于我们基于 Task 为最小的调度单位，所以很容易实现 rerun，specified_task_run 等特定运行。

5. 联邦任务多方生命周期管理

DataFunTalk



分以下几个部分：

- Task stat：Task 状态信息，如启动时间、运行状态、结束时间、超时时间等
- Task run process：Task 运行进程
- Life cron checker：Task 生命周期定时检测
- Job controller：联邦任务控制器
- Shutdown：kill process、清理任务以及同步指令到所有联邦参与方，保证联邦任务状态一致性

启动 Shutdown 的条件：

- 若 Task 运行时间超过配置超时时间或默认超时时间(一般较长)，启动 Shutdown

- 若 Task 运行进程异常终止，启动 Shutdown

- 若 Task 正常运行终止，启动 Shutdown

6. 联邦任务输入输出实时追踪

联邦任务输入输出实时追踪，首先会有几个 Definition 定义：

- metric type：指标类型，如 auc，loss，ks 等等

- metric namespace：自定义指标命名空间，如 train，predict

- metric name：自定义指标名称，如 auc0，hetero_lr_auc0
- metric data：key-value 形式的指标数据
- metric meta: key-value 形式的指标元信息，支持灵活画图

目前的 API 只有 4~5 个：

log_metric_data(metric_namespace, metric_name, metrics)

set_metric_meta(metric_namespace, metric_name, metric_meta)

get_metric_data(metric_namespace, metric_name)

get_metric_meta(metric_namespace, metric_name)

可能以前收集指标需要经过收集日志等一系列操作，任务像一座座大山一样摆在面前，现在则有可能成为我们的摇钱树，因为我们可以快速的收集各种指标，提交给需求方。

7. 联邦模型管理

左图中的两桶“大饼”，分别代表了某一方的模型，每一个“大饼”则代表了每个组件的 model，如：Dataio、FeatureBinning、FeatureSelection、FeatureTransform、HeteroLR、Pipeline。这里需要做个 Model Binding 模型的绑定，FATE-Flow 的做法还是比较简单的，我们会给每套模型赋予一对标志符 model_id 和 model_version 来唯一标识模型，model_id 由用户自定义的 role 和 party_id 及 model_key 拼接而成，model_version 也是可以自定义的，如果不自定义的话，会默认为 job_id。我们会有一个命名为 Pipeline 的模型存储 Pipeline 建模 DSL 及在线推理 DSL。

下面是某个算法模型数据结构的示例：

```
syntax = "proto3";

package com.webank.ai.fate.common.mlmodel.buffer;
option java_outer_classname = "FeatureSelectionParamProto";


message FeatureValue {
    map<string, double> feature_values = 1;
}

message LeftCols {
    repeated string original_cols = 1;
    map<string, bool> left_cols = 2;
}

message FeatureSelectionFilterParam{
    map<string, double> feature_values = 1;
    map<string, FeatureValue> host_feature_values = 2;
    LeftCols left_cols = 3;
    map<string, LeftCols> host_left_cols = 4;
    string filter_name = 5;
}

message FeatureSelectionParam{
    repeated FeatureSelectionFilterParam results = 1;
    LeftCols final_left_cols = 2;
}
```

示例 1

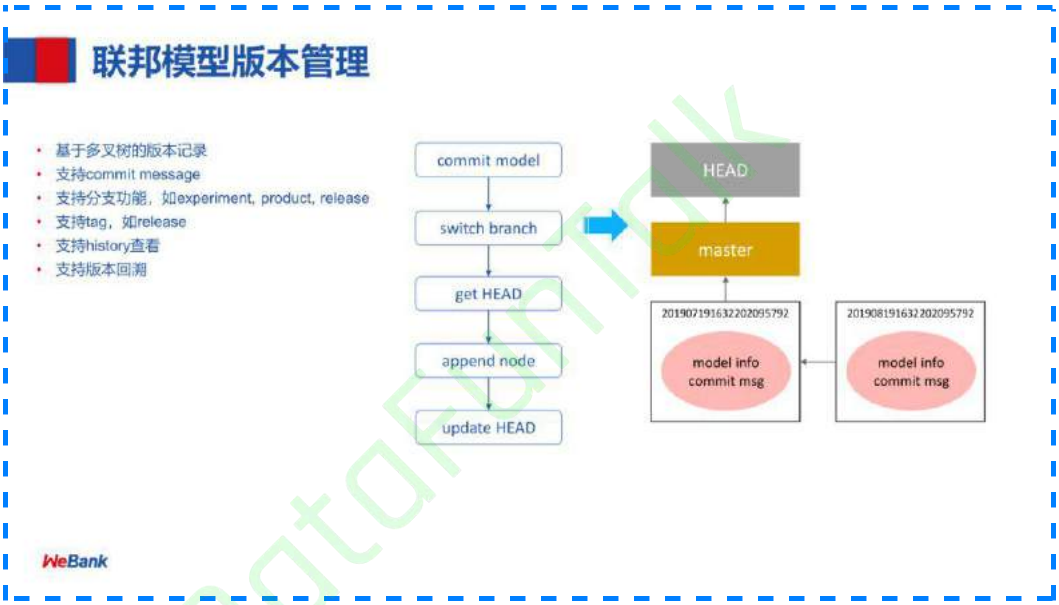
```
syntax = "proto3";

package com.webank.ai.fate.common.mlmodel.buffer;
option java_outer_classname = "FeatureSelectionMetaProto";

message FeatureSelectionMeta {
    repeated string filter_methods = 1;
    bool local_only = 2;
    repeated string cols = 3;
    UniqueValueMeta unique_meta = 4;
    IVValueSelectionMeta iv_value_meta = 5;
    IVPercentileSelectionMeta iv_percentile_meta = 6;
    VarianceOfCoeSelectionMeta variance_coe_meta = 7;
    OutlierColsSelectionMeta outlier_meta = 8;
    bool need_run = 9;
}

message UniqueValueMeta {
    double eps = 1;
}

message IVValueSelectionMeta{
    double value_threshold = 1;
}

message IVPercentileSelectionMeta{
    double percentile_threshold = 1;
}

message VarianceOfCoeSelectionMeta {
    double value_threshold = 1;
}

message OutlierColsSelectionMeta {
    double percentile = 1;
    double upper_threshold = 2;
}
```

DataFunTalk

示例 2

同时每个“大饼”算法模型 ,也由两部分组成 :ModelParam 和 ModelMeta，也就是参数和元的信息。

8. 联邦模型版本管理



模型版本管理我们参考了 Git 的实现思路，但是我们没有做的那么复杂，是基于多叉树版本的记录：

- 支持 commit message；
- 支持分支功能，如 experiment，product，release；
- 支持 tag，如 release；

- 支持 history 查看；

- 支持版本回溯，指定某一版本回滚。

9. 使用样例

上图为，FATE-Flow 的简单使用样例，主要就是使用 FATE-Flow CLI 提交一个 Job，需要提供 Job 的 DSL 描述以及配置文件，那么 FATE-Flow Server 会返回该 Job 的一些必要信息，尤其唯一 Job Id 比较重要。后面

则是查询 Job 状态以及停止 Job 的操作指令，CLI 还支持许多丰富的指令，可以参考 github 上的文档。

第三部分介绍下联邦学习建模可视化：

1. FATE-Board

大体的架构如右图，有一些 Job DashBoard 和可视化，两个基本的管理和上面的 Web UI。

FATE-Board 作为 FATE 联邦建模的可视化工具，旨在跟踪和记录联邦建模全过程的信息，并通过可视化的方式呈现模型训练过程的变化以及模型训练结果，帮助用户简单而高效地深入探索模型与理解模型。

2. 建模交互及可视化

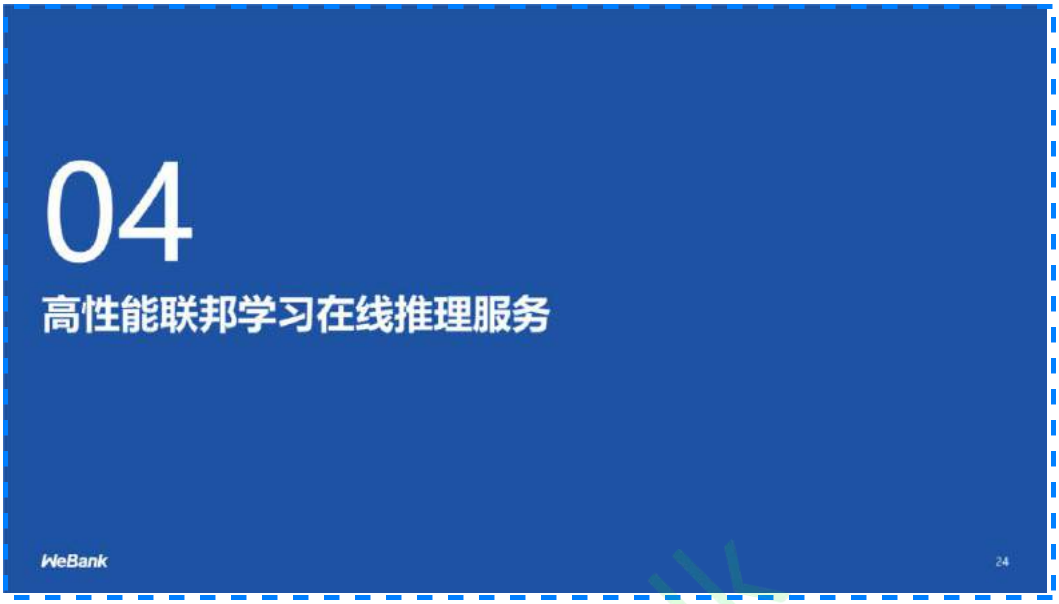
其基本步骤为：

① 用户配置 pipeline，建立 graph、定义 parameters 等；

② 用户提交 job，返回 job URL，同时启动 job 运行，进入 web 端查看 fateboard；

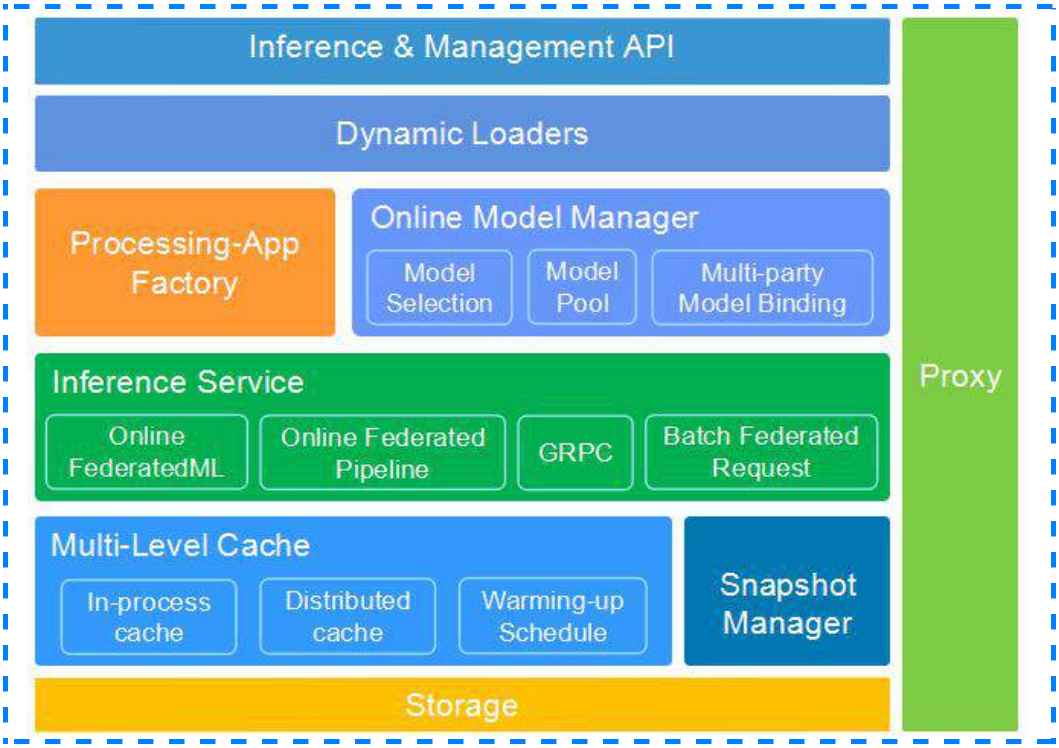
③ 观察 job 运行状态，查看运行时的统计信息，包括运行进度、日志、模型过程输出等；

④ 查看 job 运行完成的结果，包括模型输出、模型评分、日志等内容及可视化结果。



第四部主要讲怎么发挥模型的最大价值，我们构建了高性能的联邦学习在线推理服务：

1. FATE-Serving



FATE-Serving 设计原则：

- 高性能，基于 GRPC 协议，批量联邦请求，联邦参与方模型结果多级缓存
- 高可用，无状态设计，异常降级功能
- 高弹性，模型 & 数据处理 App 动态加载

FATE-Serving 模块：

- Online FederatedML：高性能在线联邦学习算法包，在线和离线时性能是不一样的，在线的时候，我们没有采用 Python，当前版本是采用 Java 写的一套联邦学习算法。

DataFunTalk

- Online Federated Pipeline：在线联邦推理 Pipeline，比较简洁高效，后面会详细介绍。

- Dynamic Loaders：推理节点动态加载器。目前支持模型和 APP 两种推理节点 ,节点通过训练或者编译等方式创建后是序列化后存储在分布式存储上的。对于在线服务，每个请求都需要经过每个节点，为了保证执行效率，需要把这些节点提前从存储中取出缓存在 Server 的内存空间，这也是业界在线服务的常用做法。

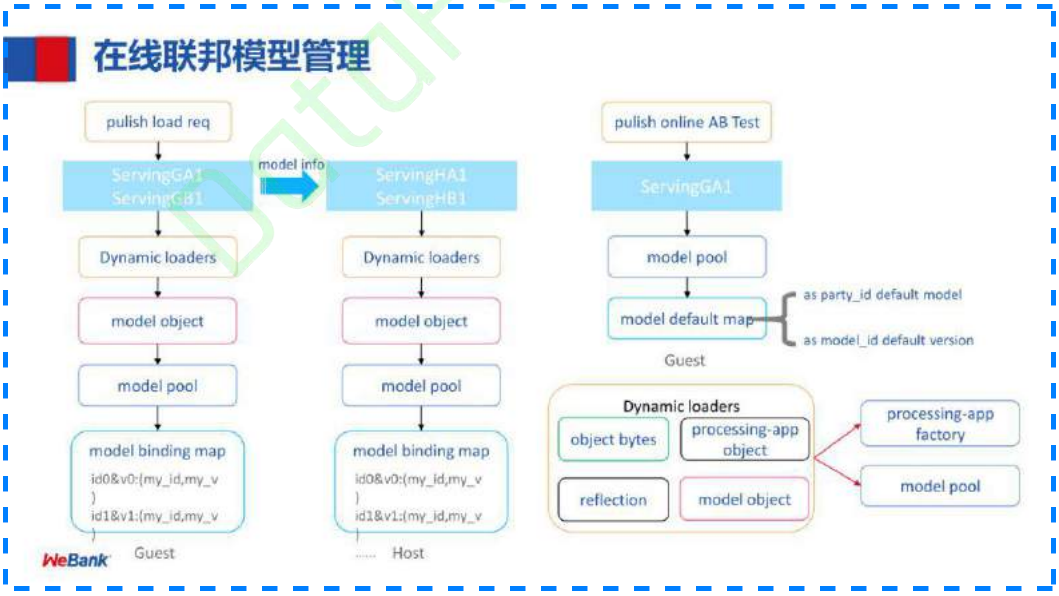
- Model Manager：在线模型管理器

- Processing-App Factory：数据处理节点工厂。在解决实际业务问题时，还需在模型预测的基础上在 Pipeline 中引入一些人工或者规则 ,打包成一个 App 发布到 Serving，在预测的时候加入一个前处理 Processing 和一个后处理 Processing。

- Multi-Level Cache：多级缓存管理器

- Snapshot Manager：快照管理，定期将在线模型、Processing-App 信息落库。

2. 在线联邦模型管理

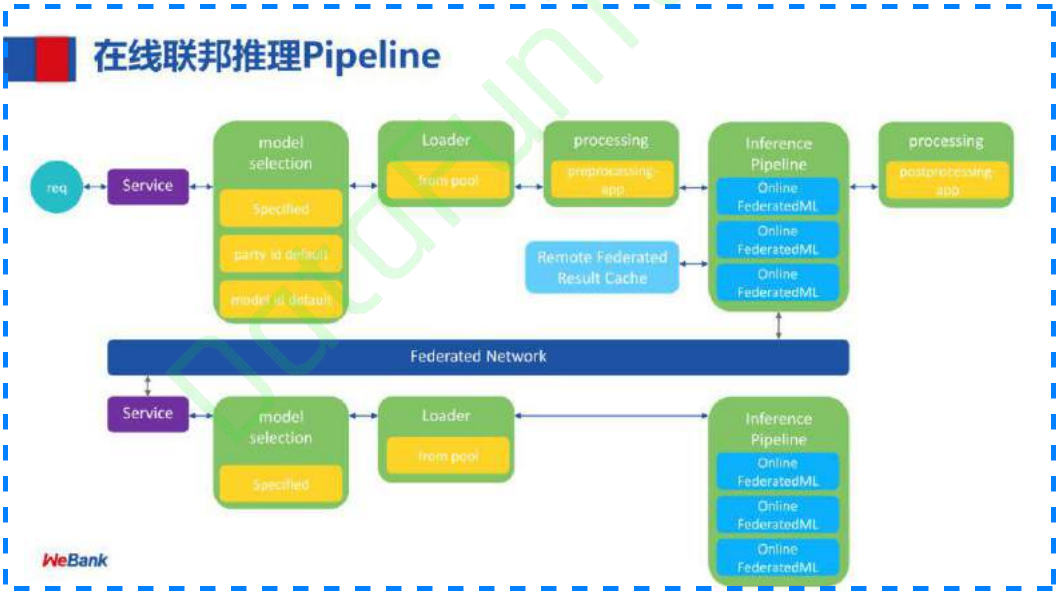


在线联邦模型管理 ,我们提供了两个 API :pulish load req 和 pulish online AB Test。为什么是两个 API？因为，Pulish load req 过程只是把整个模型通过 dynamic loaders 得到一个 model object 放入 model pool 中，是一个内存的池子。联邦各方的 model id 和 model version 都是保持一致的(在发布模型的时候 ,只需要在 Guest 侧发起就可以了 ,会自动走 proxy，把命令推到其他的 Host 中去），当各方都加载完之后，就可以应用了。

Pulish online AB Test 主要是做 default binding，比如有些业务方并不知道我们的模型版本，需要有一个默认的机制。

图中右下角，为动态加载器的一个基本功能。

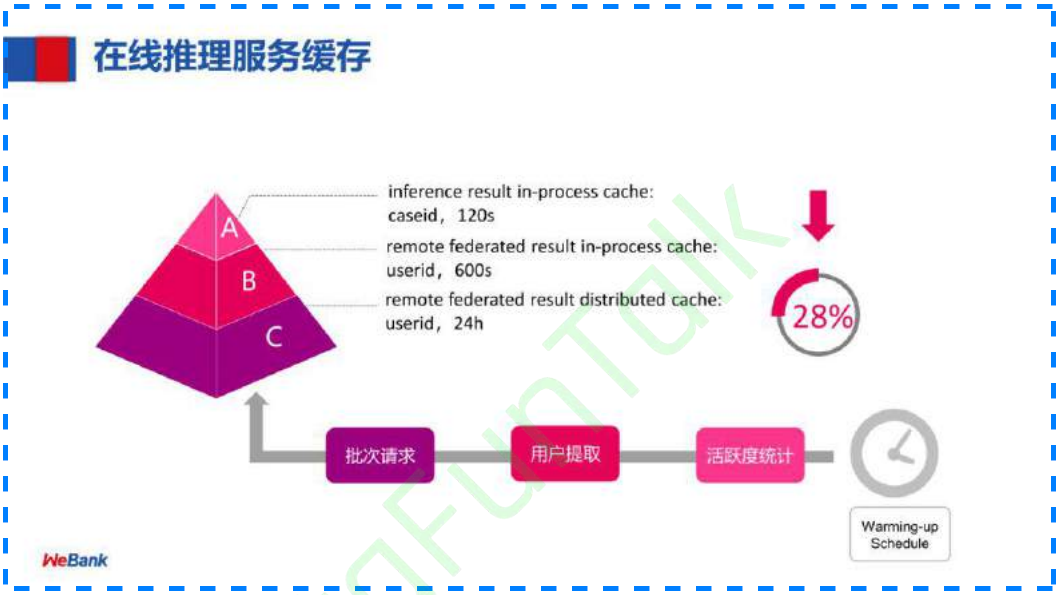
3. 在线联邦推理 Pipeline



在线联邦推理 Pipeline 流程：

Req -> Service -> model selection（ 有三种不同策略：规定的方式和两种默认的方式 ）-> loader -> processing（ 前处理 ）-> inference Pipeline（ 是纵向 model，会经过不同的组件，这里举例是三个，然后会把 user_id（用来做对齐的）发到不同的参与方，参与方会拿自己的半模型，进行最后的预测，最后把结果发到 Federated Network ）-> Processing（ 后处理 ）

4. 在线推理服务缓存



在线推理服务缓存分为三种：

Inference result in-process cache：cache key 为 caseid，缓存 120s。

Remote federated result in-process cache：cache key 为 userid，缓存 600s。

Remote federated result distributed cache：cache key 为 userid，缓存 24h，跟 A 方的特征更新周期有很大的关系。

在业务的运行当中，按照这几个配比，一天下来发现命中远端联邦推理结果 Cache 的请求量占了 28% ,也就是说有 28%的联邦请求都不需要这部分，就会大大的增加了用户的体验。

我们还有一个缓存唤醒的热身过程，我们会跑一些离线任务，然后对活跃度进行统计，把要缓存的用户按比例提取出来，在凌晨的时候发一些批次请求，让整个缓存热身。

5. 联邦模型应用生产服务流程

一个联邦模型应用到生产服务的大致流程：

① 全量加载联邦模型，通过 pulish load req 全量加载模型，每方的 Serving 都加载这个模型。

② 灰度上线联邦模型，pulish online AB test 可以指定多少用户上线 online 模型。(这时就会体现出为什么要做 pulish load req ,因为在 Guest 方灰度上线时，这个 Serving 请求的 user_id 落到其他参与方，需要找到

对应的模型，但是通常情况下 Serving 都是做负载均衡的，如果灰度这部分的 Serving 已经上线了新的模型，其它 Host 方没有预先加载这个模型，就会找不到这个模型，也就是说模型是被所有 Serving 所 load 的，但是并不是所有 Serving 当前生效的都是这个模型。）

③ 联邦模型效果验证

④ 全量上线联邦模型

最后分享下我们联邦学习的官网：

https://www.fedai.org.cn/cn/

以及 FATE 的 Github 地址：

https://github.com/FederatedAI/FATE

Hadoop or TDengine，如何做物联网大数据平台的选型？

嘉宾：关胜亮　编辑：Hoh Xil

导读：本次分享的主题为 **Hadoop or TDengine，如何做物联网大数据平台的选型？**主要介绍物联网大数据处理中可能遇到的问题；结合实际的应用场景，分析 **TDengine、InfluxDB、ClickHouse、Hadoop、MySQL** 等系统在处理时序数据时的优缺点。

——前言——

1. 大数据时代

大数据时代，大家都在说什么叫大数据，强调的就是一个“大”字，人们期望对海量数据的挖掘和运用能够获取到更多有价值的东西。其来源包括：微

信聊天数据，淘宝&京东等电商数据，高速收费站的数据，摩拜等共享单车产生的数据，股票交易数据，天文望远镜产生的数据等等，这些数据有的是物联网的数据，有的是时序数据，有的不是时序数据，比如微信和淘宝、京东等产生的数据就不在本次讨论范围内。

物联网的数据可以分为两类，静态数据和动态数据：

静态数据：通常为标签类数据，如：设备所在的地址、编号、颜色等等，这些数据不会随着设备产生新数据的过程中而发生变化。

动态数据：以时间序列为基础的数据，其特点是和时间有一一对应的关系，可以按照时间顺序记录的数据列，并且这种关系在数据处理中尤为重要。充分利用这样的关系，才能把数据库做好。现在越来越多的公司意识到处理这样的时序数据应该用专门的数据库，当然，也还有相当一部分的公司仍然用 Hadoop 来处理，在此基础上做分析和改造，这样成本就会相对较高。

物联网大数据主要体现在采集的机器数目越来越多，频次越来越高，如：电网以前可能只采集 500KV 的数据，慢慢可能采集 10KV 的数据，最后入户的 220V 的数据可能都要采集。随着这种 2 维方式的增长，采集的数据成几何量的增加，因此，传统的方法带来越来越多的挑战。

2. 物联网、工业 4.0 的技术链

对物联网、工业 4.0 的数据进行分析，会经过如下 4 个步骤：

数据采集：通常通过传感器采集。

边缘计算：通过通讯模组传输数据。

存储•查询•计算：将传输的数据输入云数据引擎进行存储。

系统：最后将处理好的数据接入大屏、app、web 界面应用等系统。

物联网的数据平台主要集中在边缘计算和云数据引擎：

边缘计算。有些场景需要用到边缘计算，有的则不需要。边缘计算（或前置机模块），主要负责采集工业上的协议（因为，传感器采集的数据，通常是不规整的），同时可以在边缘计算上做降采样，把一些频率高的数据放在边缘计算部分。

云数据引擎。这块是本次分享的重点，后面会详细介绍。

1. 传统的实时数据库

在物联网概念出现之前，设备数据的采集和分析也是非常重要的功能需求。为了解决流程控制领域的实时数据处理问题，从上世纪八十年代起，出现了一批实时数据库，以美国的 OSISoft PI 为代表，具有较高的数据处理能力，能够很好的解决传统工业生产问题。当时和实时数据库相结合的组态软件流行了很长一段时间，主要强调工控领域的实时数据采集、监视或者控制，其实现思路并不复杂，本质上是内存库，卖点在于实时响应和实时控制。这里实时控制是非常重要的，因为工控就是在强调控制，通过过来的参数，如温度高的、电流大的，能够把这些信息反馈回去，重点在于快速，能够在 1s 之内响应。也因为这个特点，这种实时库只能存储当前所有采集量的截面，相当于一个时间片，有些好的产品可以存储多个截面，但也比较有限，可能只能存储 5 分钟内的数据。

2. 传统的实时数据库面临的挑战

这样的数据处理方式是物联网吗？答案是肯定的，因为物联网的基本概念是物体和物体相联。但这只是物联网的初级阶段，解决数据采集从无到有，包括数据的实时上数和告警的问题，随着测点数暴涨，数据采集频次不断提高，传统实时数据库暴露出下列问题：

水平扩展：没有水平扩展的能力，数据量增加，只能依靠硬件 scale up（增加单个硬件性能，如把 16G 内存的机器换成 256G 内存的，但是这样的扩展能力还是有限的）。所以水平扩展，在选型过程中如果解决，也是一个重要的问题。

技术架构：技术架构陈旧，使用磁盘阵列（价格比较昂贵），还运行在 Windows 环境下。

数据分析：数据分析能力偏弱（还需要历史数据，否则只能根据当前数据进行分析，看看有没有超过阈值，或发出报警，这其实不能算作数据分析），不支持现在流行的各种大数据分析接口。

云服务：不支持云端部署，更无法支持 PaaS。

现在做物联网，在实时数据的基础上，还要进行扩充，做历史数据。通过历史的查询，归类的统计，或者机器学习的方法把历史数据增值，获得两类信息：

1. 整体情况的统计，如有 10W 个设备，统计这 10W 个设备的总体情况，也就是常说的报表。

2. 单个设备的分析，如有一台设备经常坏，为什么经常坏呢？我们可以看下这台设备采集的数据是什么样的，是不是电流经常发生突变等信息。通过对单个设备进行分析，把这块的数据模式收集起来，扩展到其它设备中，能够提高整个系统运维的安全性。同时，把单个设备进一步扩展，扩展到每个人用的手机或者手环等，能把这些大数据做好，那么我们除了 To B 业务外，也能在 To C 业务上给公司产生增值。

如果不需要历史数据，可以只做 redis，因为只看当前数据，redis 会比一些实时数据库的架构稳定性更好。

3. 通用的互联网大数据解决方案

目前的 Hadoop 方案，是一些大型的互联网企业最先使用的。在处理大数据时 ,将多个开源软件 ,如现在比较流行的 kafka ,然后把实时数据引入到 redis，把历史数据存到 Hadoop，中间可能结合 Spark 和 Flink 的计算，利用集群来处理海量数据。这是一种非常好的，通用的处理大数据的解决方案，可以处理百亿、千亿、甚至万亿级别的数据，只需保证它的服务器足够。如果有特殊需求，可以写一些代码来实现，比如做实时监控，就可以在 kafka 后面挂一个 Flink 做实时分析，做实时的流计算，把当前的 QBS、健康状态做实时统计；在比如看历史数据，可以在 Hadoop 上挂一个 MapReduce，这样我们可以通过写程序把所有的需求都实现。

那么对于物联网，为什么不用这一套解决方案来做呢？是不是不需要做物联网大数据平台的选型呢？答案显然是否定的。

4. 通用互联网方案面临的挑战

对于一些规模较小的公司，做软件开发所关注的点，Hadoop 系统并没有很好的解决，因为它比较低效，也比较复杂，成本比较高。逐一分析下：

1. 开发效率低：因牵涉到多种系统，每种系统有自己的开发语言和工具，开发精力花在了系统联调上，而且数据的一致性难以保证。如果你的公司从头开始做这件事，投入成本更大。

2. 开发成本高：这套架构需要的开发人员，市场比较稀缺，可以把这套系统搭建起来的比较有经验的研发经理，年薪没有 50W 是找不到的，一般的开发人员也比较难找，如果从新培养，但是待遇低的话，也很难留住人才。

3. 运维复杂：每个系统都有自己的运维后台，带来更高的运维代价，出问题后难以跟踪解决，系统的不稳定性大幅上升。

4. 运行效率差：非结构化数据技术来处理结构化数据，整体性能不够，系统资源消耗大。因为多套系统，数据需要在各系统之间传输，造成额外的运行代价。

5. 应用推向市场慢：集成复杂，得不到专业服务，项目实施周期长，导致人力攀升，利润缩水。

综上所述，该如何解决这些问题，如果你的公司做这样的项目能够投入的人比较少，不足 10 人，那么投入的硬件可能就是 10 个服务器，在这样的规模内，用 Hadoop 显然是不合适的，否则 1~2 年也不会做出好的产品。

另外，在国内人力成本越来越高的情况下，很多的公司都期望能够选择一个数据库。不光解决各种效率问题，更重要的是出了问题能够及时有人处理，比如用俄罗斯的 ClickHouse 或者美国的 InfluxDB，如果出现问题，花钱也很难找到人维护，所以国家搞国产化，也有这样的因素在里面。

5. 物联网、工业 4.0 数据特征：时序空间数据

那么对于物联网大数据平台，什么样的选型方案才是专业的、正确的？我们首先要对网联网数据的模式进行分析，总结物联网数据的特征，并加以充分利用，这样我们选择的数据库才是一个真正能够利用软硬件资源，发挥最大效用的网联网数据库。大家在做自己的 app 时也要考虑你的数据是否满足这样的特征，是否适合用时序数据库来做。其典型特征有：

1. 所有采集的数据都是时序的。在数据索引的基础上，把内存块、磁盘文件按照这样的时序的假设进行拆分，就不需要再做额外的索引，来节约时间，提升效率。

2. 数据都是结构化的。计算机的编程语言更擅长处理结构化的数据，比如一个 Json 格式的数据过来，虽然有很多的库直接就把它解析掉了，形成我们需要的数据，但是这个过程也是比较耗时的，因为传入的时候需要进行正向的序列化，传出的时候需要进行反向的序列化，这俩次的序列化会产生很大的 CPU 消耗。并且结构化的数据，可以采用列式存储，虽然物联网的数据量比较大，但是变化通常都比较有规律，如果采用行式存储，压缩比率不会太高，如果采用列式存储，如 001100 这种数据，我们只需要记住 0 出现几次，1 出现几次，而不需要把所有的数据都记录下来。因此，不是列式存储的数据，建议大家都不要选用。

3. 一个采集点的数据源是唯一。因为时序数据最重要的特点就是每台机器产生的数据跟其它机器产生的数据在逻辑上一定是分离的，因此，可以按照分

离的操作进行分表，Prometheus（普罗米修斯）和涛思数据库都是这样做的，逻辑上进行分离之后，就不需要把大量的数据混合在一起进行查询，大大提高了查询效率。

4. 数据很少有更新或删除操作。可以在数据写入磁盘的时候进行优化，当数据量增大的时候（几十上百 M/s 的写入速度），能够达到单块磁盘极限的量，这时，一定要减小落盘后的修改，否则在修改的时候，某些数据是没有办法写入磁盘的，这时系统整体的吞吐量是不够的。

5. 数据一般是按到期日期来删除的。在选型初期，很多公司不太注意压缩比，运行一段时间之后，数据越来越大，就不得不考虑数据的保存时长，如何快速的删除过期数据，这是需要考虑的问题。

6. 数据以写操作为主，读操作为辅。在设计过程中要有一个优先级，需要优先满足写的操作。

7. 数据流量平稳，可以较为准确的计算。有些互联网公司可能双十一的时候数据流量非常大，平时数据量相对较小，那么就不太容易估算机器的容量。

8. 数据都有统计、聚合等实时计算操作。这些操作可以在数据库中进行预先计算，可以在数据库内部做，也可以在接收的时候将数据库的结果保存在一些关键步骤，这也是之前 Hadoop 等数据库通用的方法。

9. 数据一定是指定时间段和指定区域查找的。可以把查询遍历的磁盘缩小到非常小的一部分，这是提升数据库速度的一个关键。

10. 数据量巨大，一天的数据量就超过 100 亿条。数据库能不能支撑分离存储，新的、热的数据放在 SSD 中，老的数据放在 HD 上，或者其它更老的数据放在网络存储中。

这就是物联网大数据平台在选型中需要注意的问题。

——实际应用场景分析——

接下来通过分析一些系统的现状，来看看出现了哪些问题：

1. 某智能园区配电监控系统

- 强调实时数据处理，包括告警、失电设备分析、故障快速恢复方案，大屏展示
- 历史数据分析较少
- 难以升级改造，提高数据采集频率和性能需要推翻重做
- TDengine完美解决历史数据和实时数据的问题

历史数据库采用行转列，按天分表，5分钟一条历史护具，，每表288字段

- Col1 id
- Col2 collectName
- Col3 0分钟数据值
- Col4 5分钟数据值 .....

这是某智能园区配电监控系统，采用的还是 10 年前的架构，实时数据库是这个架构的最核心部分，数据从采集设备过来之后，经过数据采集服务器（前置机）进入到实时数据库和历史数据库中（实时数据库以内存库为基础，随着技术的进步，也进行了扩展，开发了历史数据库，并且把数据分为三层，分别为热层、稳层和冷层。热层是指数据在内存中，稳层是指数据在硬盘中，冷层是指对硬盘中的文件进行了压缩或者二次归档。这种冷温热层的区分，已经和现在的时序数据库系统比较接近了，但是使用上并不是很方便。），采集的数据通过采集服务器进入到了内存库进行了实时计算，同时历史数据采用 Oracle 进行存储，并且这种方式也采用了行转列的方式，在时间维度上对数据进行拆分，按天分表，5 分钟一条历史数据，每表 288 字段。

这个系统的卖点在于实时数据处理，包括告警、失电设备分析、故障快速恢复方案，可以通过大屏展示一些数据指标。

其问题在于：

1. 历史数据分析较少

2. 难以升级改造，提高数据采集频率和性能需要推翻重做

这时可以采用 InfluxDB 或者 TDengine 时序数据库替换实时数据库和历史数据库，不过 InfluxDB 达到同样的效果，可能会需要比较高的硬件成本，这也是需要考虑的因素。

2. 某车联网数据仓库

第二个是某车联网数据仓库，采用的是 Hadoop 系统的解决方案：

- 记录了百万级别车的历史数据，分钟级别的采集频率
- 每天数据增量在 2TB 左右，需要保存 10 年
- 作为数据仓库，无实时查询要求
- 历史查询仅仅在发生事故时使用

存在的问题：

- 采用百台级别的硬件服务器，硬件扩容成本和软件维护成本很大
- 因为存储在 Hadoop 系统中的数据采用的是 lzo 的压缩格式，综合压缩比只能达到 45%

- 有强烈的改造需求，希望提供一些增值服务，比如每台车的车主想要查看自己车辆的信息，都去过哪些地方，停留了多久等，这时对系统的查询需求就会比较大，但是 Hadoop 是没法做到实时查询的，做到 10s 级别的查询都非常困难。随着记录的车辆越来越多，国家要求更多的车辆的数据要保存的车联网中，那可能就不是百万级别的车，可能 300W、500W 甚至更多，服务器就得相应增加到 300 台或者 500 台，虽然能够处理，但是盈利空间并不是很大，没有动力来支持做这样的扩容。

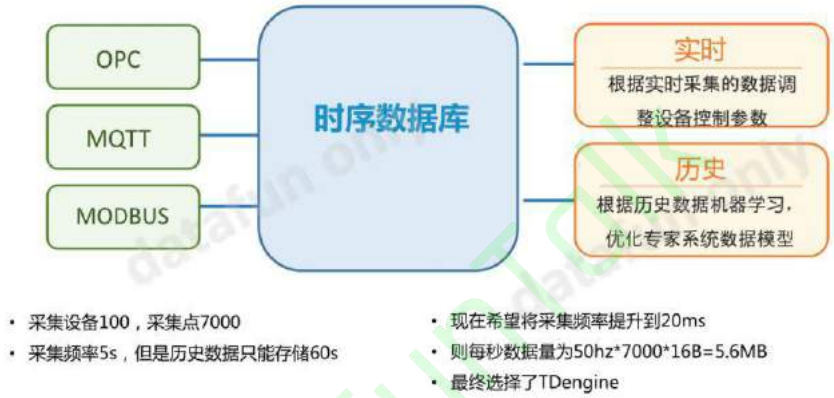
这时，我们就建议更换它的数据库，这里比较适合的有：TDengine、ClickHouse，ClickHouse 适合做一些数据仓库，但是实时写入的能力相对差一些，TDengine 在实时查询上会更好，压缩比可以做到更高，当时测试的时候，压缩比可以做到 4%，比原来提升了 10 倍。

3. 某大型机械制造商的设备管理系统

第三个是大型机械制造商提供给客户的设备管理系统，每套设大概卖到 50 万或者 100 万，已经销售了几十万级别的设备。数据进来之后，首先进入 kafka 中，通过 kafka 将实时的压力分摊出去，实时数据会进入 redis 中，通过统计进入关系库中，然后给到 App 做最新状态的显示，历史输入会进入 Cassendra 中，Cassendra 的特点是写入速度非常快，但是查询非常耗时。这样的系统其实是需要更新的，但是动力并不高，因为它还是能基本满足用户的需求，这时就需要挖掘出用户新的需求，提供增值服务。

从这个例子和上一个例子我们可以看到，如果只是偶尔的查询，一般的数据库都能满足要求。但是物联网不只是物体和物体的联接，还包括用户和物体的交互，不止能够看报表，还要能够走到 C 端的用户中，这时物联网平台才能有影响力，才能产生更多的增值服务。所以当我们的客户不再是公司、管理人员、运维人员，而是真正的用户，那这套系统将非常具有价值。在这种需求下，如何去选型是我们需要考虑，也就是谁的实时数据处理的好，谁的历史数据处理的好，能够满足这样的需求。

4. 某智能工厂的高频数据采集系统



这是某智能工厂的高频数据采集系统：

- 采集设备 100，采集点 7000
- 采集频率 5s，但是历史数据只能存储 60s

目前的诉求：

现在希望将采集频率提升到 20ms，每秒数据量达到 50hz*7000*16B=5.6MB，并且采集规模扩大到所有产品线，以期更好的提高生产效能。

这里可以采用 TDengine 和 Prometheus，它们各有优缺点，大家可以在选型的时候考虑下。

5. 某电动车制造商的车辆实时检测系统

最后一个案例是某电动车制造商的车辆实时检测系统，目标比较火，是面向终端用户的。用户在购买电动车之后可以选择是否购买车辆实时检测系统，实际就是在车辆上安装这样的采集盒，定期的将车辆的轨迹、电池等参数上传到云端，成批次的写入 MySQL 数据库，之后车主就可以随时查询车辆的轨迹。这套系统，在最初是非常 OK 的，但是随着车辆的增加达到 1000 辆时，写入压力就非常大了，CPU 占有率非常高，导致没有办法处理实时查询的请求，最终只能降低入库的频率，但这样大大降低了用户体验。

这时它需要：

- 提高实时的写入能力
- 写入的数据可以快速查询
- 可以水平和动态的扩容

明确了这样的业务需求之后，它做了技术选项，当时竞争的数据库也比较多，最终选择了 TDengine ,说明 TDengine 在这样的业务需求下 ,是非常出色的。

——时序数据库选型问题总结——

最后，总结下时序数据库选型时需要注意的问题，时序数据库的基本功能就是提供高效可靠的数据采集、数据插入、数据查询和计算等通用的功能，判断一款时序数据库是否好用，主要是看能否在你的业务需求上产生亮点，对业务产生增值：

性能提升：提高实时写入（用点/s 的指标进行衡量，10W 点/s 只是起点）、实时查询、聚合计算、历史查询性能，更多更快。

业务提升：带来新的业务需求，离线功能转在线功能。

总拥有成本：

硬件成本：磁盘空间、计算资源、机房费用。2. 运维成本：备份、迁移、DBA 工具、人员成本、运维工具。

研发成本：丰富的 API 接口、低学习成本、低人员成本。

总之，在物联网的选型上，如何找到自己痛点，在单位硬件的条件下，测试这几个痛点的主要性能，才能更好的做物联网大数据平台的选型。

DataFunTalk

实时计算引擎在贝壳的应用与实践

嘉宾：王磊　编辑：Hoh Xil

导读：本次分享的主题为**实时计算引擎在贝壳的应用与实践。**主要内容包括：

- 背景介绍
- 流式计算平台
- 实时分析监控平台-FAST
- 后续规划

——背景介绍——

贝壳找房由链家网升级而来，是以技术驱动的品质居住服务平台，聚合和赋能全行业的优质服务者，打造开放的品质居住服务生态，致力于为两亿家庭提供包括二手房、新房、租赁和装修全方位居住服务。

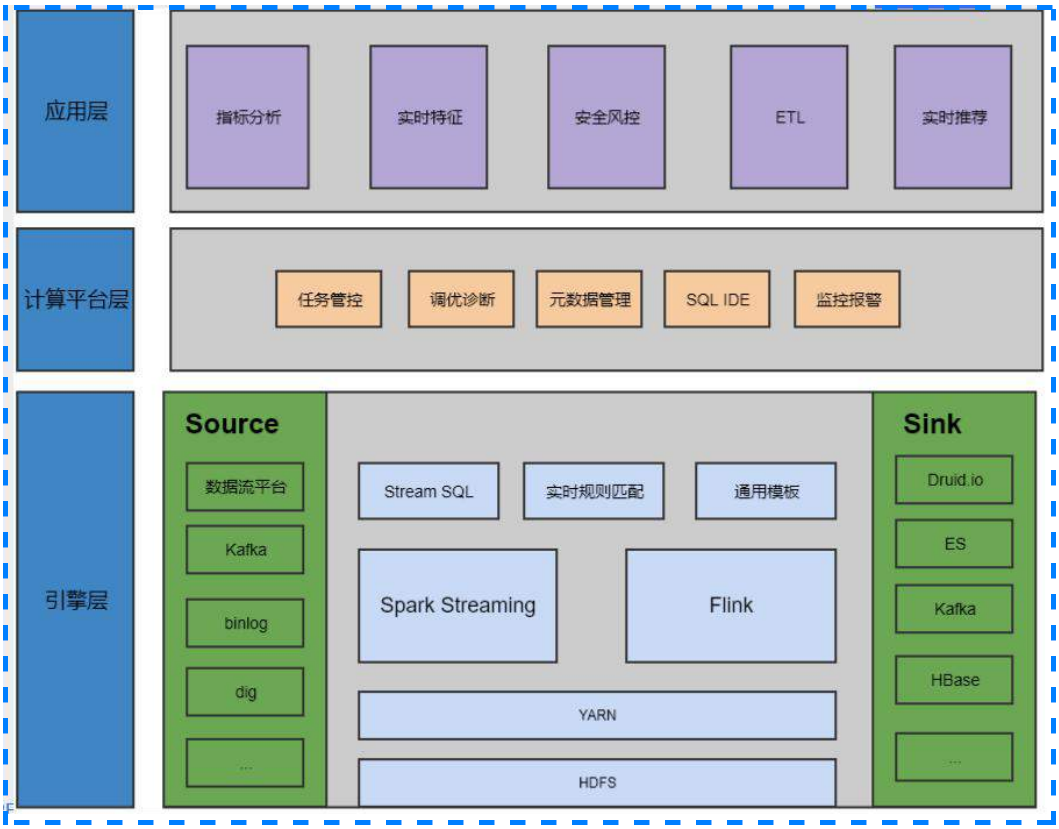
贝壳找房大数据架构团队负责公司大数据存储平台、计算平台、实时数据流平台的架构、性能优化、研发等，提供高效的大数据 OLAP 引擎、以及大数据工具链组件研发，为公司提供稳定、高效、开放的大数据基础组件与基础平台。

DataFunTalk

贝壳目前有 1000 多人的产品技术团队。从实时数据应用角度，公司内主要应用的实时数据，一个是线上的**日志**，大概有两千多个线上的服务，每个服务又输出了很多的日志，日志数据是流式数据应用最多的。第二部分就是**埋点**，在 APP，web 端上报的经纪人作业情况和 C 端用户的行为，这部分通过前端的埋点技术上报。第三部分就是**业务**的数据，业务用 kafka 做消息队列产生的实时数据。

——流式计算平台——

1. 流式计算平台



平台目前主要建设 Spark Streaming，Flink 两种在实时计算中比较常见的计算引擎。平台化的背景就是早期如果公司内有业务想用数据流进行计算，可能需要申请客户端，自己去搭建一个客户端，然后向集群上提交实时作业。这个产生的问题就是如果每个业务方都去自己这样做成本比较高，每个业务都需要关心自己作业的运维问题，还有监控，实时数据作业的监控建设的水平也是参差不齐。Spark Streaming，Flink 对于业务同学直接开发也有一定的学习成本，很难直接用上大数据的能力做实时计算。我们计算平台流数据源主要都是用 Kafka。Kafka 数据中数据分为几类，一个是数据流，数据流指的是线上的业务日志、访问日志等，会收集到数据流平台。Binlog 是线上 MySQL、TIDB 产生的 binlog 作为实时数据源。Dig 是内部前端埋点项目的代号。计算平台底层集群使用 YARN 做资源调度。再上层就是我们现在主要基于社区版 Flink、扩展了开源 Flink 的 SQL 能力。还有提供一些通用的实时处理模板。流计算的输出要覆盖线上业务所有需要的存储分析引擎，例如 ES、Kafka、Hbase 等等。在这些底层基础之上，首先要做的就是实时任务的管理管控。包括如何帮助平台上这些所有 Flink 或者 Spark 任务，进行资源的调优，对实时数据流中的数据的元数据化。另外还有在 Web 端提供的 SQL IDE、任务运行状态的监控报警。在计算平台之上业务方做的一些应用：指标分析，实时特征，安全风控，ETL、实时推荐等等。

2. 目前现状

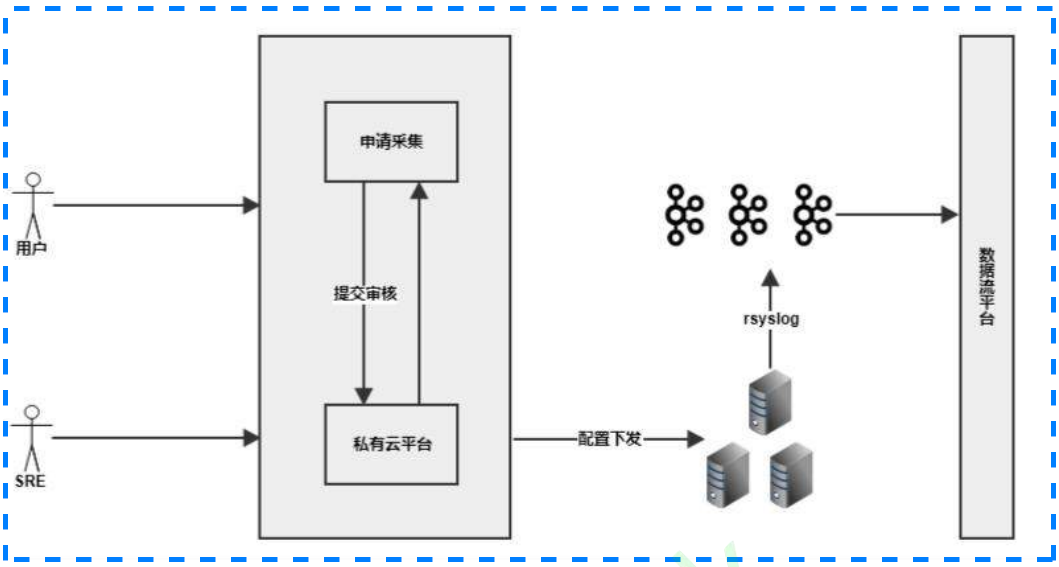
目前 YARN 平台大概有七百多个节点，一千多个实时任务，每天的消息量千亿级，高峰单个任务消息量百万条/s。

3. 为什么要选择 Flink

Flink 提供了 Exactly-Once 一致性语义，并且具有非常完善的多种窗口机制，引入了 Event Time 与 WaterMark，还提供丰富状态的状态访问。Flink SQL 降低了实时计算的使用门槛，并且 Flink 社区非常活跃。

我们希望更多使用 SQL 的方式开发计算任务。SQL 的好处在于我们如果用写代码的方式计算需要从新建工程开始、引一些相关的依赖、编码、打包、提交。把这个事变成 SQL 的方式只需要 SQL 文本。我们用平台的 SQL 解析引擎去加载每个任务的 SQL 文本，就可以生成一个 SQL 任务。

4. 实时数据接入



最终的数据流平台是 Kafka 集群。服务器上的日志这部分数据是怎么到达 Kafka 集群的呢？我们是通过一个私有云平台。业务方可以在私有云上提供项目名与文件路径，提交自己的采集申请，由运维人员审核之后，自动下发配置通过 rsyslog 将实时的日志上报到 topic 上，最终到数据流平台。

① 元数据化

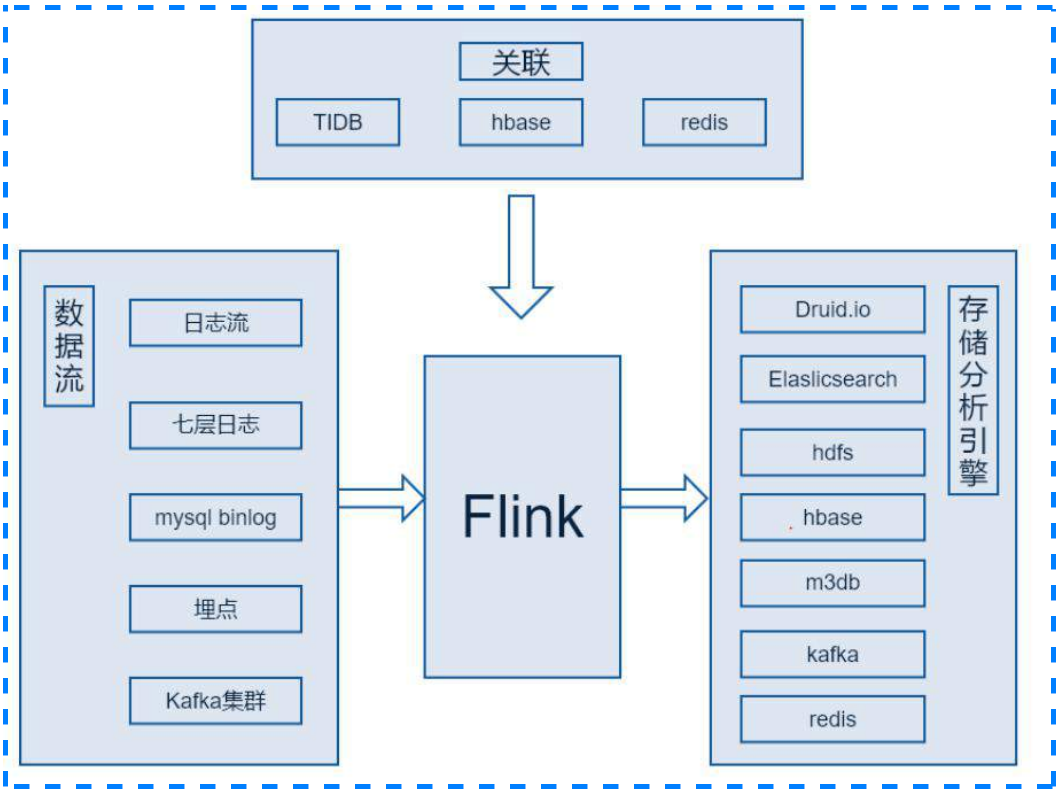
对于数据流平台中的实时数据。我们会有一个初期的处理，对标准的数据格式比如 Json 数据、Tab 分隔，可以预先解析实时数据生成它的 Schema，如果后期进行实时计算用到 SQL，可以根据我们解析的元数据帮助用户自动的生成 DDL。节省了很多任务开发过程中繁琐的建表操作。

② 实时云端控制台

实时云端控制台其实是基于 Kafka 做的实时控制台消费。通常需要预览 Kafka 中的数据都是去客户端输入命令，在实际数据流的使用过程中，有很多用户有预览数据的需求，为了统一管理客户端我们把它迁移到了 web 端，可以通过 web 端实时消费 kafka 的数据，还可以做一些类似 grep 的简单过滤。

5. 数据通道

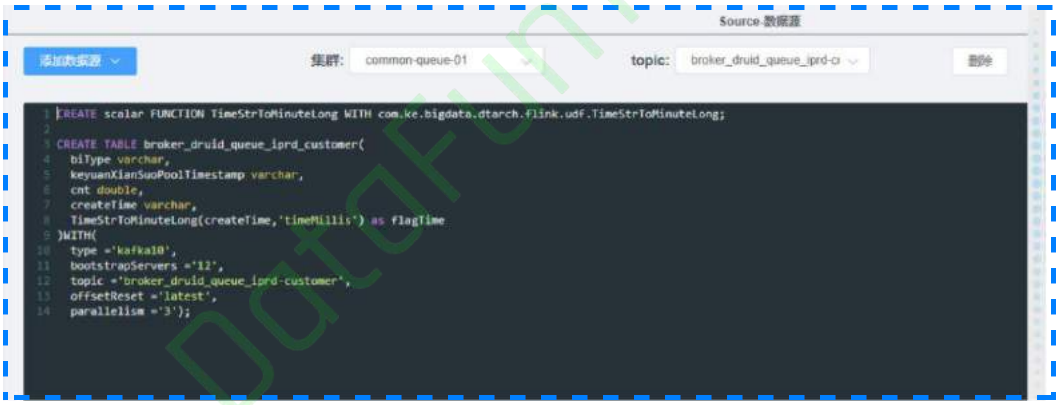
DataFunTalk



实时计算的能力其实可以把它总结成一个数据通道的能力。实时计算可能不会完全满足我们的实时分析的需求，比如通常的需求都是一个多维分析、即席查询的需求。最终我们的方案是通过 Flink 这个引擎去帮我们做这个数据处理与预计算，最终都会落到一个对应存储分析引擎。

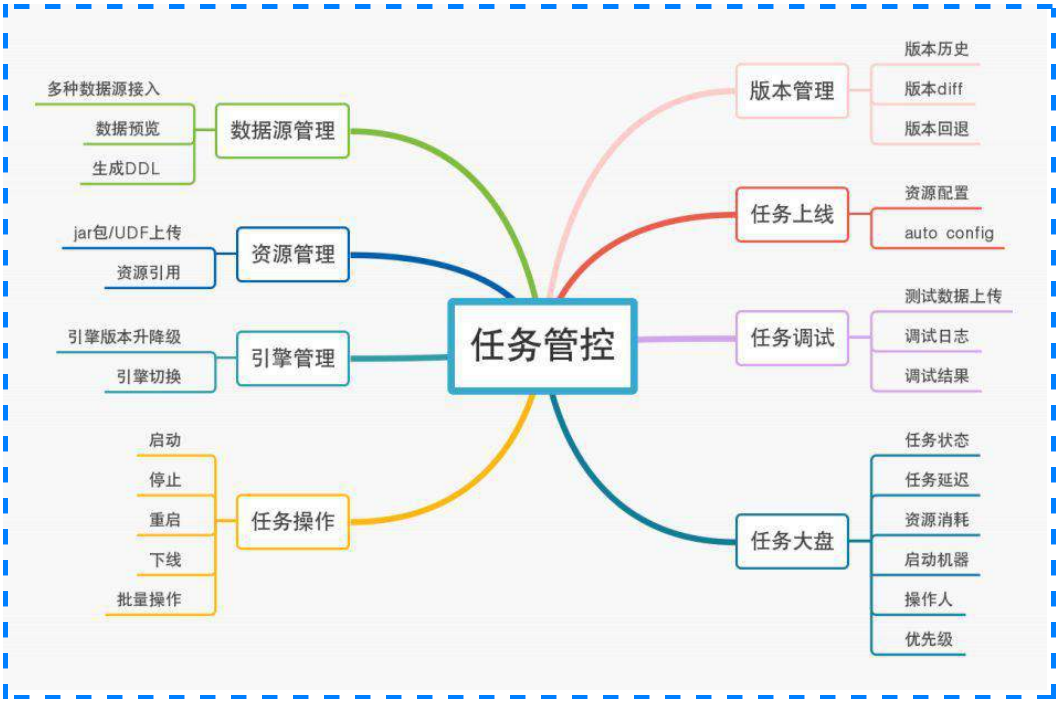
存储分析引擎根据业务需求去选择合适的对应输出。包括实时 OLAP 引擎 druid.io、搜索与分析引擎 Elastic Search、时序数据库 M3db 等等。

6. 任务开发



业务方或数据开发如何在平台上开发任务？有两部分，一个是通常的写代码打成 jar 包提交到平台上，平台帮它管理任务的流程。还有就是主要做的 SQL 的编辑环境，包括前边说到的元数据化在这里可以体现出来。我们把某个 topic 里的数据元数据化之后，这个 DDL 语句可以自动生成，包括将 UDF 关联上任务。

7. 任务管控



对于一个任务流程的管理主要包括上图的这些能力，我们主要支持 Spark2.3、Flink1.8 与 Flink1.9。新的任务优先用 Flink 解决，平台同时支持 Flink 的 per job、session 两种模式。session 模式用于小任务合并以合理利用集群资源，用户可以在平台完成实时任务的所有相关操作。

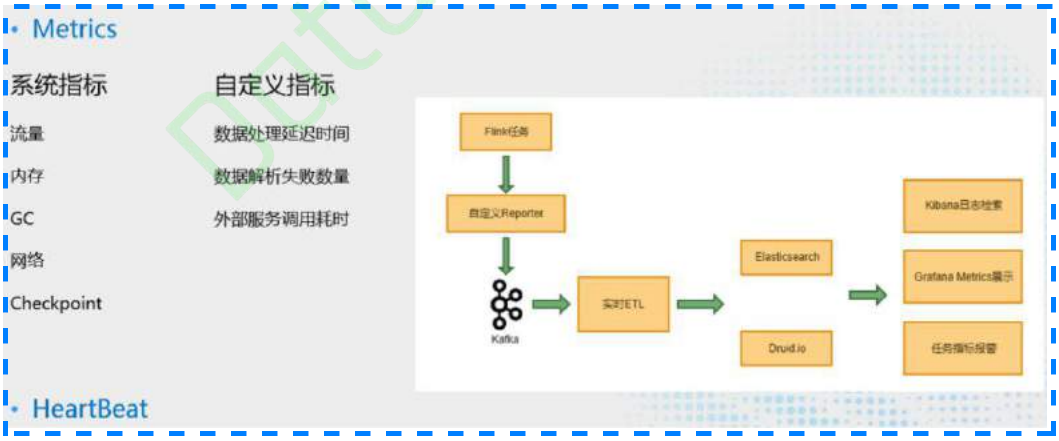
8. Flink SQL 生产实践

SQL 扩展：

在 Flink1.9 之前社区版对 SQL 的 DDL 没有支持，另外实际使用时，对维表关联、输入输出源会有一些定制需求。所以我们基于 Apache Calcite 扩展了 Flink SQL 的能力，定义了 DDL、实现了流与多种数据源维表的关联、UDF 注册等需求的研发。

关于维表的关联，我们使用 TiDB 作为线上 MySQL 的从库，通过关联 TiDB 实现与线上业务数据库的实时关联。对于 Hbase 的关联通过 Async IO 去异步读取 hbase 的数据，并用 LRU 策略去加载维表数据，并支持了 redis 的实时查询。

9. Flink 任务监控和调优



Flink 任务在集群中运行会产生各种指标，包括机器系统指标：Hostname、CPU、Memory、Network、IO，还有任务运行组件指标：JobManager、TaskManager、Job、Task、Operater 等相关指标。这些 metrics 默认会在 Flink UI 中展示出来。除此之外 Flink 原生还支持了几种主流的第三方上报方式：JMXReporter、InfluxdbReporter 等等可以直接配置使用。另外还支持自定义 Report。

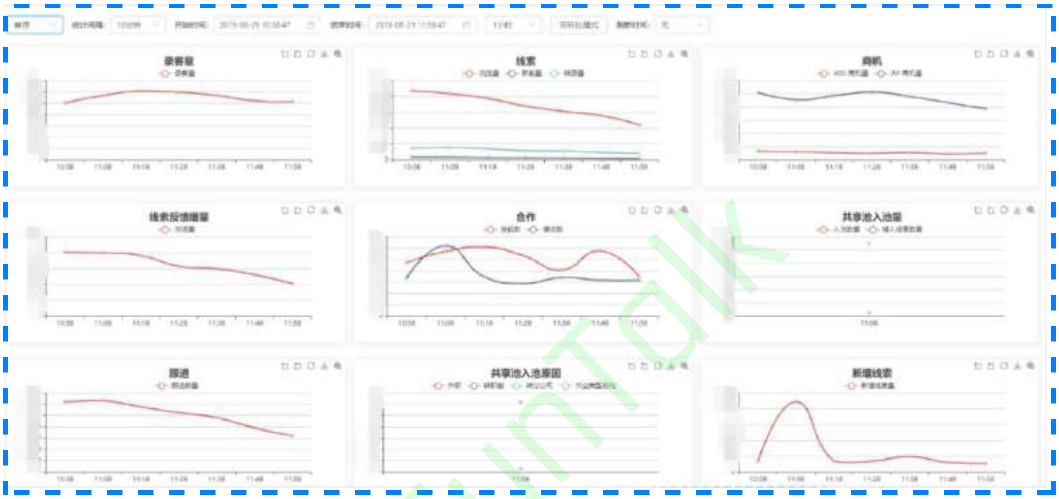
在 Flink 的 metrics 基础上，我们还根据业务场景自定义了数据处理实际延迟时间、数据解析失败量、外部服务调用耗时等指标。所有的指标通过自定义的 Reporter 上报到 Kafka，再通过一个实时的 ETL，把指标结构化后输出到 ES 和 Druid 里，按任务的维度在 Grafana 中分类展示，并提供明细的数据查

询。根据这部分 metrics 数据，我们也会定义一些规则去监控任务的各种异常状态，做一些定制化的报警。

这个是收集上来一些指标的监控，包括任务的输入输出速率，算子，JVM，任务运行时的其它指标。

分析 Flink 任务的延迟，倾斜，或者失败的原因很多都是依靠这些指标，比如根据 JVM 的 metrics 情况对堆内存做出调整。

10. 应用场景-业务 BI 指标实时分析

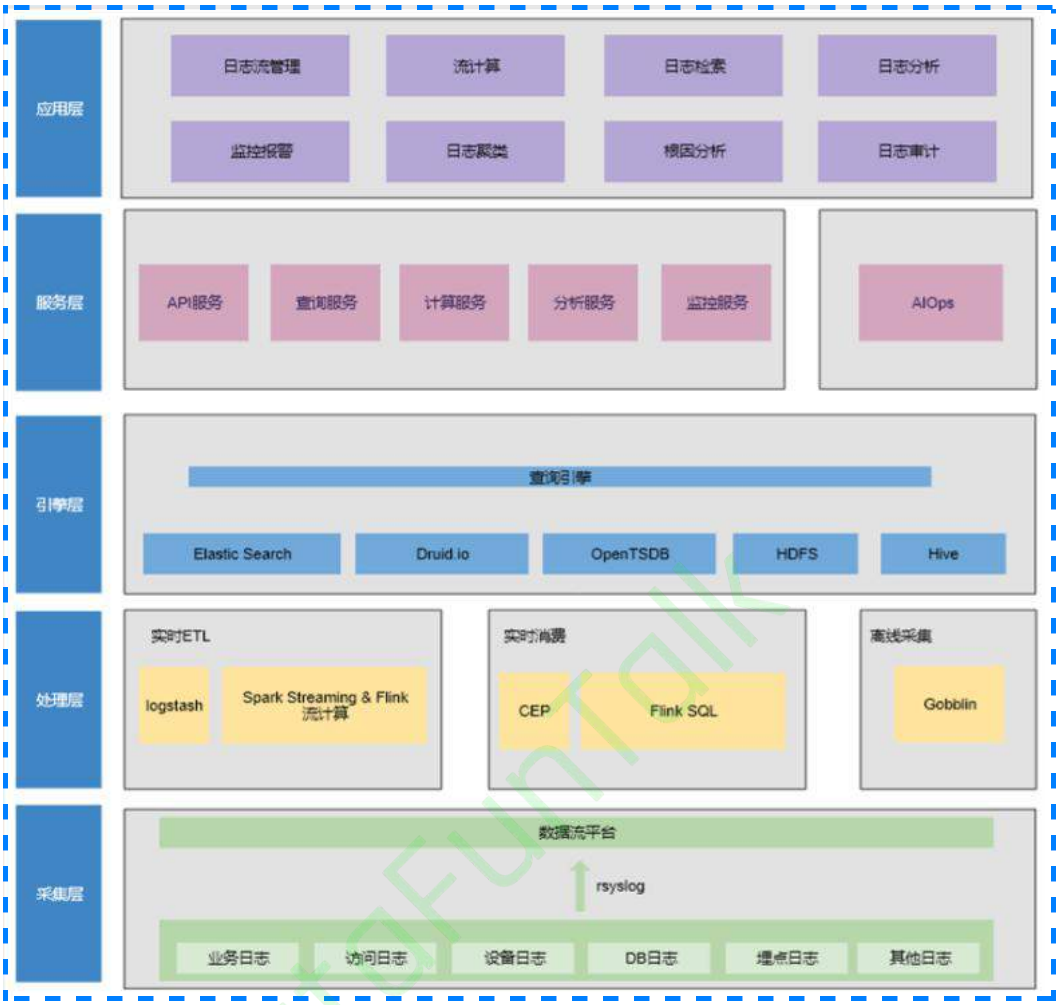


上图是流计算在业务指标实时分析上的应用场景。它是基于业务数据库 binlog 的实时分析，这些业务 binlog 能够反映出业务实时变化情况。如上图是其中客源的部分业务指标，通过流计算对 binlog 中对应业务表的 DML 操作的提取以及分析。可以实时掌握业务 BI 指标的变化趋势。

——实时分析监控平台-FAST——

实时计算除了满足业务的需求，还有一个非常通用的需求就是监控。我们内部的平台叫做 FAST 天眼平台。

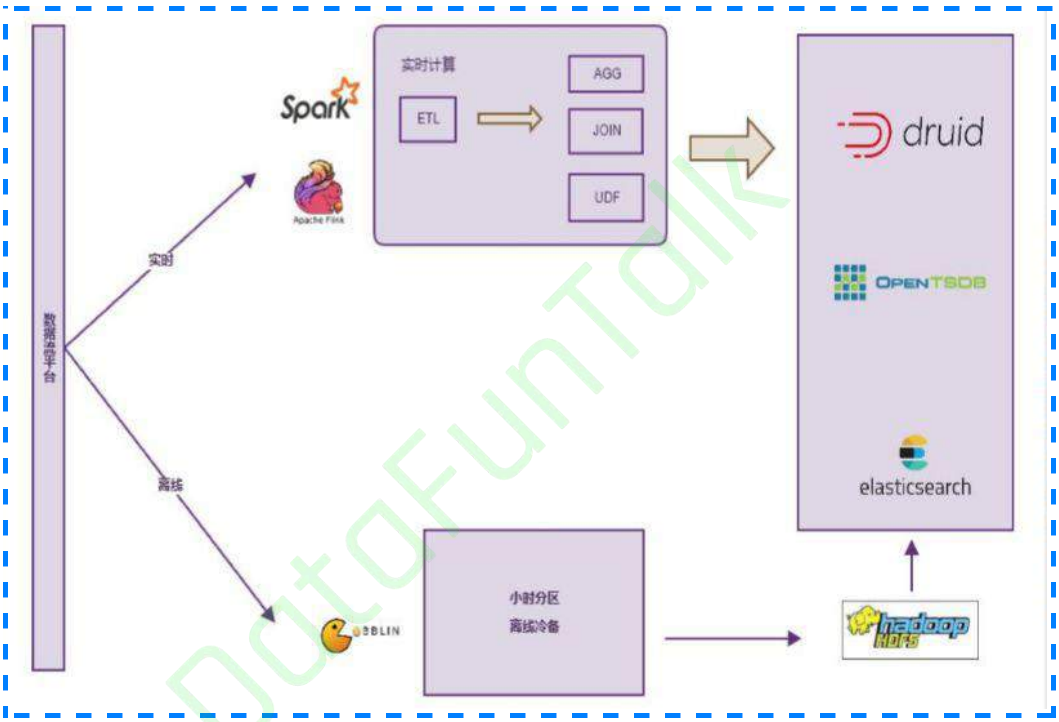
1. 实时分析监控平台架构



这个监控分析平台主要是分析日志。日志的种类包括业务的 log、Nginx 的 access log，还有一些 DB 的日志、中间件日志、网络设备等日志。这些日志通常通过 rsyslog 采集到 kafka 里，通过 Spark Streaming/Flink 实时消费。清洗计算后结构化的数据。会落到日志的的存储和分析引擎，在这一步也会用 gobblin 把 kafka 里的数据离线的备份到 hdfs。一是因为很多情况下实时不能保证数据的准确性。需要离线校准的时候需要一份 kafka 中的原始数据。二是可以满足日志长期存储后续用于审计。在服务层我们提供了一些基础服务，包括所有引擎对外提供的 API，查询，计算，分析，监控报警。依托这些清洗之后的数据，通

过这些日志找出他们之间的关联关系，可以用来实践 AIOps。在应用方面，可以看到实时计算与日志结合具体能帮我们做哪些事。有日志的检索，日志聚类。比如日志一分钟报了一万条 Error，如果 RD 去排查问题不可能看完这一万条再找到问题，所以需要一些文本聚类。包括日志相关的分析。

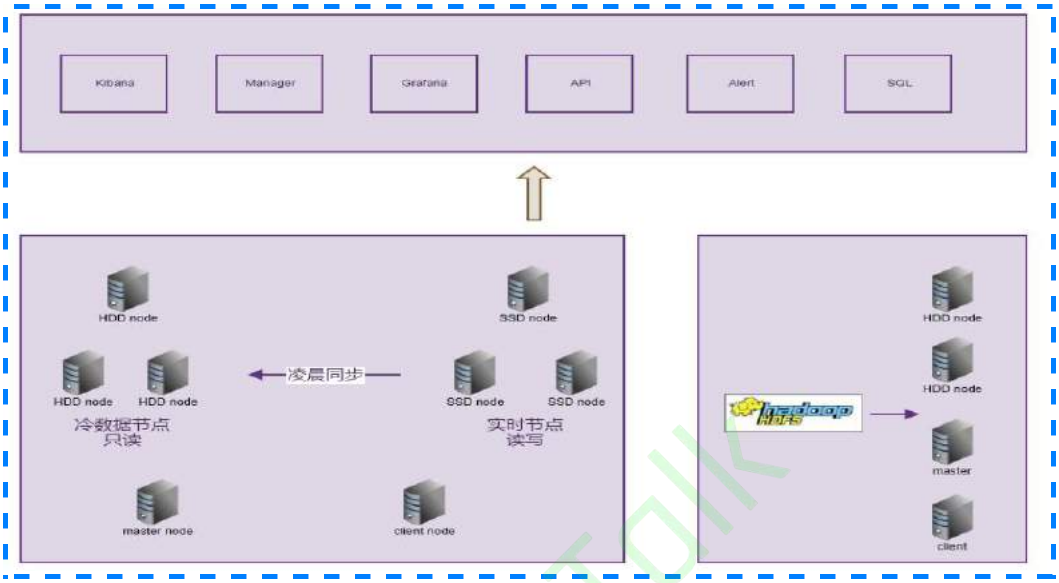
2. 日志计算



在日志的解析和计算方面我们提供了可视化的配置页面。可以通过正则、分隔符、json 的解析提取字段，由用户去解析自己的日志。这一部分计算引擎对用户是屏蔽的，因为对用户来说，只要配置规则即可，通过配置生成实时计算任务。对于规范化日志这些可以自动解析，如果是任意的非结构化日志就需要自己配置规则，然后用 Spark Streaming 或者是 Flink 去做 ETL，包括聚合，join 这些操作。离线的部

分会每个小时把原始数据拉一份到 HDFS 冷备，在 hive 中用这些规则进行离线的 ETL。

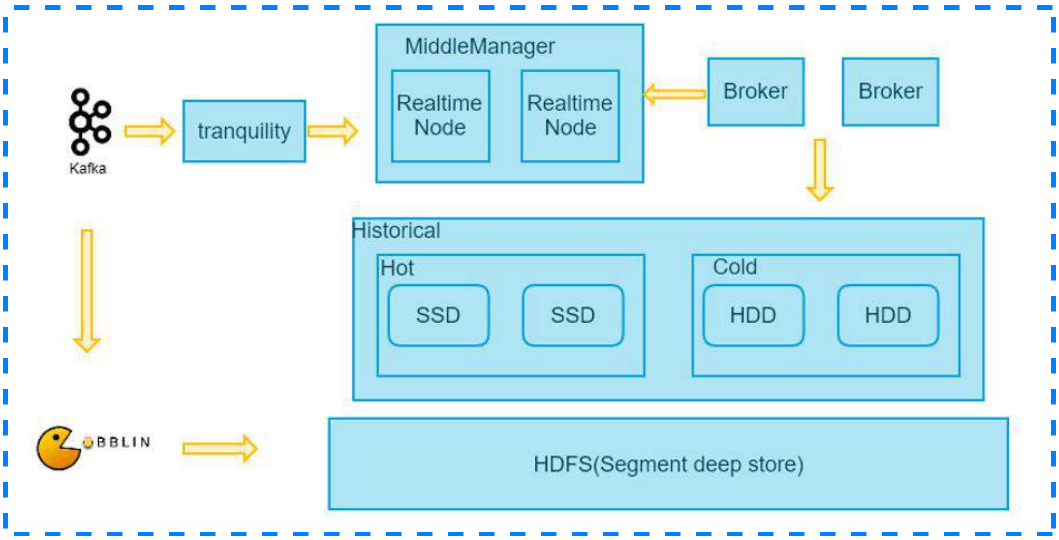
3. 数据存储 - Elasticsearch 存储集群架构



实时计算的流程最终把日志的明细数据存储到 ES 集群，最初我们的几套 ES 集群存储是全 nvme SSD 的。随着接入的服务越来越多，日志量越来越大。需要一个满足日志存储时间很长但成本不会明显增加的架构。具体的实现是通过对 ES 节点划分为 warm 与 hot 冷热两种节点。因为检索场景中，一般最近一两天的日志是查询频率比较高的，所以挂载大容量的 HDD 节点适合存储历史数据即查询频率较低且没有实时写入的索引。在凌晨业务低峰期把冷数据通过标签转到只读的冷数据节点。对于查询几个月之前的数据需求，这时需要之前冷备的那份 HDFS 上的数据，通过 hadoop-es 将他加载到 ES，基于 ES 集群上中的数据。我们也提供了检索

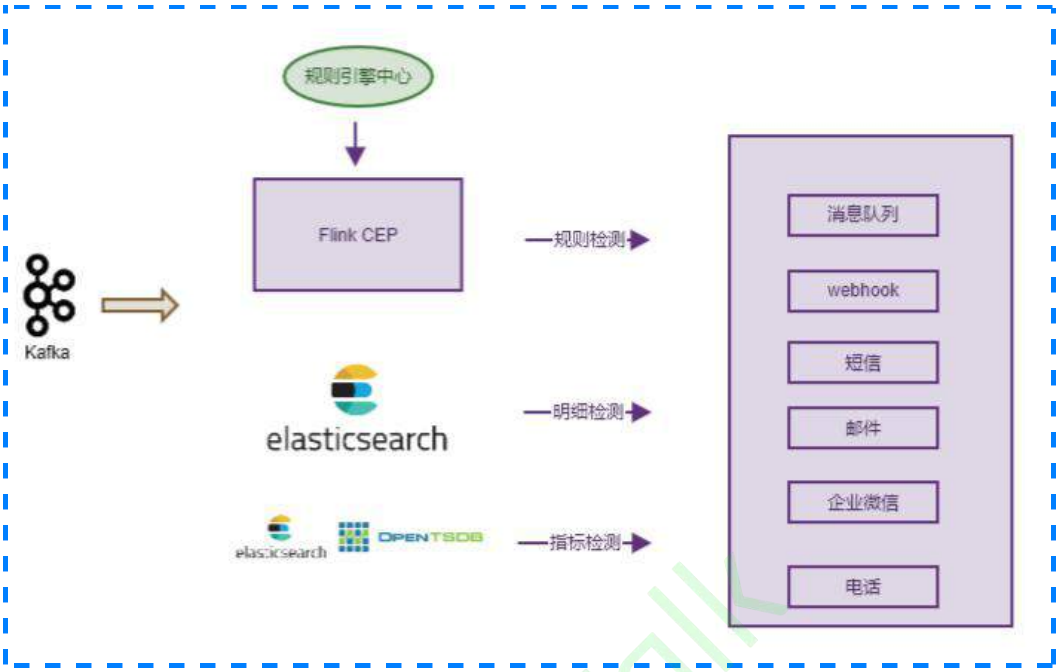
api 服务、下载服务，比如提供给 QA 通过脚本拉取日志进行版本比对的 diff 测试。

4. 数据存储 - Druid.io 集群架构



实时分析这方面现在用的是主要是 Druid.io。Druid 是一个实时 OLAP 的引擎。实时的将我们实时计算的结果经 Kafka，用 Tranquility 将数据写入 Druid 的 datasource。但因为 Druid 一般我们会设置一个时间窗口，如果有上层实时计算任务延迟，数据就可能会丢失，因此需要离线校准来保障数据准确性。一般用 MapReduce 将 hive 中离线结构化的数据重新索引到 Druid 对应的 datasource，覆盖实时写入的数据。Druid 在存储上也分为热数据的节点和冷数据的节点，hdfs 作为 deep store。

5. 异常检测



监控报警、异常检测是实时计算的主要应用点，我们自研的报警服务可以检测多种实时数据来源。覆盖明细类和指标类的异常检测，支持明细检测、指标聚合、同环比、流量抖动等等异常事件。可以根据业务需求灵活配置规则、订阅报警。Flink CEP 的能力我们也在监控场景引入进来应用在爬虫检测。实时检测到的异常事件会通过各种渠道通知订阅者：电话、企业微信、短信、邮件、callback 等。

6. 应用案例

这个例子是 FAST 天眼平台在业务日志实时监控场景的一个实际案例，左边是业务方在企业微信里可以实时收到自己负责的项目线上的错误日志的推送，右边是实时计算的结果存到时序数据库中，用户可以根据自己的业务场景在 Grafana 中配各种 Dashboard。比如实时的流量、接口的瓶颈，包括一些 QPS，异常看板的展示。整个过程可以由平台用户自助完成。

这个是我们全网服务稳定性分析能力，图中是某一个线上域名的分析报告的一部分。数据来源是公司的 7 层负载均衡的访问日志。可以用它来做整个公司下所有服

务稳定性的分析。展示正常流量、异常流量、SLA 值、请求路径响应时间趋势、QPS 趋势等。可以由这类数据产生实时 QPS 排名，实时稳定性红黑榜等应用。通过实时+离线的结合。可以支持任意时间范围的服务稳定性分析。

——后续规划——

完善 SQL 解析引擎。持续丰富 SQL 解析能力，建设 SQL 调试能力、提升开发效率，降低流计算的使用门槛。

动态的资源管理。目前在资源分配场景，有些任务不需要那么多资源或者优化代码可以提升的计算效率，但是用户会把资源分配的很多。因此需要我们在这个平台能监测任务的资源实际利用率，进行动态的资源调整，合理利用资源。

实时任务问题诊断。希望通过收集和分析实时计算任务中历史常见的错误，后续在任务出现问题或报错时，平台能够给出建议解决方案。

智能监控。希望能用实时计算+机器学习，在流量预测、异常检测、根因分析等 AIOps 场景孵化出更多能力。

通用高效的数据修复方法：Row level repair

嘉宾：Asias He　编辑：赵志慧

导读：随着大数据的进一步发展，NoSQL 数据库系统迅速发展并得到了广泛的应用。其中，Apache Cassandra 是最广泛使用的数据库之一。对于 Cassandra 的优化是大家研究的热点，而 ScyllaDB 则为其提供了一个新的思路。ScyllaDB 是一个基于 C++ 的开源的高性能的 Cassandra 的实现，较之 Cassandra 在性能上有了很大的提升。Nodetool repair 是 Cassandra 日常维护的重要一环，今天主要和大家分享一下 ScyllaDB 在这方面的优化。

今天的介绍会围绕下面五点展开：

- ScyllaDB 介绍
- Row level repair 介绍
- Row level repair 实现
- 实验结果
- 总结

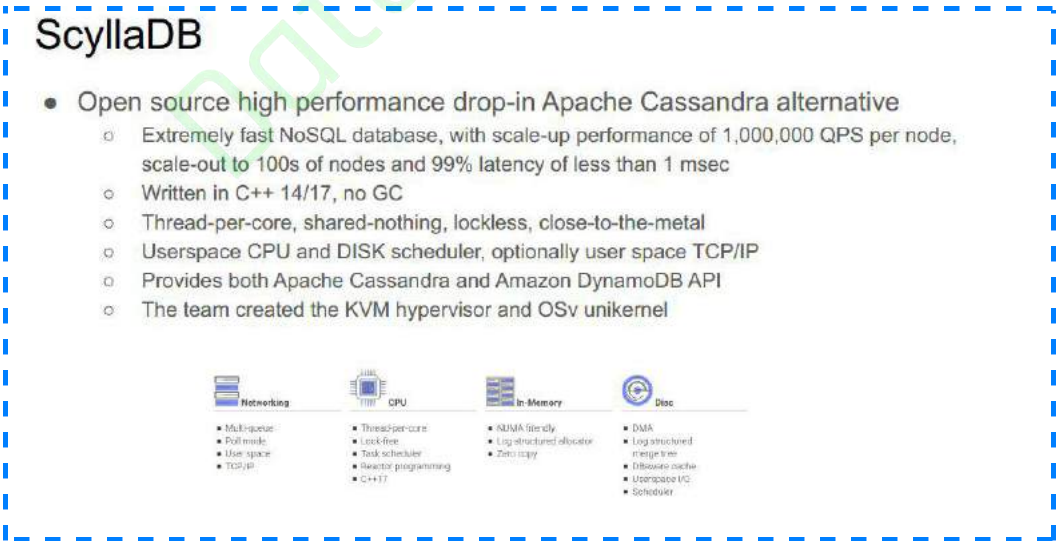
ScyllaDB 介绍

首先给大家简单介绍一下 ScyllaDB：

1. ScyllaDB 的产生背景

我们公司是一家具有较多的底层软件开发经验的公司，团队创始人是 KVM 和 OSv 的作者。对于 Cassandra 数据库的优化，我们进行了一系列尝试。最开始是从操作系统的角度，通过提高操作系统的性能来提高 Cassandra 应用的性能，其效果是提高了 Cassandra 约 20%的性能而无法再获得更高的性能提升。为了更好地优化 Cassandra，团队开始思考是否可以重新实现 Cassandra。我们首先开发了一个非常高性能的 C++ 的开源框架 Seastar，然后基于 Seastar 框架改写的 Cassandra 数据库，即 ScyllaDB。

2. ScyllaDB 的特点



ScyllaDB 是一个开源的高性能的 Cassandra 的实现，具有以下几个特点：

- 一个速度极快的 NoSQL 数据库

  单个节点的 QPS 可以达到 1000000；可以扩展多个节点以提高性能，99%情况下延迟低于 1 毫秒。

- 使用 C++ 开发，没有 GC

- 每个物理 CPU 只部署一个线程，线程间无共享，无锁

  每个物理 CPU 只部署一个线程，每个线程内部会有自己的 task，scheduler。每个线程跑完全独立的 Cassandra 的任务，因此不同线程之间没有共享，从而也没有锁。这样的效果是获得了较大的性能提升，并且在有较多物理核的时候具有较高的可扩展性。

- 在用户态实现 CPU 和 DISK scheduler，以及可选 TCP/IP 协议栈

  在用户态实现 CPU scheduler，不同的任务设定不同的优先级，分配不同的 CPU 资源；进行了磁盘的优化，不使用操作系统的 page cache，完全 DMA 操作，实现 DISK scheduler，精确控制各个模块 DISK 的使用情况；实现了可选的用户态的 TCP/IP 协议栈。

- 提供 Apache Cassandra 和 Amazon DynamoDB 的 API

Row level repair（行级修复）

1. 什么是 repair

首先介绍一下什么是 repair：

修复是 Cassandra 中一个重要的维护操作。Cassandra 中的数据具有多个副本，有可能在写操作发生时某些节点不在线，从而拿不到数据的副本，导致了数据的不一致。这种情况发生时，需要让各个节点的数据一致，即 repair。

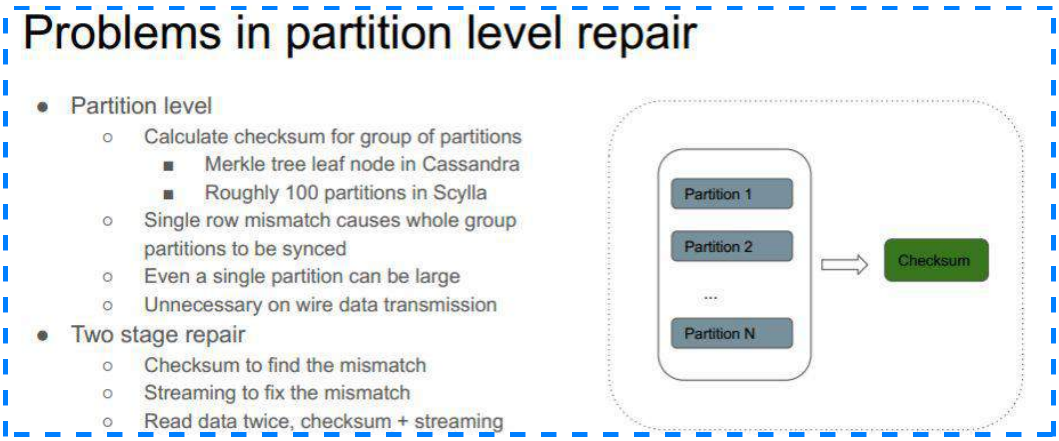
repair 的两个步骤：

第一步：发现不同节点数据副本之间的不一致。

第二步：修复不一致的数据。

2. Partition level repair（ 分区级修复 ）的问题

DataFunTalk



Partition level

目前的 Cassandra 以及 ScyllaDB 3.1 版本之前使用的是 partition level repair。为发现数据不一致需要对一组 partitions 进行哈希得到一组 checksum。对于 Cassandra 是 Merkle tree（默克尔树）中的叶子节点，对于 ScyllaDB 是将 token range 进行拆分，直到每个 range 里面有约 100 个 partition。

其问题在于，哈希的粒度太大，会导致不必要的数据传输：

① 单个行的不一致会导致所有一组 partitions 在网络上传输。

② 有些节点上的分区会很大（达到 5G）。

Two stage repair

Repair 过程包括两个阶段：

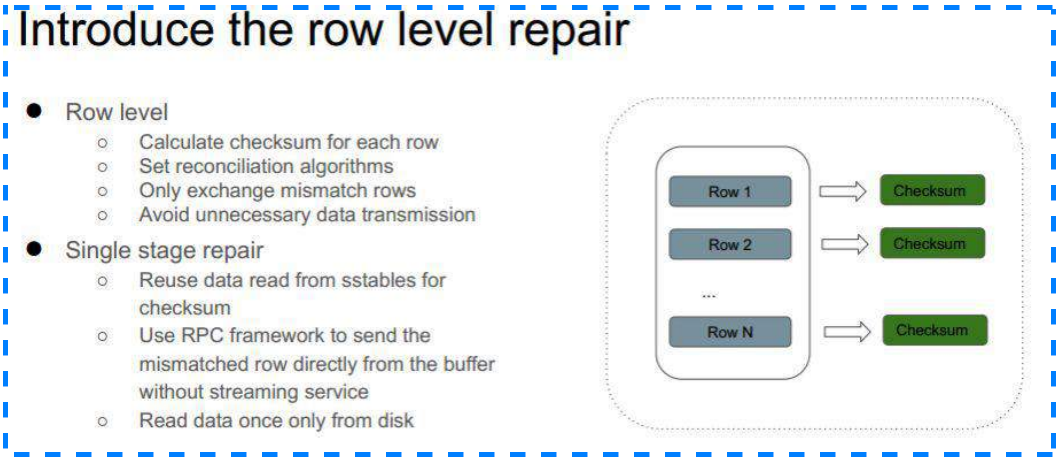
① 通过计算比较哈希值寻找不一致的数据，记录下来不一致的节点和 token range。

DataFunTalk

② Streaming :根据记录的节点和 token range 传输数据进行 repair。

其问题在于，需要两次读取数据，读磁盘开销较高。

3. Row level repair（ 行级修复 ）介绍



Row level

考虑 partition level 修复粒度太大，存在不必要数据传输的问题，减小修复的粒度：

① 对每一行进行哈希计算得到一个 checksum。

② 利用 set reconciliation 算法，使每个节点得到一个统一的哈希集合，包含所有应有数据的哈希值。

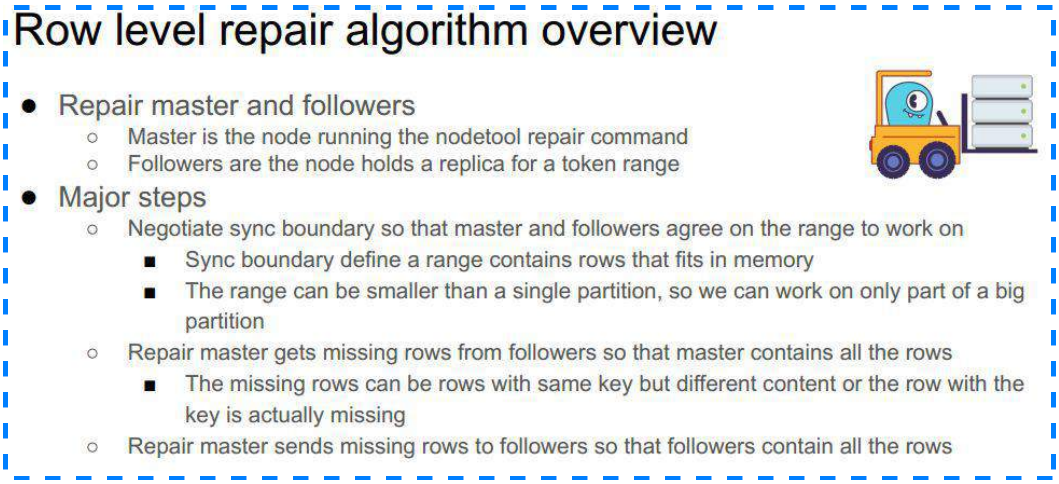
③ 只传输不一致的数据行。

Single stage repair

Row level repair 只有一个阶段，因为通过缩小粒度，正在工作的数据集规模被大大缩小，可以完全放进内存。数据只需要从磁盘读取一次，使用 RPC 框架传输缓存中的不一致数据而不需要额外的 streaming 操作，大大降低了读数据的开销。

下面，重点分享下 row level repair 的实现细节。

Row level repair 的实现



Row level repair 具有两个重要元素：

① Master：运行 nodetool repair 命令的节点。

② Followers：具有数据备份的节点。

Row level repair 有三个主要步骤：

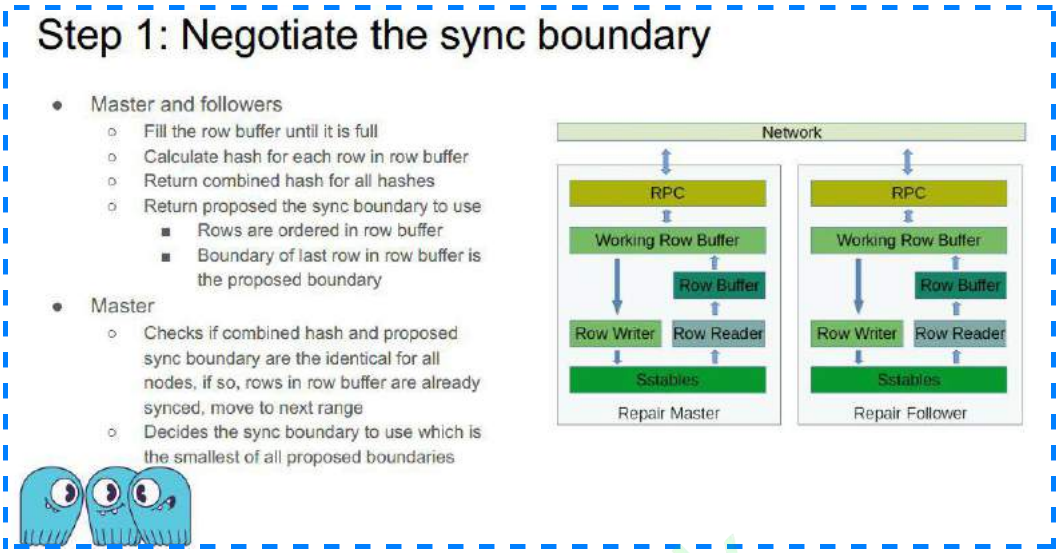
① 首先，master 与其 followers 协商 sync boundary（同步边界），以便确定 repair 工作开始的 range。Sync boundary 确定了一个 range，使其中的数据行可以全部放进内存。这个 range 可以小于一个 partition，因此允许 repair 工作在一个大的 partition 的一小部分上。

② 然后，master 从 followers 获取上述范围内它缺少的数据行，使得 master 最终包含所有数据行。这里缺少的数据行有两个含义：其一是确实缺少的数据，其二是有但是内容不同的数据。

③ 最后，master 通过分析得知 followers 各自缺少的数据行，并将缺失的数据行发送给对应的 follower，最终所有 followers 包含所有的数据行。

下面详细介绍 row level repair 的各个步骤。

步骤一：协商 sync boundary（同步边界）

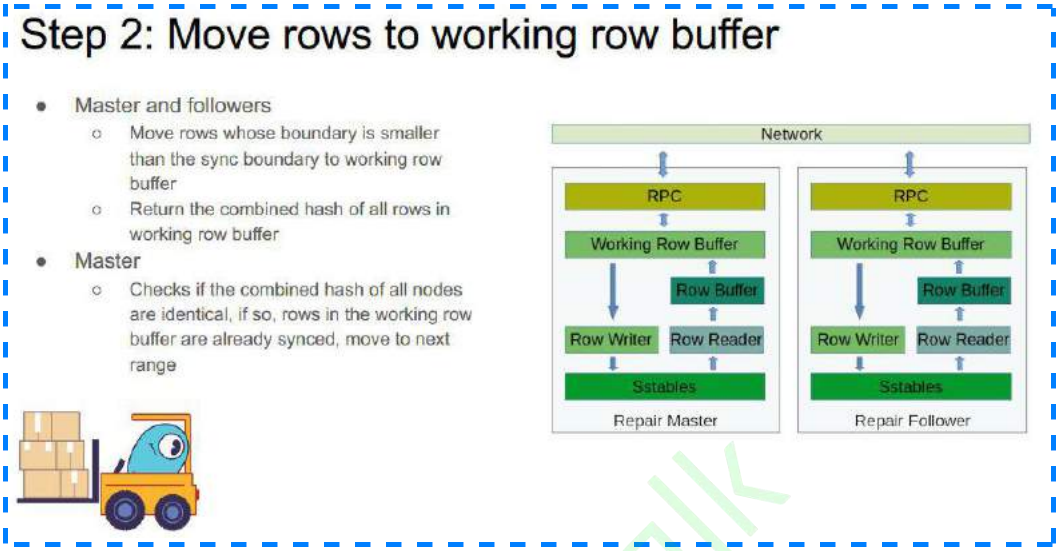


首先，Master 和 followers 读取数据直到设定大小的数据行缓存满（如 32M），计算缓存中每行数据的哈希，并对得到的所有哈希计算一个整体哈希值（如异或）。Followers 将得到的联合哈希值返回给 master，同时也将 sync boundary 返回给 master。这里请注意，缓存中的数据行是有序的，缓存中的最后一行将作为返回的 sync boundary。

然后，此时的 master 节点已经拿到了所有 followers 节点的整体哈希值和 sync boundary，master 比较这些哈希值和 sync boundary。如果所有节点哈希值一致且 sync boundary 一致，则代表此时这个 range 的数据完全一致，因此继续处理下一个 range 的数据。如果 sync boundary 不一致，master 会选择最小的一个作为本轮的 sync

boundary。如果选择的不是最小的，则会导致有多余的数据在本轮同步，违反了有限数据到内存的原则。

步骤二：移动数据到 working row buffer（ 工作缓存 ）

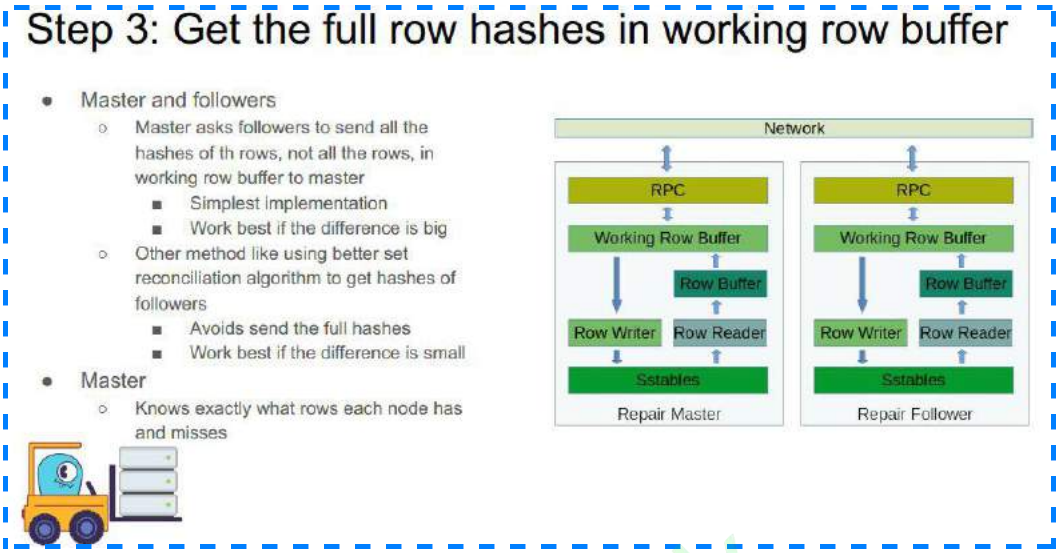


经过第一步的操作，此时所有节点具有一个统一的 sync boundary。

首先，每个节点把 sync boundary 之内的所有数据移动到 working row buffer 中，每个 follower 会返回一个整体哈希值。

然后，master 节点检查 working row buffer 中各个节点的整体哈希值是否一致，如果一致，则继续处理下一个 range 的数据。如果不一致，则进行下面第三个步骤。

步骤三：得到所有数据行哈希值



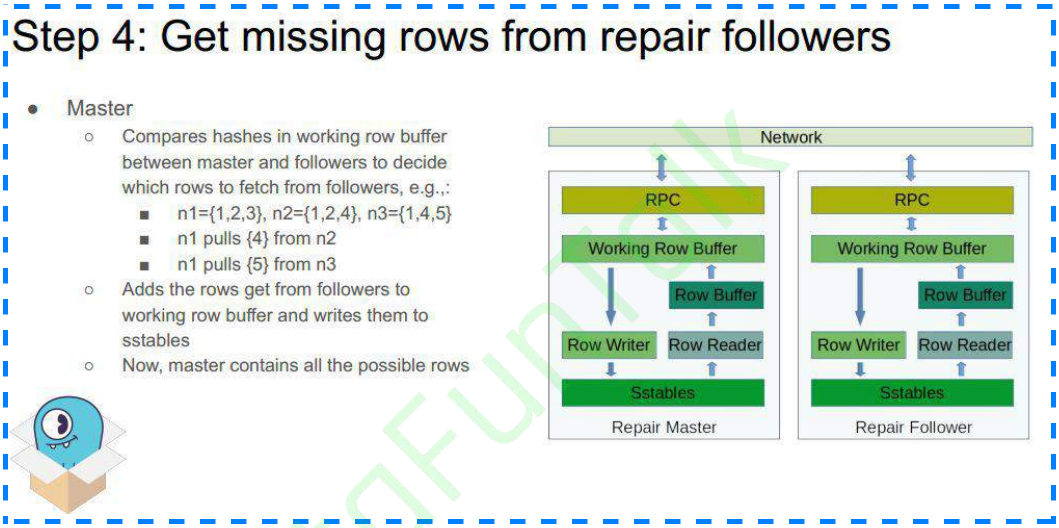
如果存在节点 working row buffer 中数据的整体哈希值不一致，此时则需要对数据进行 repair。Repair 工作首先要进行的，就是得到各个节点 working row buffer 中所有数据的哈希值集合。这个过程有两个比较常用的方法：

第一种是比较暴力的方法：Master 节点要求所有 follower 节点将其所有数据行的哈希值集合发送给 master，这种方法最容易实现，并且在数据差异特别大的时候是最有效（经测试，如果差异数据占比 15%以上，这种暴力方式最有效）。

第二种方法（如 IDF）巧妙地构建一些很小的数据结构，followers 不需要传输所有的哈希值，而是只发送这个数据结构让 master 推测出其具有的数据行。这种方法在数据差异较小的时候最有效。

当拿到所有的哈希之后，master 节点就很清晰地知道了每个 follower 节点有哪些数据行，缺少那些数据行。接下来则需要进行第四个步骤，得到所有的数据行。

步骤四：获取所有缺少的数据行

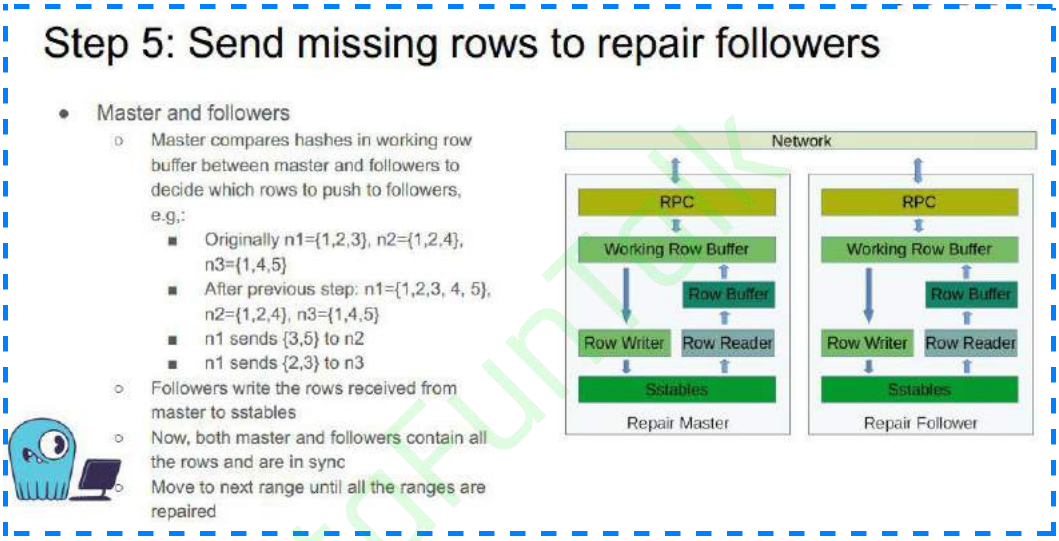


Master 节点通过第三个步骤获得了所有节点 working row buffer 中的数据行哈希。Master 比较本节点的哈希值集合和 follower 节点的哈希值集合，得到本节点缺少的数据行，从而确定应该去每个 follower 节点获取哪些数据，例如：

n1 节点包含数据 {1，2，3}，n2 节点包含数据 {1，2，4}，n3 节点包含数据 {1，4，5}，n1 节点是 master 节点，则 n1 需要从 n2 获取数据 {4}，从 n3 获取数据 {5}，最终 n1 包含数据 {1，2，3，4，5}。

然后 Master 从 follower 节点读取缺少的数据到 working row buffer，并写进磁盘 SStables。此时，master 节点已经包含了所有的数据信息。

步骤五：将缺少的数据发送给 followers



通过前四步，master 节点已经具有了 sync boundary 以内的所有数据，接下来则需要对 follower 节点进行修复。

Master 通过比较本节点和 follower 节点 working row buffer 中数据的哈希值 ,确定需要将哪些数据推送到哪些节点。例如步骤四中的例子：

最开始 n1 节点包含数据 {1，2，3}，n2 节点包含数据 {1，2，4}，n3 节点包含数据 {1，4，5}。经过第四步， n1 包含数据 {1，2，3，4，

5}，n2 包含数据 {1，2，4} ，n3 包含数据 {1，2，5}。在本步骤，n1 将 {3，5} 发送给 n2，将 {2，3} 发送给 n3。

然后 follower 节点将从 master 节点接受到的数据写进磁盘 SStables。此时 master 和 follower 节点中 sync boundary 之内的数据已经达到一致，本轮的 repair 结束。

如果还有更多数据需要 repair，则重复这五个步骤，知道所有数据修复完毕。

实验结果

Repair test results

Test case	Description	Time to repair		Ratio
		Partition Level Repair	Row Level Repair	
0% synced	One of the nodes has zero data. The other two nodes have 1 billion identical rows.	49.0 min	37.07 min	x1.32 faster
100% synced	All of the 3 nodes have 1 billion identical rows.	47.6min	9.8 min	x4.85 faster
99.9% synced	Each node has 1 billion identical rows and 1 billion * 0.1% distinct rows.	96.4 min	14.2min	x6.78 faster

Test setup:
- Three-node Scylla 3.0/3.1 cluster
- AWS i3.4xlarge instances
- Each node has 1 billion rows and 1TB data
- Each row is 1 KB
- Each partition only has 1 row
- No background read or write workload

实验测试了一个三节点集群，每个节点包含十亿行数据，大小为 1TB，每行数据大小为 1KB，每个分区只有一行数据。测试包含三个场景：

第一个场景是 ,其中一个节点完全没有数据 ,另外两个节点有相同的数据，模拟节点的重建。

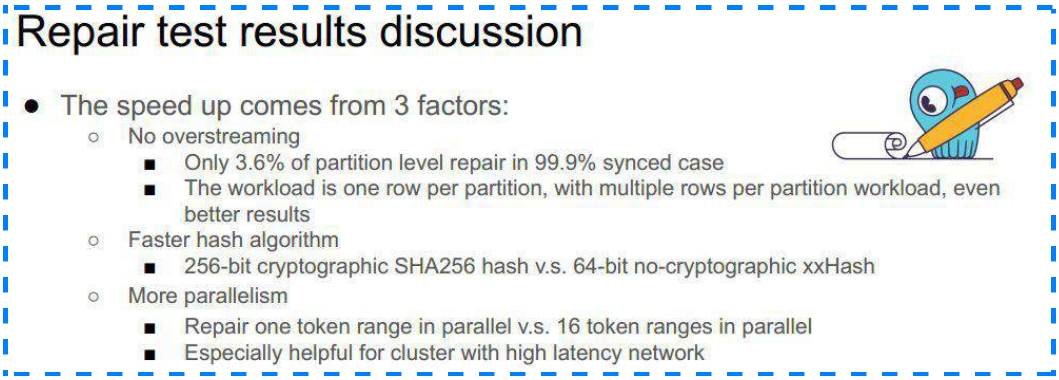
第二个场景是，各个节点数据完全一致，用来测试数据一致的情况下的 repair 操作的速度。

第三个场景是最常见的情况，即各个节点大量数据是一致的，只有千分之一的数据行不一致。因为在集群运维过程中 ,是比较推荐定期 repair 的，如果定期 repair，实际数据差异是比较小的。

第三种场景，row level repair 速度获得了六倍的提升，原因是为什么呢？

Description	Partition Level Repair	Row Level Repair	Transfer data ratio
Bytes sent on wire for 99.9% synced case on master	120.52 GiB	4.28 GiB	3.6%
Bytes received on wire for 99.9% synced case on master	60.09 GiB	2.14 GiB	3.6%

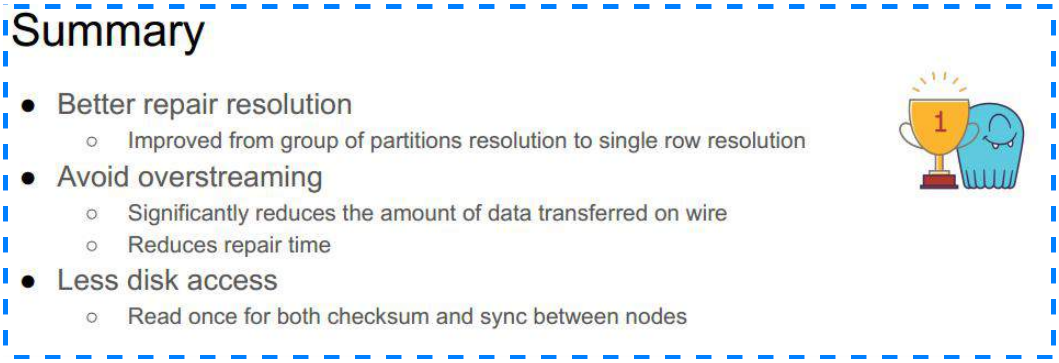
第三种场景，row level repair 速度获得了六倍的提升，原因是为什么呢？我们对 master 节点需要真正传输和接受的数据量做了一个统计，row level repair 只有 partition level repair 的约 4%，数据传输量的大幅度减少是速度提升的最主要原因。



综上 ,从实验结果分析得知 ,row level repair 速度快的原因有三个方面：

- 没有多余的数据传输，本次实验每个 partition 只有一行数据，当每个 partition 中数据更多时，速度提升会更多。
- 更快的哈希方法，partition level repair 数据量比较多，因此使用效果更强的 256 位加密 SHA256 哈希方法，而 row level repair 数据量较少，因此使用 64 位无加密 xxhash，因此速度更快。
- 提高了 repair 的并行度，partition level repair 只并行修复一个 token range 的数据，但是 row level repair 同时并行修复更多的 token range 的数据。Scylla 3.1 是 16 个，3.2 更加优化，可以根据每个节点的内存情况确定并行数量。在未来的版本中 ,还会进一步优化，根据节点之间网络的延迟自动计算一个最优的并行数。这一点对于跨 DC 的集群非常有用，跨 DC 的集群节点之间延迟较高。在这种情况下，为了提高 repair 的吞吐量，需要提高有效缓存的大小。

总结



通过前面的介绍，大家应该对 row level repair 有了大致的了解。Row level repair 降低了数据修复的粒度，从分区粒度到单行数据的粒度，减少了数据的传输量，减少了数据修复所需的时间，并且减少了 IO 请求，减少了磁盘的读取。

综上，Row level repair 是一个通用的高效的数据修复方法，希望能对大家有所帮助。

Cassandra 在 360 的实践与改进

嘉宾：王锋　编辑：王彦

导读：2010 年，Dropbox 在线云存储在国外被用户熟知，同时国内如 360、金山、百度等各个厂商也都陆续推出了自家的网盘类产品；而在 "360 云盘" 背后的存储技术支撑之一就是以 Cassandra 为基础的云端存储方案。自此，Cassandra 在 360 实现技术落地和大规模生产应用，并被持续改进优化，最终形成高峰时期超 10k+ 物理节点的使用规模，成为互联网公司中 Cassandra 生产环境落地规模最大的公司。

本次分享的主要内容是 Cassandra 在 360 的落地实践过程中遇到的问题，以及一些重要的改进和优化，主要包括：

- Cassandra 的特点简介
- Cassandra 在 360 的选型
- Cassandra 在 360 的应用场景
- Cassandra 在 360 的技术演进

——Cassandra 的特点简介——

Cassandra 大致有以下的特点：

Cassandra 完全无中心化设计使得其具备极高的可用性和可平滑的拓展性，并且具有模式灵活，多数据中心，范围查询，列表数据结构，分布式写操作等优势：

❶ 由于其架构在中小规模部署时不需要主节点，相较于完全中心化的分布式存储设计具有更优的成本优势，从 3 台物理机开始一直拓展到几百台物理机，均可完全不停服情况下平滑拓展，整个过程只需要把拓展节点的进程启动加入集群；

❷ 模式灵活使得 Cassandra 可以在系统运行时随意添加或移除字段，这是一个很惊人的效率提升，特别是在大型部署上；

❸ 多数据中心是指可以调整节点布局来避免某一个数据中心失效，一个备用的数据中心将至少有每条记录的完全复制；

❹ 范围查询是指如果你不喜欢全部的键值查询，则可以设置键的范围来查询，对于每个用户的索引，这是非常方便的；

❺ 分布式写操作是指有可以在任何地方任何时间集中读或写任何数据，并且不会有任何单点失败。

除了以上几点，Cassandra 还有如下的优点：

❶ 海量数据，随时在线，分布式数据库，性能极优

❷ always online，无中心，无单点故障引发抖动

❸ 节点对等，配置一致，线性扩展，易于维护

❹ cql/sdk 能力，易用性好，开发者 & DBA 快速上手

❺ 功能强大，主要有以下几点功能，多 DC 两地三中心，在线更改 schema，丰富数据结构，丰富的索引，监控及工具。

——Cassandra 在 360 的选型——

DataFunTalk

选型之始，我们总结评估了云存储的技术需求特征和当时可承载大规模数据的分布式 K-V 数据库——Cassandra 和 HBase，最终权衡之后，使用 Cassandra 做为主要在线存储。对于 "网盘/云盘" 类产品，其主要流量特征为 "写多读少"，要求服务可靠性和数据安全性极高，基本不可容忍服务中断和数据丢失的情况。具体选型分析如下：

❶ Cassandra 相较于 HBase，前者是完全无中心设计，而后者（包括依赖的 HDFS）整体来看是强中心化设计，因此 Cassandra 与生俱来不存在关键单点故障影响服务的问题；

❷ Cassandra 使用最终一致性策略，而非 HBase 的强一致性策略，配合读写策略的处理，Cassandra 可以在确保数据安全性、可靠性、一致性的前提下，出现节点宕机而不需要恢复时间，集群读写不产生任何停顿，而此场景

下，HBase 需要等待 region 重新分配过程，而这个过程大概会有数秒至数分钟的待分配 region 不可读写；

❸ 从技术细节上，虽然二者均采用了 LSM 的架构，但 Cassandra 直接操作本地磁盘，而 HBase 需要依赖 HDFS 共享存储，加之上述所说的架构设计差异，同等基础设施性能的 Cassandra 写入性能优于 HBase 近乎一个数量级，读性能基本持平。

综上所述，Cassandra 可以更好的适用于云盘在线场景。

——Cassandra 在 360 的应用场景——

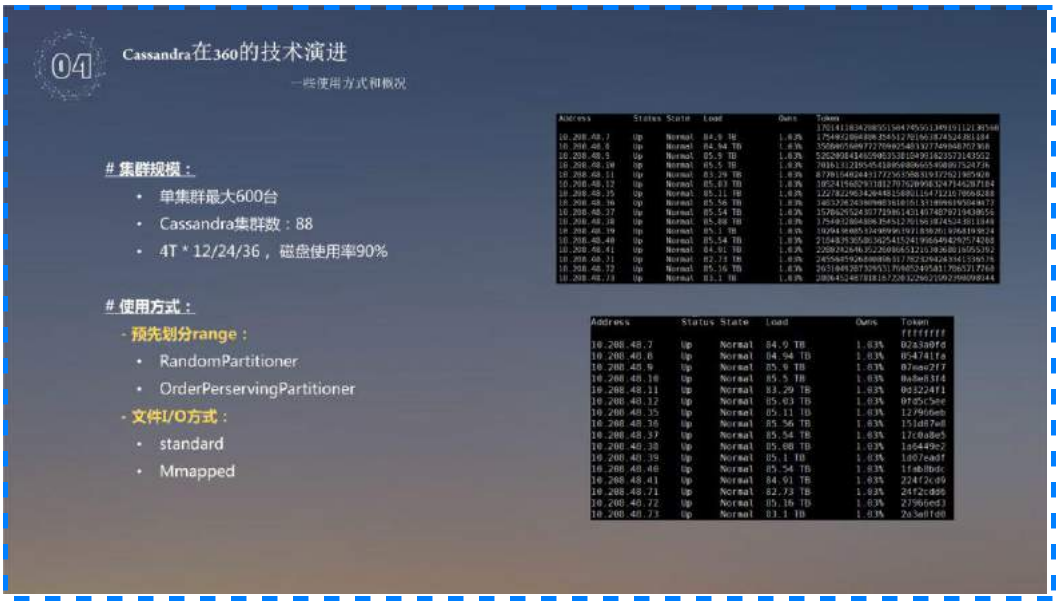
基于 360 云盘使用 Cassandra，云盘从 15 台机器一直到 14000+台机器，应用场景主要是个人数据，自己产品中间的图片，内部视频对象等；在 2015

年，360 云盘转型为企业云盘，机器数量就下降了，到 2018 年，智汇云又继续前行，目前机器差不多是 3000 左右的规模，以上是 360 的应用场景。

——Cassandra 在 360 的技术演进——

Cassandra 自 2010 在 360 开始调研技术落地；2011 年使用 Cassandra 0.7.3 作为基础版本应用于生产环境；2012 年完善数据可靠性和安全性，实现不停机和不单纯依赖读修复的数据快速恢复；2013-2014 年以节省成本为目的，实现可擦除编码技术应用于 Cassandra，在确保数据安全和可靠性的前提下实现成本降低 60% ;2014-2015 年面对超大规模集群的超复杂性问题，实现运维自动化 ,集群具备自主自愈、自主风控等自主运维能力（近 1w5 物理节点，89 个集群，两人运维 ）；2018 年，我们发现 Cassandra 社区版本与 360 版本相当于是不同场景殊途同归 （ 社区为轻 Value，360 为重

Value），并且社区很多好的思路非常值得考虑，因此我们重新调整研发策略，与社区共同成长。



Cassandra 是一种无中心的系统，对于消息的广播，有一些规模的限制，基本单节点到 600 台的时候就差不多了，当时云盘的集群规模，单集群是 600 台，Cassandra 集群规模达到了 88 个，磁盘使用率达到了 90%，主要是为了成本考虑，把磁盘使用率达到了 90%。这其中用的是预先划分 range，毕竟当时没有 VNode，使用预先划分首先是使用 RandomPartitioner，使用例如 hash，md5 让数据随机打到环上，这个是使用最多的；还有一种是 OrdePerservingPartitinoer，这是一种保序的方式，把一些 key 保序的存在环上，文件 I/O 使用的 standard 跟 Mmapped，Mmapped 理论上是减少内存拷贝，对性能很好，但是如果数据量涨到 80%到 90%的时候，Linux

DataFunTalk

内核页表的开销占用量很大，所以最后放弃了 Mmapped 的方式，使用了 standard 的方式。

对于 Cassandra 的改进 ,第一个就是进行可靠性的改进 ,使用 Local Repair 跟 Backup。影响数据可靠性的因素有：

❶ One/Quorum 存在新增副本不足的问题；

❷ 磁盘/扇区故障：文件损坏、数据丢失（ 月均故障 25-30 块 ）；

❸ 现有数据恢复机制存在不完善之处。

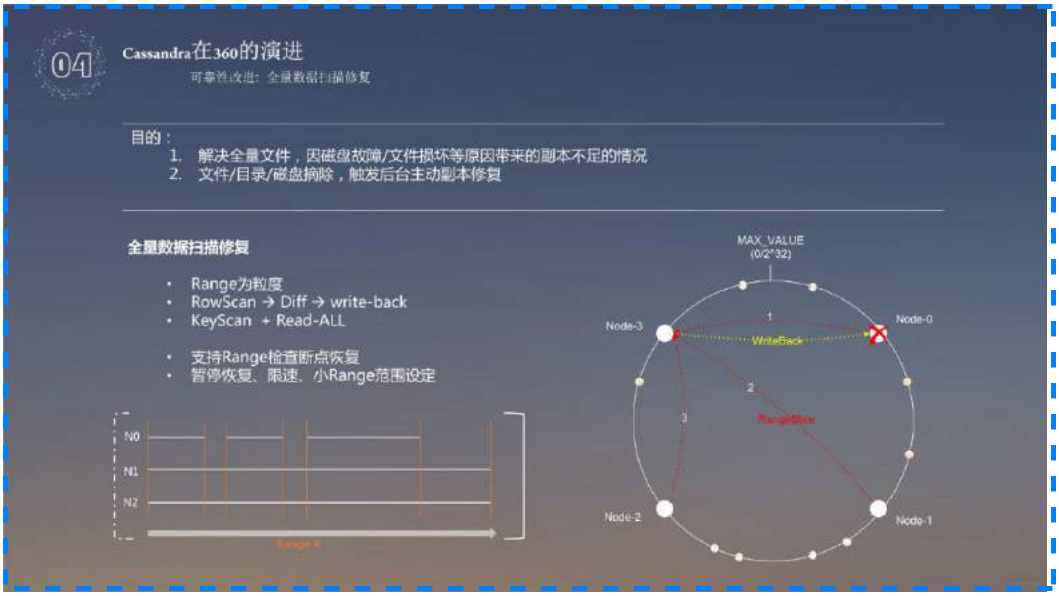
因素 ❶ 是第三副本是否可以成功写入的问题，使用非 ALL 策略写入 Cassandra 时，只要满足写入策略即返回成功，例如 quorum 级别写入 3 副本数据，当两个节点写入成功即返回写入成功，虽然原始设计为了保障第

三副本写入成功使用 hintedhandoff 机制来保证，但程序设计最多能支撑 3 小时的时间，虽然该项可配但也受限于接入节点的存储容量，因此 360 针对此问题做了优化，研发 proxycheck 功能将未成功写入打散到全集群，当故障节点恢复时，基于 proxycheck 会修复残缺副本；

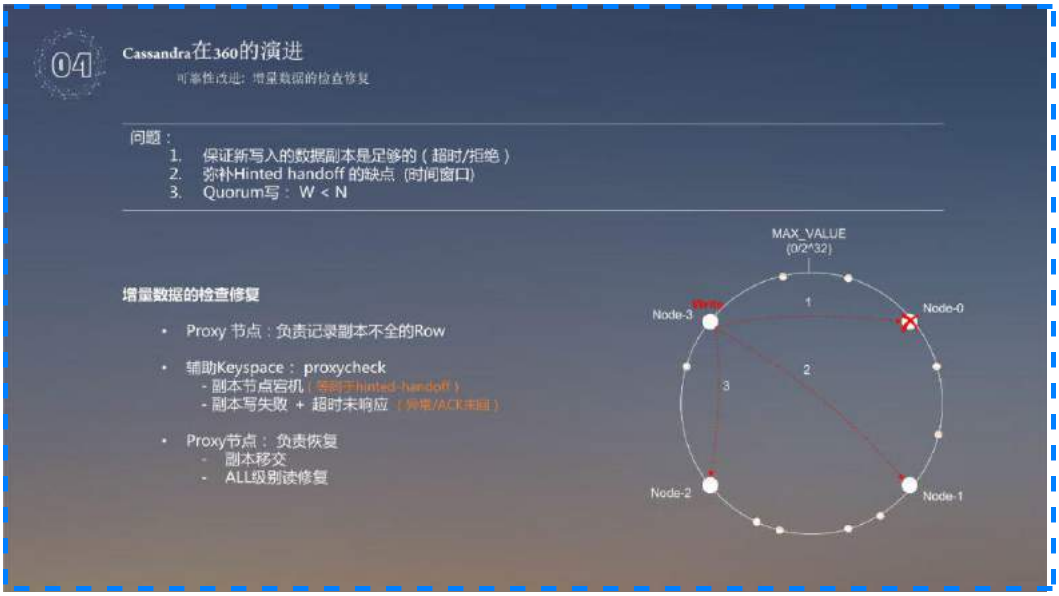
因素 ❷ 是磁盘故障，虽然小规模磁盘很少见磁盘损坏，但对于极大规模的存储系统来说，磁盘故障就变得不可忽略了，而 Cassandra 的架构又决定了如果磁盘损坏造成了副本残缺很难发现，只能等待读修复触发或者 repair 工具修复，但对长时间不读取的冷数据很显然存在较大数据风险；

因素 ❸ 是修复机制，无论是因素 ❷ 导致的还是其他问题造成的数据残缺都需要恢复机制尽快恢复数据，但 Cassandra 读修复对冷数据不友好，repair 工具会耗尽整个集群的资源，对于这些挑战，除了读修复，我们实现了一套相当于 RowRepair 的机制。

首先来说一下文件/磁盘的自动摘除， 存在的问题主要有两点，一点是读写异常 ,SEEKIOException 影响正常读写 ,另外一点是各种修复机制 ,Compact 机制执行失效，针对以上的两点问题，主要采用了基于文件异常访问次数的统计，摘除故障文件数据比重，外部发现基于 SmartCtl 规则反馈，将以上的问题反馈到系统中，就可以精确的知道哪块磁盘有哪些问题。



修复磁盘故障摘除，此处针对的是全量数据的磁盘故障摘除，使用全盘数据扫描恢复的目的主要有两点，一是用来解决全量文件，因磁盘故障/文件损坏等原因带来的副本不足的情况，二是文件/目录/磁盘摘除，触发后台主动副本修复。全盘数据扫描修复，从 Range 的开始，三个节点都读数据，如果数据存在冲突，就使用另外两个节点去解决数据冲突，最后把数据恢复。每个节点都会附一个 range，range 的主要作用就是从三个节点上把数据取过来进行比对，然后把解决冲突的数据恢复掉，另外一种方式使用 KeyScan+Read-All，使用 KeyScan 拿到的是一些 key，对于大量的插入，像云盘用户是大量的插入比较多，删除的操作很少，这种场景下数据存储使用的是 key-value 的数据格式存储，这种情况下，如果节点上丢掉了哪些数据，可以直接使用 key 来修复这些丢掉的数据。通过这两种方式可以解决文件丢失或者损坏的问题。



解决了全量数据，接着解决增量数据的检查修复问题。增量数据检查修复主要存在以下三个问题：

❶ 如何保证新写入的数据副本是足够的（ 拒绝/超时 ）

❷ 如何弥补 Hinted handoff 的缺点（ 时间窗口 ）

❸ Quorum 写存在 W<N

针对以上问题，Hinted handoff 对于 i/o 负载或者 i/o 假死没有考虑到，这种情况下，Hintedhandoff 没有去把出问题的东西记下来，时间窗口存在的问题是如果超时了，丢失的数据可能就记录不下来，所以需要把这两种情况记录下来，以便更好的解决增量数据存在的问题。其原理是：如果提供两种方式，第一种如果 proxycheck 把 value 记录进来以后，数据有问题，可以直接使用另外的副本进行数据修复，还有一种如果不记录的话，可以使

用 all 级别读修复来对数据进行恢复。使用 Proxy 节点负责记录副本不全的 row，超时拒绝导致的三个副本可能只写成功了两个，这种情况也需要记录下来，这种情况下，实时的去做数据的恢复或者副本的补全，使用 proxycheck 表来存储辅助的 Keyspace，把所有检测到的副本不足的数据都记录到这张表中，Proxy 节点还用于记录数据的修复，把数据存储，proxycheck 用了两副本，这样做会加大系统的开销，但是数据的可靠性得到保证。

数据的恢复，涉及到存储，同时，还需要用到数据的备份。当时没有所谓的多 DC 方式，都是自研的备份系统，当时 Cassandra 的集群数量有 88 个，如果采用 Cassandra+Cassandra 的主备模式，那将又是 88 个集群，这是对运维和成本的巨大挑战；因此我们选择了在极大规模场景下扩展更好的 HBase 作为备份存储，使用 Cassandra（主）+HBase（备）方案，这样全球 88 个集群数据备份集中至四大备份中心。大量的数据备份，经常使用的

方式就是消息队列，数据的汇聚会增加运维的成本以及数据的落地然后再去做，这样操作的话，延时会比较高。所以在 Cassandra 中做了一个机制，每个节点负责自己的 range 管理，可以记录到自己的缓存表中，从缓冲表取出来备份到数据中心，使用 Thirft 接口，HBase 跟 Cassandra 的接口完全是兼容的，这样设计 HBase 备份中心就相当于一个 Cassandra 的数据中心了，如果数据大量丢失，或者数据出现大量的错误问题，可以直接无缝切换到 HBase 上提供服务，然后再使用 HBase 备份的数据慢慢恢复丢失的数据，用户完全不会感觉到服务异常，提高了用户的体验。

前面介绍的是数据方面的问题，下面介绍下如何提高磁盘的利用率也就是降低成本。主要是利用虚拟目录来提高磁盘的利用率，磁盘的利用率提高主要问题存在两点：

❶ 节点数量大，SSTable 文件多，磁盘空间导致无法做 major 消重；

❷ SSTable 文件数多，Scan 操作导致 CPU 消耗严重。

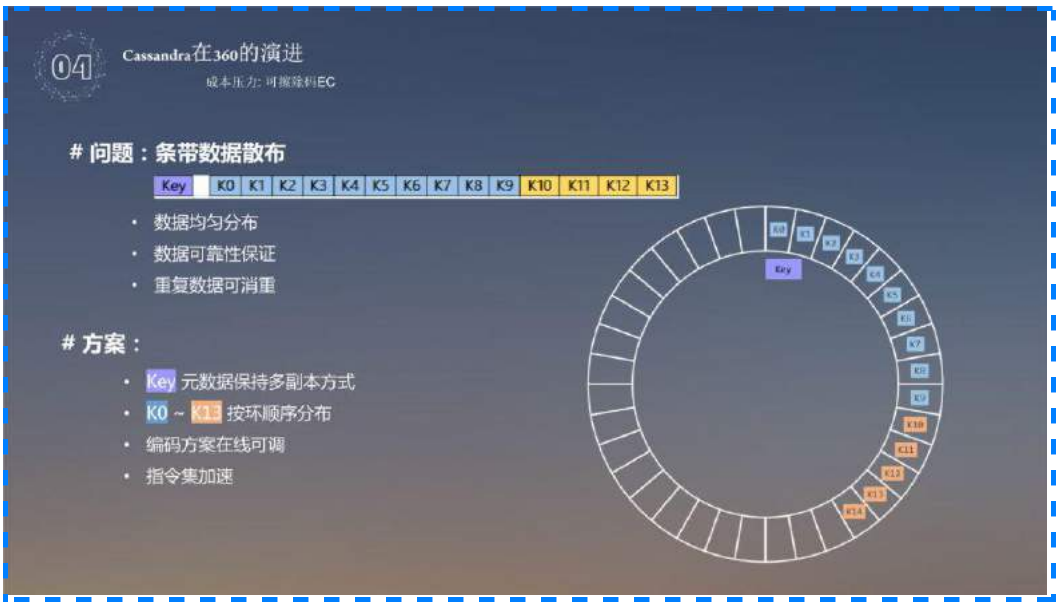
对于这两个问题，当时磁盘的利用率达到 50%就无法再提高利用率，继而我们采用了分而治之的思想，把一个大 range 使用 Daily—Compact 完成数据 SubRange，切分为几个小 range，每个 range 代表一个目录，由于切分以后，数据量变小，每个 range 都可以做自己的 major，可以把重复的部分都清除掉（ 但是如果在磁盘利用率 90%以上，做一次 major 就很消耗 CPU 性能 ），这样做以后，对于 Scan 请求定位 SSTable 打开的文件会更少，效率就会更高，速度也会更快。

避免写放大的问题。对于如何减少写放大问题，主要存在以下两个问题：

❶ 原有的 Compaction 机制（SizeTiered/Leveled）较难避免数据重复参与 Compaction 的问题；

❷ 尤其在 SizeTiered 按文件大小分组 Compaction，插入删除频繁的业务难以消重。

针对上述问题，我们采用给 SSTable 增加 level 概念。正常的是给每层的数据从 level 0 -> level 2，到 level 2 后，compaction 就不会参加，也就说最多做两次。360 对于这一块做了如下的改进：让每层的 compaction 结合虚拟目录，在 level 0 做 compaction 的时候，分成各种各样的虚拟目录进行 subrange，subrange 里边再去做 compaction，这样的话，就相当于虚拟目录没有重复的数据出现，控制文件参与 compaction 的次数，通过这两种方式，使磁盘的利用率达到了 90%左右。



成本压力。基于成本的考量，使用了 EC 的方式，让 3 副本变成了 1.4 个副本，在较少副本数量的同时保证数据的可靠性，同时从数据可用性上考虑的话，数据可用性基本保持以下两点就可以：

❶ 副本方式，也就是连续 3 节点磁盘故障，数据必丢失；

❷ 条带方式，相邻的 14 节点故障任意 4 个数据仍可修复。

对于这个内容，EC 是把原有的数据进行块切分，算出校验块，然后校验块打散到整个集群中，如果丢失了几个块，可以用其他的 10 个块进行修复，再把分散的块 key 存储到 cfindex 的表中。对于前边的条带方式，主要使用切分 value ,value 采用的是 512k 切成等份的 4 等份 ,可以得到 4 个校验块，需要全部打散到不同的数据块上，比如下图中的 k13，k14 不能放到一台机器上，这样才有意义，一旦数据丢失了，还可以方便恢复，如果四个块在一台机器上，坏了两台机器就没法恢复了，key 的数据有两部分，一个是元数据，一个是条带数据。元数据还是保持多副本的形式，但是算出来的条带数据实际上是按环分布，分成单副本的方式去存储，这样其实就可以达到三副本到 1.4 副本 ,编码可以在线调整 ,还可以使用指令集加速 ,通过指令集对 EC 进行加速，这块比较难的问题是如何把 Key 值分散在整个环上，而且还在不同的机器上，如果使用 Md5 算出来 value 值当作 key 值，还是有可能 Key 值存储在一台机器上，所以还是采用了 OrderPreservingPartitioner 保序的方式去存储。

接着做了一个 Keyspace 级别的 Patition 策略，以前的 key 存在以下问题：

❶ RandomPartition 可以解决大部分 Key 随机分布的问题；

❷ key 存储有序问题，OrderPreservingPartitioner；

❸ 是条带数据散布的需要，Keyspace 级别的 Partitioner 设定。

前面说 key 存储用到了 OrderPreservingPartitioner，这样在一套系统里需要两套不同的 partition 机制，如果进行数据交互，就需要既要保持 RandomPartition 的结构，还要保持 OrderPreservingPartitioner 的结构。这样的话，数据交换会变的异常的复杂，所以做了一个消息传递，过程

中还是使用 LongToken 去存，在使用时，还是需要维护两套，当撤出或者加入环中时，都要进行转化，所以系统会看到两套内容。

其他的改进点如下：

❶ Hinted handoff ：外部工具，解决宕机时间过长，超过 Hinted 时间窗口；

❷ MemTable Flush 选盘策略：避免并发 dump MemTable 带来的 CPU 开销，避免小文件的产生；

❸ Cassandra 集群管控，配置自动加载，磁盘自动下线报修。

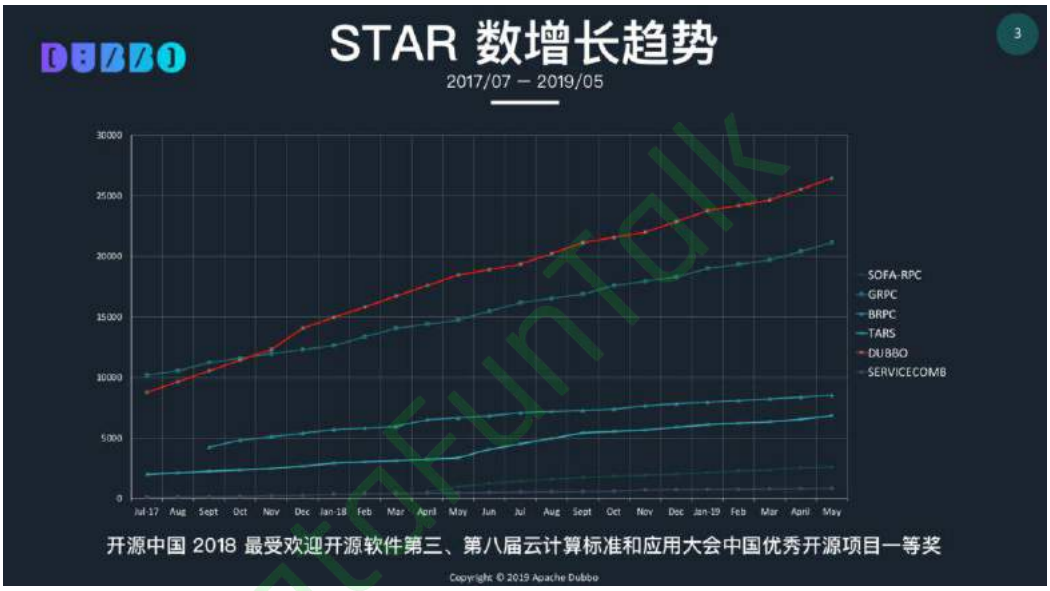
Apache Dubbo Roadmap 2019

分享嘉宾：秦金卫 编辑：董亮亮

导读：Apache Dubbo 是一款高性能、轻量级的开源 Java RPC 框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。自 2011 年开源以来，Dubbo 就是国人开发的最知名开源项目之一，也是我们实现分布式服务化和微服务架构的核心技术之一。本议题介绍 Dubbo 的发展历程、技术生态和 2019 年规划，分享微服务架构的一些实战经验，以及参与开源项目的心得体会。

Dubbo 是一个分布式、高性能、透明化的 RPC 服务框架，提供服务自动注册、自动发现等高效服务治理方案。它是 Apache 第五个国人主导的顶级项目，到 2019 年 5 月初，Dubbo 在 Github 上有 26683 个星，处于领先的位置，今年 5 月，Apache 软件基金会宣布 Dubbo 正式毕业，成为 Apache

的顶级项目，Dubbo 从 2017 年 7 月开始重新维护到现在 22 个月以来，发布了 17 个版本，最大的变化是修复了之前积攒的 bug，同时加入了很多新特性，第三个变化是重新成立了技术社区，贡献者的数量从原先的 41 到现在的 201，引入了 200 多个贡献者，Commiter 所在公司的数量从 4 家扩展到了 18 家，这些人代表了 Dubbo 社区的声音。



接下来介绍下 Dubbo Star 数的增长趋势，从图中可以看出 Dubbo 和其它的框架相比，自 2017 年 7 月阿里重启 Dubbo 开源，到目前为止 Github Star 数，Contributor 数都有了非常大的提升，Dubbo 增长速度很快，处于领先的位置，这主要是由于 Dubbo 整体的功能比较完善并且社区发展的比较红火。

接下来介绍下 Dubbo 的功能体系和生态体系。从功能体系来看目前微服务关注的点 Dubbo 基本都有涉及，图中蓝色的方块代表的是 Dubbo 重新维护之前就有的功能，绿色的方块代表的是 Dubbo 重新维护之后规划去做的，Dubbo 从它的核心的 Api，注册中心，集群的策略，再到它支持的协议和支持的序列化格式和传输，以及它的路由和负载均衡相关的策略都比较完善，之前是有一些功能，但是因为过了几年相比现在的流行的框架比较滞后，当时的设想就是把这些缺的功能给补全，甚至把阿里内部比较前沿的一些技术也融合进去，使 Dubbo 处于一种比较前沿的位置。

Dubbo 从一开始的定位就是一种典型的 RPC 的扩展，它不光支持各种 RPC 协议，比如两个系统之间的 RPC 通信你使用 Hessian 或者 Http 协议，同时你可以把这些对外暴露的服务注册到注册中心，指定协议调用时使用的序列化的格式，使用哪种格式的传输，如果有多台机器做集群，通过什

么样的策略来选择哪些机器符合服务调用的条件，最终哪台机器会被负载均衡选中，来提供服务，这一整套的流程，Dubbo 提供了一个完备的框架，这些早在 10 年前就被设计 Dubbo 的程序员架构师考虑到，重新启动以后，我们将 Dubbo 和 Spring Boot 和 Spring Cloud 进行了深度的融合，把 Sping Cloud 里面一些极佳的特性和好的工具也会融合到 Dubbo 里，另外一个方面就是响应式编程，Dubbo 重启以后对之前支持的不好的异步化的 Api 进行重新梳理和设计，我们就可以使用异步化的 Api 来处理生产和消费的问题，同时在进一步的基础上我们积极拥抱了 RSocket，通过 RScocket 可以很自然地做到响应式的编程模型，性能高效，这会是 3.0 里比较大的一个特性，最后一个方面是 FluentApi，这个主要是考虑到易用性。Dubbo 在注册方面加了很多功能，拥抱 K8s，这是目前微服务鼓励原生做 Service Mesh 方面的基础设施去做的工作。集群方面，不仅支持常规方面的路由，也支持打 Tag 的方式，对集群内部的机器打标签来进行灰度，同时也支持柔性和反压的特性，集群的机器可以根据自身的负载能力动态做出反馈，实现流量权重的动态分配，集群会更具有弹性和柔性，在这个基础上，我们做了更多集群高可用和稳定性相关的工作。协议方面重启以后，除了支持 Rest 协议，同时也包含了 JsonRpc 协议，它可以看做是类似 WebSerivce 的 Rpc 调用，具有简单和跨平台的优点，RSocket 会在 3.0，在序列化方面，加入了 FST，它比较精简，序列化以后的数据量减小到原来的 1/10，还有一个配置协议，是谷歌 2016 年提出来的，底层基于 UDP，但是在 UDP 的基础上

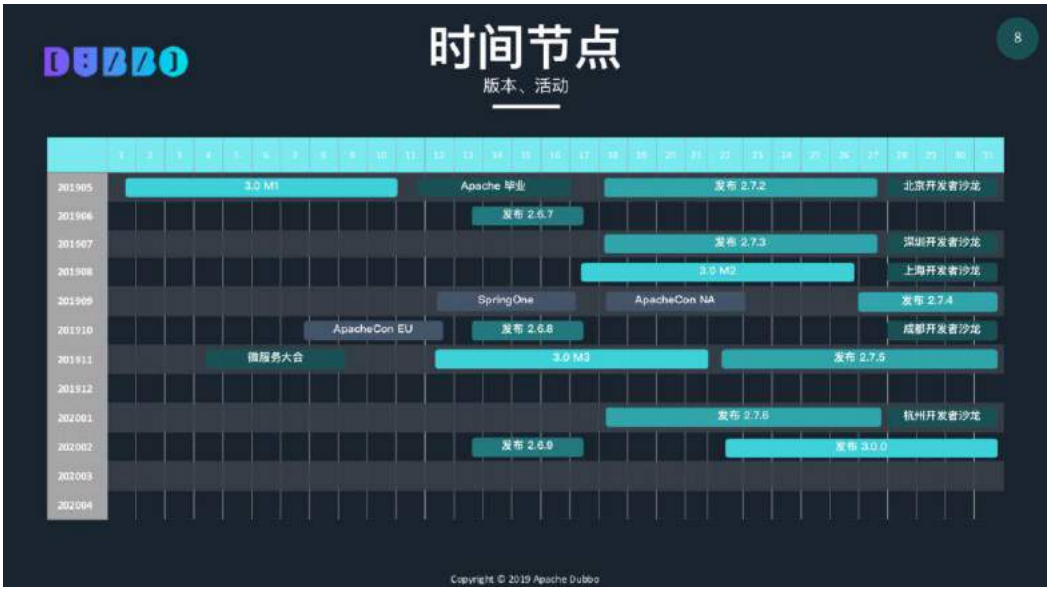
封装了可靠性和安全性，这个协议我们觉得未来有很大的发展趋势，这些是目前的扩展和当时规划的事情。

从图中可以看到，当时规划的事情，黑白框表示的是都已经完成的，剩下的是目正在做的， 以上是目前的一个时间点看到的 Dubbo 的一些功能组件。

接下来我们看一下 Dubbo 目前的一个生态，生态这方面最主要的讲的是 Dubbo 方面的多语言支持，Dubbo 和 Sping Cloud 原生的 Rest 协议相比，差异化在于大家一直认为 Dubbo 只支持 Java，而 Sping Cloud 基于 Rest 可以在不同的异构的系统里使用，所以 Dubbo 为了适配 Spring Cloud 的这个功能，为了后面更好的拥抱云原生，Service Mesh 这一层，做了多语言的接入，为了支持跨语言 Go，Js 主要是 Node Js，Python 等的接入，这些协议都可以通过代理或者 SideCar 的方式来进行接入调用 Dubbo 的服务 ,然后服务发现和其它的管理相对应的基础设施在整套的核心体系的外围提供基础的支持。

实际上今天为止，Js 和 Go 的已经比较完善了，各种外围的配套工具，像服务的发现和配置，新的管理控制平台，各种测试的支持和 Trace 跟踪是由 Spring Cloud 团队的的一个同事做的适配，这是 Dubbo 目前整体的一个

生态，可以看出 Dubbo 在微服务里配套还是比较齐全的。其实对微服务来说最大的问题不是技术问题，因为技术的问题发展的已经很完善了，我们知道把一个系统引入微服务以后，拆成粒度比较小的系统有很多好处，比如每个系统的开发和维护成本会变低，因为系统变得简单了，但是更大的问题也出现了，比如说把一个很复杂的业务系统拆成 20 个小系统，这 20 个小系统都需要单独的做测试，部署，维护，当系统出了问题，需要对 20 个系统去排查问题，这些对自动化测试，自动化运维，全链路跟踪，监控做提出了非常高的要求，这就对整个研发团队的技术成熟度提出了一定要求，只有当测试，运维，业务方面的监控，自动化的 Trace 做的比较完善时，容器化的改造完成以后，再去做服务化会比较合理。微服务和单体在软件设计里面有两条曲线，叫成熟度的曲线，当你的团队规模很小的时候，单体的成本要比微服务的成本低很多，但是随着你的团队规模的增大，微服务的成本虽然起点很高，但是它的复杂度是比较平稳的，不会增加很多，比如你有 50 个开发人员和 500 个开发人员，你使用微服务的方式在保持业务的复杂度和质量的情况下是可以往前扩展的，但是单体的复杂度和成本会随着业务的复杂度逐渐递增，中间有个交叉点，这个交叉点和团队规模，业务复杂度和你的技术成熟度这几个选项有关系，超过这个交叉点就需要进行微服务改造，超不过这个点业也许单体是更好的选择。

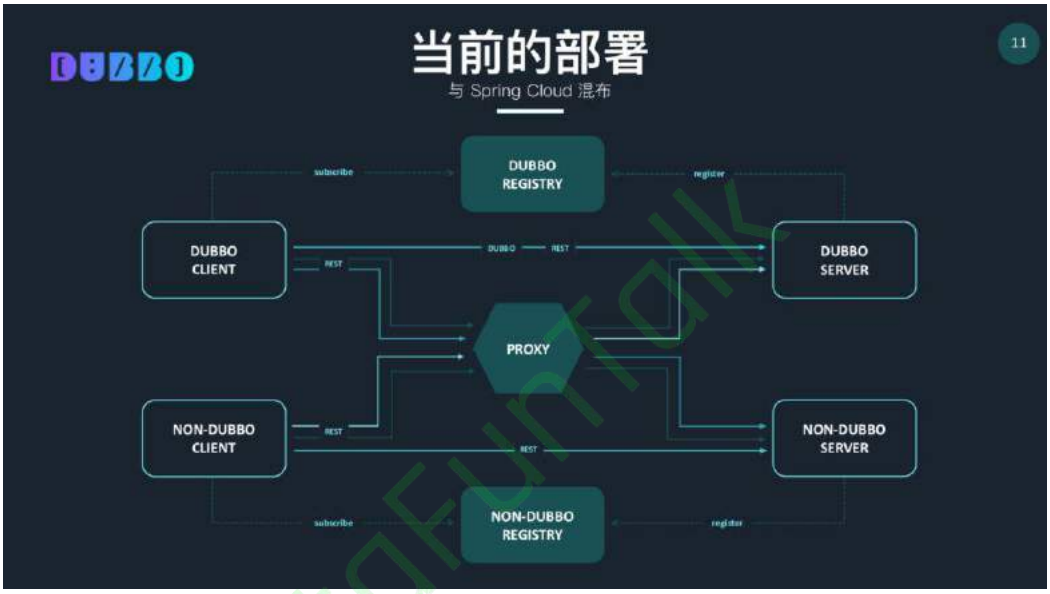


再看一下整个 Dubbo 在 2019 年需要做哪些事情，在 5 月中旬 Apache 已经毕业了，6 月份发布了 2.6.7 的版本，再往下基本上每个月发布一个版本，3.0 的版本计划在 2020 年 2 月份推出，目前维护的主流的版本是 2.7，现在 Dubbo 为了维护社群的生态，每个月会在一个城市举办技术沙龙。

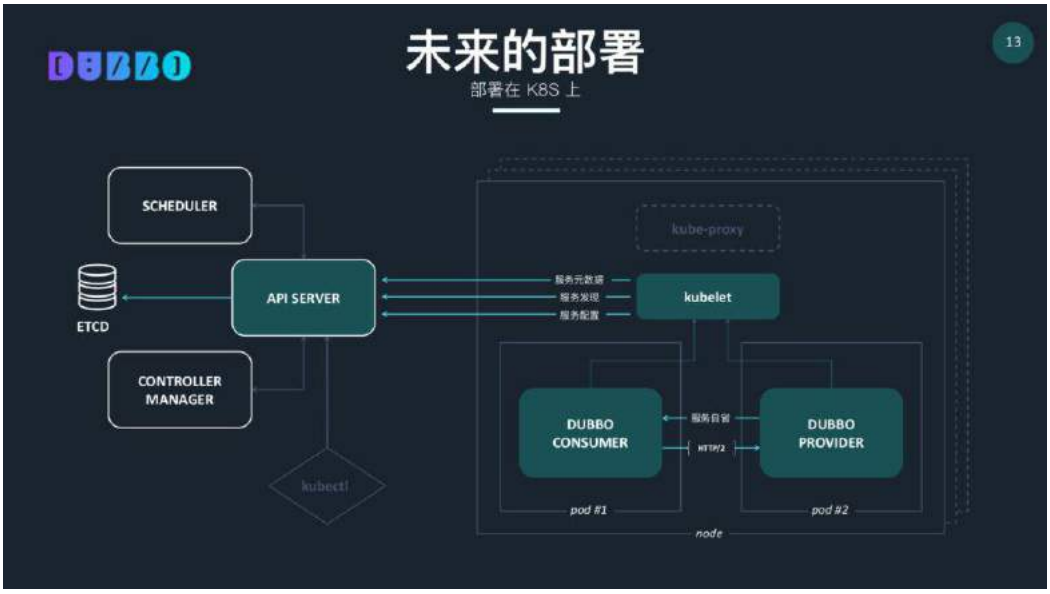
Dubbo 2.7 是维护的主流的版本 ,特别是新注册的机制和与 k8s 的配合都会在陆续的两个月推出来，同时计划在 Q3 的时候会进一步把 k8s 完善掉，把 Dubbo 的 Go 语言的版本发布出来，现在这一块很多公司已经在用，但是没有合并大 Dubbo 的主干分支上，还在单独的一个分支上，但是已经比较成熟，Q4 的时候主要去支持下 Service Mesh ,包括在 2020 年的 Q1 的时候主要去拥抱下云原生和 Service Mesh 的浪潮。

Dubbo 3.0 的路线图更加明确，一方面会继续拥抱云原生，同时会把响应式和异步化做好，把对 Rx 的支持做好， 同时我们一直在设想，做另外一件事情，响应式微服务这个概念非常火，我们知道微服务架构不管外部系统还是内部系统都走的 Rest，对外提供 Rest 是因为容易接入好管理，但是我们知道走 Rest，Http，Json 不是一种效率很高的调用方式，所以在内部应该会有更易用，更高效的方式去管理，去调用，所以这个是现在提响应式微服务架构在思考和尝试的事情。比如说我们也尝试另外一套技术栈，叫 Vert.X，

Vert.X 对外提供 Rest 服务，它对外提供 Rest 调用，内部是一种 EventBus，内部所有的节点通讯是不走 RPC 调用的，类似 actor 模型，一个节点可以直接通过地址向另一个节点发消息，只要在内部一个大的集群内，这样一种方式要远远比走 RPC 调用更高效，而且这种异步的处理方式可以使系统不管存储量还是延迟降低很多，这是后面我们需要考虑的重点因素，好多的东西都会围绕这个来做。

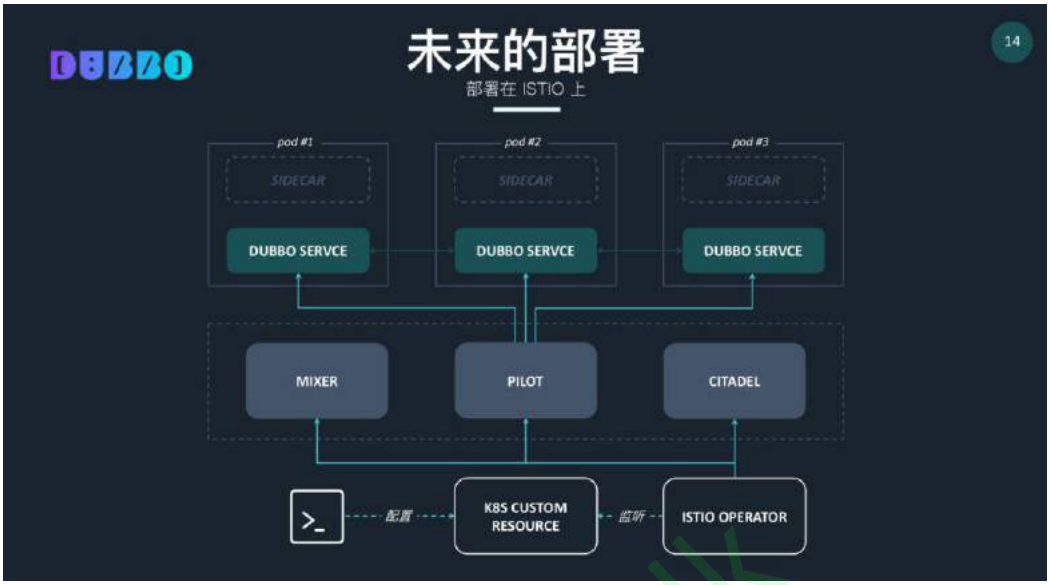


最后两块我们看下整个的部署模式的改造，一块是和 Spring Cloud 结合，另一块是和 Service Mesh 相结合，从当前的部署模式可以看到，Dubbo 的 Client 和 Server 分别去注册和订阅它的注册中心，然后调用的时候通过一个代理和 Dubbo 的 Server 去通讯，然后非 Dubbo 的情况下，会不走 Dubbo 的注册和发现机制，但是一样是可以调用的，希望在未来可以和 Sping Cloud 的技术栈打通，所有的调用都是直连的。



另外一个方面主要是与 Service Mesh 的结合，主要是利用 K8s 的基础的元数据的管理能力和服务发现能力，所有的 Service Mesh 其实就是，不管是使用 Dubbo 还是 Spring Cloud 就会发现我们在我们的程序中会有很多控制路由，控制我们网络流量的流控，比如做我们线程调用的管控和调用速度的管控，和系统 load 负载的管控，这些和业务都没有关系，我们用的中间件可以把这些从业务复杂度中抽离出来，下沉，但是现在负责业务处理的和负责策略处理的在一个进程里，Servie Mesh 的想法是把这些策略的东西向下沉，沉到更下一部的基础设施层，就跟我们业务本身的进程没有关系，业务的进程只需要处理业务本身就可以了，这样的话，下面处理这些策略限流，流控管理，跟踪，服务发现，元数据，把这些东西抽象出来，就是 Service Mesh ，它贯穿在提供服务能力的各个节点上，相互之间可以通讯，这样的话就组成网格，所以就叫做 Mesh，如果 Service Mesh 的东西发展的非常好，我们微服务业务层就会做的很薄，开发人员只需要专注于自己的业务，

提供自己的服务，消费自己的服务，而不用关心自己的服务在哪里，下沉的 Mesh 层都帮忙给处理掉了。



最后是 Dubbo 在 ISTIO 上的部署，除了部署在 K8s 上，还可以部署在 ISTIO 上，其实都是类似的方式，这些都是 Dubbo 未来要做的事情。