

MemFactory: Unified Inference & Training Framework for Agent Memory

Ziliang Guo, Ziheng Li, Bo Tang, Feiyu Xiong, Zhiyu Li[†]

MemTensor

Abstract

Memory-augmented Large Language Models (LLMs) are essential for developing capable, long-term AI agents. Recently, applying Reinforcement Learning (RL) to optimize memory operations—such as extraction, updating, and retrieval—has emerged as a highly promising research direction. However, existing implementations remain highly fragmented and task-specific, lacking a unified infrastructure to streamline the integration, training, and evaluation of these complex pipelines. To address this gap, we present **MemFactory**, the first unified, highly modular training and inference framework specifically designed for memory-augmented agents. Inspired by the success of unified fine-tuning frameworks like LLaMA-Factory, MemFactory abstracts the memory lifecycle into atomic, plug-and-play components, enabling researchers to seamlessly construct custom memory agents via a “Lego-like” architecture. Furthermore, the framework natively integrates Group Relative Policy Optimization (GRPO) to fine-tune internal memory management policies driven by multi-dimensional environmental rewards. MemFactory provides out-of-the-box support for recent cutting-edge paradigms, including Memory-R1, RMM, and MemAgent. We empirically validate MemFactory on the open-source MemAgent architecture using its publicly available training and evaluation data. Across both in-domain and out-of-distribution evaluation sets, MemFactory consistently improves performance over the corresponding base models, with relative gains of up to 14.8%. By providing a standardized, extensible, and easy-to-use infrastructure, MemFactory significantly lowers the barrier to entry, paving the way for future innovations in memory-driven AI agents.

Date: April 2, 2026

Correspondence: Team Leader at lizy@memtensor.cn

Author Legend: †Corresponding Author

Code: <https://github.com/Valsure/MemFactory>

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding, reasoning, and generation. However, to evolve from simple conversational interfaces into fully autonomous AI agents, these models must possess the ability to maintain long-term context, accumulate historical experiences, and personalize their behaviors over continuous interactions. Consequently, memory-augmented LLMs have become a critical focal point in current AI research. While early approaches predominantly relied on static Retrieval-Augmented Generation (RAG) paradigms or heuristic-based memory updating rules, these methods often struggle with memory redundancy, conflicting information, and context-limit constraints during prolonged

execution.

To overcome these limitations, a highly promising trend has emerged: optimizing memory operations through Reinforcement Learning (RL). By treating memory management as a sequential decision-making process, RL allows agents to learn optimal policies for *when* and *what* to extract, update, or retrieve based on delayed environmental feedback. Recent pioneering works, such as Memory-R1 [13], MemAgent [15], and RMM [11], have successfully demonstrated that models trained with RL can autonomously develop sophisticated memory strategies, significantly outperforming their heuristically driven counterparts.

Despite these rapid advancements, the ecosystem for Memory-RL research remains heavily fragmented. Current implementations are typically highly customized, deeply coupled with specific tasks, and scattered across isolated repositories. For researchers and developers, reproducing these memory strategies, or combining different modules (e.g., swapping a retrieval module in Memory-R1 with an LRM-based reranker) requires non-trivial and redundant engineering efforts. In the realm of standard LLM fine-tuning, unified frameworks like LLaMA-Factory [18] have successfully democratized model adaptation by providing systematic, code-free, and highly scalable infrastructures. Yet, a comparable unified infrastructure specifically tailored for the complex lifecycle of memory-augmented agents remains glaringly absent.

To bridge this crucial gap, we introduce **MemFactory**, an open-source, modular, and unified training and inference framework designed to streamline the development of memory-augmented agents. Inspired by the architectural elegance of LLaMA-Factory [18], MemFactory abstracts the intricate memory pipeline into standardized, plug-and-play atomic operations—namely extraction, updating, and retrieval. By orchestrating these atomic modules through a flexible *Agent Layer*, researchers can effortlessly construct, customize, and experiment with diverse memory architectures. Crucially, MemFactory is built from the ground up to support RL-driven policy optimization. Through its integrated *Trainer Layer*, the framework natively employs Group Relative Policy Optimization (GRPO) to fine-tune the agent’s internal memory management strategies based on multi-dimensional rewards from the environment.

In summary, the main contributions of our work are as follows:

- **A Unified Memory-RL Infrastructure:** We present MemFactory, the first comprehensive framework that unifies the training, evaluation, and inference pipelines for memory-augmented AI agents, significantly lowering the engineering barrier for future research.
- **Highly Modular and Extensible Design:** By decoupling the memory lifecycle into atomic modules (Extractors, Updaters, Retrievers), MemFactory enables a “Lego-like” assembly paradigm, allowing seamless integration and mixing of various memory strategies.
- **Out-of-the-box Support for SOTA Paradigms:** The framework natively incorporates and standardizes recent cutting-edge Memory-RL works, including Memory-R1, MemAgent, and RMM, providing readily executable baselines.
- **Empirical Validation:** We empirically demonstrate the effectiveness of MemFactory by training an open-source MemAgent architecture using our built-in GRPO pipeline, achieving consistent improvements across both main-task and out-of-distribution evaluation sets.

2 Related Work

2.1 Unified LLM Training and Inference Frameworks

The proliferation of Large Language Models (LLMs) has catalyzed the development of standardized infrastructures for efficient model adaptation. Beyond the foundational capabilities of LLaMA-Factory [18], which unifies diverse Parameter-Efficient Fine-Tuning (PEFT) and alignment algorithms (e.g., SFT, DPO) within a modular architecture, other integrated frameworks have emerged to address specific scaling and alignment needs. For instance, MS-Swift provides a versatile ecosystem for multi-modal model fine-tuning and deployment, while specialized libraries like VERL [10] offer high-throughput infrastructures specifically optimized for large-scale Reinforcement Learning (RL) pipelines.

Despite the success of these frameworks in standardizing LLM alignment, they remain predominantly designed for stateless sequence-to-sequence modeling. They lack the architectural primitives required to manage the complex, stateful lifecycle of memory-augmented AI agents, such as iterative memory extraction, dynamic database updating, or sequential interactions with evolving environments. Recent memory-centric systems have begun to standardize the construction of memory modules themselves. For example, MemEngine [17] provides a unified and modular library to develop LLM-based memory agents, while Mem0 [1] offers a scalable memory-centric architecture for extracting, consolidating, and retrieving long-term conversational memories. MemFactory draws architectural inspiration from the “Lego-like” modularity of LLaMA-Factory [18] and the RL-centric design of VERL, while also taking cues from recent memory-centric abstractions that improve the composability of memory modules. In this way, MemFactory fundamentally re-engineers the infrastructure to support the unique paradigm of Memory-RL. By integrating static fine-tuning, modular memory construction, and dynamic policy optimization, the framework enables researchers to easily build, train, and evaluate customized memory agents.

2.2 Adaptive Memory via Reinforcement Learning

Traditional memory-augmented LLMs typically rely on heuristic rules and static pipelines for memory storage and retrieval. These pre-defined strategies struggle with long-horizon tasks, frequently accumulating noise or misinterpreting conflicting information. To overcome these limitations, recent research frames memory management as a sequential decision-making problem, utilizing Reinforcement Learning (RL) to dynamically optimize memory policies via outcome-based rewards.

Pioneering works in this direction have explored distinct facets of the memory lifecycle:

- **Structured Memory Operations:** *Memory-R1* [13] trains a Memory Manager to execute discrete CRUD operations (e.g., ADD, UPDATE, DELETE) and an Answer Agent for memory distillation. Optimized via PPO and GRPO, it demonstrates that exact-match rewards are sufficient to teach sophisticated memory consolidation without dense human annotation.
- **Recurrent State Optimization:** To bypass the quadratic bottlenecks of long contexts, *MemAgent* [15] treats a fixed-length latent memory variable as a recurrent state across text segments. It employs Multi-Conversation DAPO to learn an optimal “overwrite” policy, aggressively compressing history while preserving task-critical facts.
- **Retrieval Refinement:** *RMM* [11] dynamically optimizes memory retrieval for personalized dialogues via “Retrospective Reflection.” It leverages unsupervised LLM attribution signals (i.e., whether a retrieved memory was actually cited) as binary rewards to update a lightweight reranker online via the REINFORCE algorithm.

Despite demonstrating the efficacy of RL across diverse memory paradigms—from explicit database updating to recurrent state compression and retrieval optimization—these implementations remain heavily siloed. Each is tightly coupled to custom training pipelines, data formats, and task assumptions. **MemFactory** resolves this fragmentation. By abstracting these core mechanisms into unified modules (e.g., the **Updater** for Memory-R1, the **Agent Module** for MemAgent, and the **RerankRetriever** for RMM), our framework provides a consolidated platform to seamlessly reproduce, evaluate, and integrate these state-of-the-art algorithms out-of-the-box.

2.3 Reinforcement Learning for Policy Optimization

Reinforcement Learning (RL) has become central to aligning Large Language Models (LLMs) with complex reasoning tasks [7]. While Proximal Policy Optimization (PPO) [8] is the prevailing algorithm, it requires an auxiliary value network (critic) comparable in size to the policy model. This effectively doubles the training memory footprint—a prohibitive overhead for memory-augmented agents, where context windows are already severely saturated by prolonged dialogue histories and retrieved memories.

To address this computational bottleneck, Group Relative Policy Optimization (GRPO) [9] entirely eliminates the need for a parameterized value model. For a given input, GRPO samples a group of G candidate responses

and estimates the baseline through intra-group reward normalization. Specifically, the advantage \hat{A}_i for the i -th candidate with reward r_i is calculated as:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_1, \dots, r_G\})}{\text{std}(\{r_1, \dots, r_G\})} \quad (1)$$

This formulation significantly reduces memory requirements while maintaining high sample efficiency, particularly when optimizing rule-based or outcome-driven rewards (e.g., exact-match or LLM-as-a-judge scores) prevalent in memory evaluation tasks.

MemFactory natively integrates GRPO into its *Trainer Layer*. By leveraging GRPO’s memory efficiency, our framework substantially lowers the hardware barriers for Memory-RL research. It enables researchers to efficiently fine-tune complex, long-context memory policies—spanning extraction, updating, and recurrent state transitions—even under constrained computational resources.

3 MemFactory Framework

MemFactory is designed with a highly modular architecture comprising four primary components: *Module Layer*, *Agent Layer*, *Environment Layer*, and *Trainer Layer*. The *Module Layer* operates as the fundamental core of the framework, abstracting the complex memory engineering pipeline into atomic operations, such as memory extraction, updating, and retrieval. By representing these atomic behaviors as plug-and-play modules, the framework ensures high extensibility and customization. The *Agent Layer* builds upon the *Module Layer* and serves as the central policy executor by systematically assembling these modules to implement various memory strategies and generate rollout trajectories during interaction. The *Environment Layer* functions as both a dataloader and a reward manager. It standardizes raw data from diverse datasets into unified contextual states and evaluates the agent’s actions to provide multi-dimensional reward signals. Finally, the *Trainer Layer* employs Group Relative Policy Optimization (GRPO) to fine-tune the pre-trained models loaded within the agents, optimizing their internal memory management policies based on the environment’s feedback. Figure 1 illustrates the overall architecture and the interdependencies of these components within MemFactory.

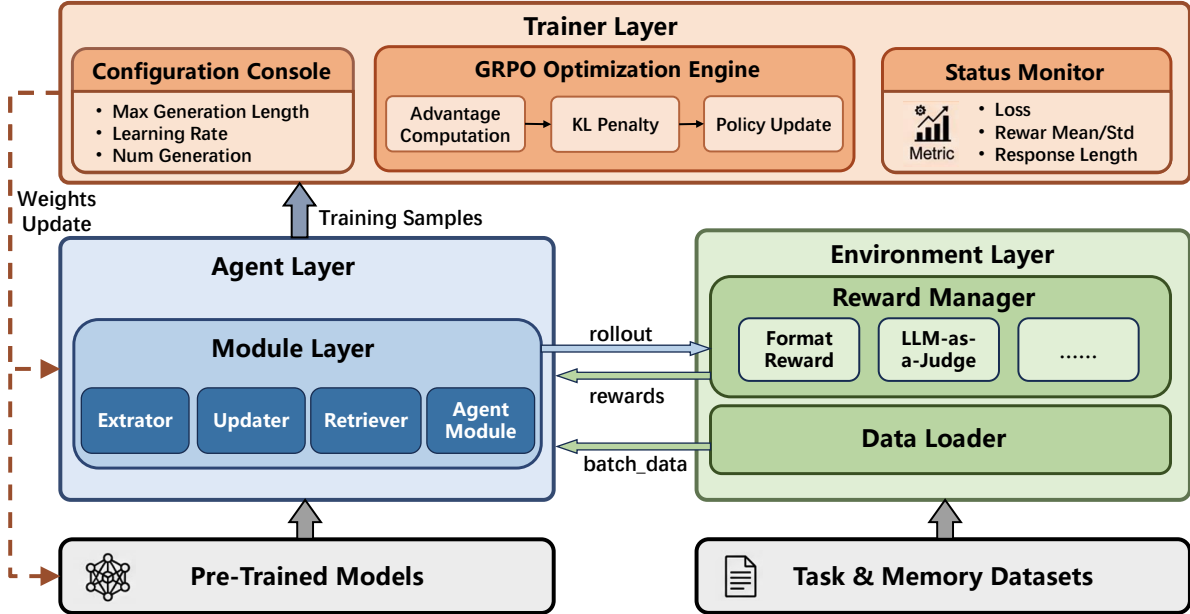


Figure 1 The overall architecture of MemFactory, illustrating the interdependencies among the Modules, Agent, Environment, and Trainer layers.

3.1 Module Layer: Atomic Memory Operations

The *Module Layer* is designed to decompose the complex memory lifecycle into manageable, atomic operations. Inspired by pioneering memory systems such as MemOS [5] and Mem0 [1], we formulate the memory pipeline into three fundamental operations: extraction, update, and retrieval, along with their corresponding module classes. Furthermore, to accommodate end-to-end memory policies seen in works like MemAct [16] and MemAgent [15], we introduce an additional *Agent Module* class within this layer.

Each class of modules supports various internal implementations. However, they all conform to standardized interfaces to facilitate integration with the *Agent Layer* and the *Trainer Layer*. Specifically, a module typically implements three core methods: **generate**, **rollout**, and **inference**. Table 1 outlines the specific roles of these methods.

Table 1 Standard interfaces supported by modules in the *Module Layer*.

Interface	Model	Reward	Notes
generate	Training Model	Deferred	Produces intermediate states; reward is assigned only after the full interaction or trajectory is completed.
rollout	Training Model	Final reward	Produces trajectories that receive the final task-level reward.
inference	Inference Model / APIs	None	Used only for inference and is not trainable.

In the following, we introduce the functions, design, and implementation of these four modules.

3.1.1 Memory Extractor

The primary function of the memory extractor is to parse the raw contexts into structured memory pieces. It extracts facts, experiences, and insights from the historical context and saves them as discrete memory entries. Inspired by Memory-R1 [13], we implement the **NaiveExtractor** class to handle this extraction process.

3.1.2 Memory Updater

After the extractor generates candidate memory entries, the updater compares these candidates with existing memories to produce a memory update plan. Similarly, we draw inspiration from Memory-R1 [13] to design the **NaiveUpdater** class, ensuring that the memory bank remains concise and highly accurate during prolonged user interactions. Specifically, the updater assigns one of four operations to manage the memory states: **ADD** to incorporate new information, **DEL** to remove obsolete or contradicted facts, **UPDATE** to modify existing entries, and **NONE** for memories that do not require changes.

3.1.3 Memory Retriever

The memory retriever is tasked with fetching the most relevant memory from the memory bank to ground the agent’s responses. We implement the **NaiveRetriever** to perform standard semantic-based retrieval. To further enhance retrieval precision, we also provide the **RerankRetriever**. Reranking is a widely adopted post-retrieval technique [3]. In our implementation, the **RerankRetriever** leverages Large Reasoning Models (LRMs) to re-evaluate and rerank the initially retrieved memories, ultimately outputting the refined retrieval results.

3.1.4 Agent Module

For certain highly integrated memory strategies, strictly decoupling the memory pipeline into separate extraction and update phases is unnecessary. For example, in the paradigm proposed by MemAgent [15], the agent synthesizes new memory states directly from the latest context and previous memory states. When

utilizing the memory, it places the entire memory directly into the context without performing any retrieval. To natively support such end-to-end approaches, we define the *Agent Module* class. Thanks to the open-source release of MemAgent [15], we implemented the `RecurrentMemoryModule` following its design. This allows researchers to optimize unified, recurrent memory policies directly within our GRPO training framework.

3.2 *Agent Layer: Module Integration and Policy Execution*

The *Agent Layer* builds upon the *Module layer* to serve as the central policy executor. We adopt a composable, “Lego-like” paradigm to construct robust memory agents. Researchers are highly encouraged to build custom agents tailored to specific tasks by swapping underlying module implementations and experimenting with various module interface combinations.

3.2.1 *Agent Initialization*

During initialization, we use `AutoClasses` from the Transformers library [12] to load pre-trained models. To efficiently handle long-context tasks and reduce memory usage, we integrate FlashAttention-2 [2]. It should be noted that the loaded pre-trained model is the direct target for optimization. After the agent completes its rollout and environment interaction, the *Trainer Layer* directly fine-tunes this specific model.

3.2.2 *Agent Rollout*

The agent’s rollout process is implemented by its constituent modules using their designated interfaces. When a module is used purely for inference and not for training, the agent calls the `inference` interface. This interface supports OpenAI-style APIs and high-throughput inference engines such as vLLM [4].

Conversely, during the training phase, the agent uses the `generate` or `rollout` interfaces. To prevent tensor dimension mismatch during batch generation, we dynamically pad prompt tensors on the left and generated response tensors on the right. We also align the action masks with the sequence lengths to ensure accurate advantage computation in the subsequent reinforcement learning pipeline.

3.2.3 *Agent Inference*

Although MemFactory is primarily a training framework, it inherently supports pure inference. Researchers can instantiate an agent using the same modular composition, configuring each module to use the `inference` interface while bypassing the *Trainer Layer*. This capability provides a practical way to compare different memory workflows and module combinations before committing to computationally expensive RL optimization.

3.3 *Environment Layer: Data Processing and Reward Computation*

The *Environment Layer* serves as the interface between the agent and the task. It processes raw datasets into standardized states and computes reward signals for policy optimization. To address different application scenarios, we draw inspiration from the “long-term vs. short-term memory” taxonomy proposed in the survey *Memory in the Age of AI Agents* [3]. Based on this, we design two primary environments: `MemoryBankEnv` for long-term memory management, and `LongcontextEnv` for short-term, context-based memory.

As a dataloader, this layer converts diverse raw data into unified dictionary states. The main difference between the two environments is how they handle context: `MemoryBankEnv` maintains an explicit, updatable memory bank, whereas `LongcontextEnv` directly processes prolonged dialogue histories. As a reward manager, the environment provides multi-dimensional reward signals, including Format Rewards and LLM-as-a-Judge evaluations.

3.4 *Trainer Layer: Policy Optimization*

The *Trainer Layer* serves as the core optimization engine, dedicated to fine-tuning the agents’ memory management policies. Currently, we implement a trainer based on the Group Relative Policy Optimization (GRPO) algorithm [9]. During the training loop, GRPO samples a group of rollout trajectories for each input and updates the policy based on relative advantages computed from the environment’s rewards. Furthermore,

Table 2 Performance of **MemoryAgent** trained via MemFactory on three test sets (all scores averaged over 4 independent trials, denoted as avg@4).

Model	Setting	eval_50	eval_100	eval_fwe_16384	Average
Qwen3-1.7B	Base checkpoint	0.4727	0.4297	0.0332	0.3118
	+ MemFactory RL	0.5684	0.4863	0.0195	0.3581
Qwen3-4B-Instruct	Base checkpoint	0.6523	0.5645	0.6270	0.6146
	+ MemFactory RL	0.7051	0.6309	0.6426	0.6595

this layer supports comprehensive hyperparameter configurations and natively integrates SwanLab [6]. This allows researchers to monitor training metrics, reward distributions, and generation trajectories in real time, ensuring a transparent and customizable reinforcement learning process.

4 Empirical Study

To demonstrate the modularity of **MemFactory**, we assemble three representative agents inspired by classic memory paradigms: **MemoryR1Agent** based on Memory-R1 [13], **MemoryAgent** based on MemAgent [15], and **MemoryRMMAgent** based on RMM [11]. These three agents are provided out-of-the-box for immediate use.

To validate the effectiveness of the framework, we conduct our empirical study specifically on **MemoryAgent**. The original MemAgent work provides a comprehensive, publicly available dataset for both training and evaluation, making it the ideal choice for our experimental design.

4.1 Experimental Setup

We utilize the training and evaluation datasets provided by the MemAgent release [15]. We train MemAgent-style agents using two open-source Large Language Models (LLMs) as base models: **Qwen3-1.7B** and **Qwen3-4B-Instruct** [14]. The training data are adapted from the original MemAgent training dataset. To improve training efficiency while preserving the long-context nature of the tasks, we simplify the training samples by reducing the context length to approximately one-third of the original, resulting in 50 to 80 documents per sample.

For evaluation, we select three specific test sets from the MemAgent data: two main-task datasets (**eval_50** and **eval_100**) and one out-of-distribution (OOD) dataset (**eval_fwe_16384**). Each model is trained for 250 valid steps. Detailed hyperparameter configurations are available in our open-source repository. To account for generation variance, reported scores are averaged over 4 independent trials per question (avg@4). Notably, the entire training and evaluation pipeline can be executed on a single NVIDIA A800 80GB GPU, demonstrating that MemFactory is highly reproducible even with modest computational resources.

4.2 Results and Analysis

As shown in Table 2, MemFactory consistently improves the average performance for both base models. For **Qwen3-1.7B**, the framework achieves a relative increase of 14.8% in the average score. This is driven primarily by strong gains in the main-task settings, although performance on the OOD benchmark slightly decreases. In contrast, the larger **Qwen3-4B-Instruct** model demonstrates more robust and uniform improvements across all evaluation sets. It achieves a 7.3% relative increase on average, including consistent gains on the OOD benchmark, indicating that the larger model effectively transfers the learned recurrent memory policy to unseen settings.

In general, these results validate two key conclusions. First, MemFactory successfully reproduces and optimizes MemAgent-style recurrent memory policies using standard training datasets. Second, even with a lightweight single-GPU setup and simplified training data, the framework delivers measurable performance improvements on challenging long-context tasks. These findings confirm that MemFactory serves as a concise, user-friendly, and effective infrastructure for empirical Memory-RL research.

5 Conclusion

In this paper, we present **MemFactory**, a modular framework for memory-augmented agents. By abstracting the memory lifecycle into standardized operations across four core layers (*Module*, *Agent*, *Environment*, and *Trainer*), MemFactory provides an accessible foundation that simplifies the design, training, and innovation of memory architectures. Additionally, we assemble out-of-the-box agents inspired by classic works—such as Memory-RL [13], MemAgent [15], and RMM [11]—for immediate use.

Our empirical study confirms that MemFactory serves as a concise, user-friendly, and effective framework for Memory-RL research. While the current version still covers only a limited set of representative paradigms and leaves room for further improvement in training efficiency, these limitations do not diminish its practical value as a unified infrastructure for developing memory-augmented agents. We hope MemFactory can serve as a solid foundation for future research on modular memory systems and reinforcement learning for long-horizon agents.

References

- [1] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL <https://arxiv.org/abs/2504.19413>.
- [2] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL <https://arxiv.org/abs/2307.08691>.
- [3] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL <https://arxiv.org/abs/2512.13564>.
- [4] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL <https://arxiv.org/abs/2309.06180>.
- [5] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. Memos: A memory os for ai system, 2025. URL <https://arxiv.org/abs/2507.03724>.
- [6] Zeyi Lin, Shaohong Chen, Kang Li, Qiushan Jiang, Zirui Cai, Kaifang Ji, and The SwanLab team. SwanLab, 2023. URL <https://github.com/swanhubx/swanlab>.
- [7] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL <https://arxiv.org/abs/2203.02155>.
- [8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL <https://arxiv.org/abs/1707.06347>.
- [9] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [10] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys '25*, page 1279–1297. ACM, March 2025. doi: 10.1145/3689031.3696075. URL <http://dx.doi.org/10.1145/3689031.3696075>.
- [11] Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long T. Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, Anand Iyer, Tianlong Chen, Huan Liu, Chen-Yu Lee, and Tomas Pfister. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents, 2025. URL <https://arxiv.org/abs/2503.08026>.
- [12] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface’s transformers: State-of-the-art natural language processing, 2020. URL <https://arxiv.org/abs/1910.03771>.
- [13] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-rl: Enhancing large language model agents to manage and utilize memories via reinforcement learning, 2026. URL <https://arxiv.org/abs/2508.19828>.

- [14] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- [15] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiyang Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2025. URL <https://arxiv.org/abs/2507.02259>.
- [16] Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks, 2026. URL <https://arxiv.org/abs/2510.12635>.
- [17] Zeyu Zhang, Quanyu Dai, Xu Chen, Rui Li, Zhongyang Li, and Zhenhua Dong. Memengine: A unified and modular library for developing advanced memory of llm-based agents, 2025. URL <https://arxiv.org/abs/2505.02099>.
- [18] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models, 2024. URL <https://arxiv.org/abs/2403.13372>.