Representing Model Weights with Language using Tree Experts

Eliahu Horwitz* Bar Cavia* Jonathan Kahana* Yedid Hoshen
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel
{eliahu.horwitz, bar.cavia, jonathan.kahana, yedid.hoshen}@mail.huji.ac.il

Abstract

The increasing availability of public models begs the question: can we train neural networks that use other networks as input? This paper learns to represent models within a joint space that embeds both model weights and language. However, machine learning on model weights is challenging as model weights often exhibit significant variation unrelated to the models' semantic properties (nuisance variation). We identify a key property of real-world models: most public models belong to a small set of Model Trees, where all models within a tree are fine-tuned from a common ancestor (e.g., a foundation model). Importantly, we find that within each tree there is less nuisance variation between models. For example, while classifying models according to their training dataset generally requires complex architectures, in our case, even a linear classifier trained on a single layer is often effective. While effective, linear layers are computationally expensive as model weights are very high dimensional. To address this, we introduce Probing Experts (ProbeX), a theoretically motivated, lightweight probing method. Notably, ProbeX is the first probing method designed to learn from the weights of just a single model layer. We also construct and release a dataset that simulates the structure of public model repositories. Our results show that ProbeX can effectively map the weights of large models into a shared weight-language embedding space. Furthermore, we demonstrate the impressive generalization of our method, achieving zero-shot model classification and retrieval.

1. Introduction

In recent years, the number of publicly available neural network models has skyrocketed, with over one million models now hosted on Hugging Face. This growth raises a new research question: can we use the neural networks themselves as inputs for training new models? The emerging field of weight space learning [22, 28, 36] studies how to learn metanetworks, neural networks that take the weights of other neural networks as inputs (see Fig. 2). Previous works learned metanetworks that predict training data attributes [21, 30], model performance [22], and even generate new models [6, 24]. In this work, we explore metanetworks that embed model weights into a joint space with language, enabling tasks such as zero-shot model classification and retrieval.

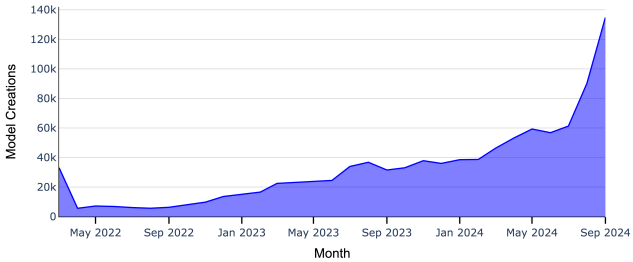

Figure 1. ***Recent Growth in Hugging Face Models:*** The increasing availability of public models, with over one million now hosted on Hugging Face, raises the question: can we train neural networks that use other networks as input?

In general, the core challenge of any representation learning algorithm is to include only the semantic information while excluding non-semantic (nuisance) factors. Model weights, in particular, pose unique challenges. As model weights parameterize the relationship between input data and the target task, they capture semantic information, e.g., details about the training distribution and the target task. However, alongside this semantic information, model weights also include nuisance factors. One notable nuisance factor in model weights that has gained considerable attention is neuron permutation [13]. A substantial body of research has focused on developing methods that are invariant to these permutations, using specialized architectures [19, 21–23, 36] or carefully designed data augmentations [14, 28–30].

In this paper, we highlight another nuisance attribute of model weights, known as Model Trees [16]. A Model Tree describes a set of models that share a common ancestor, where each model has either been fine-tuned from or used

*Equal contribution

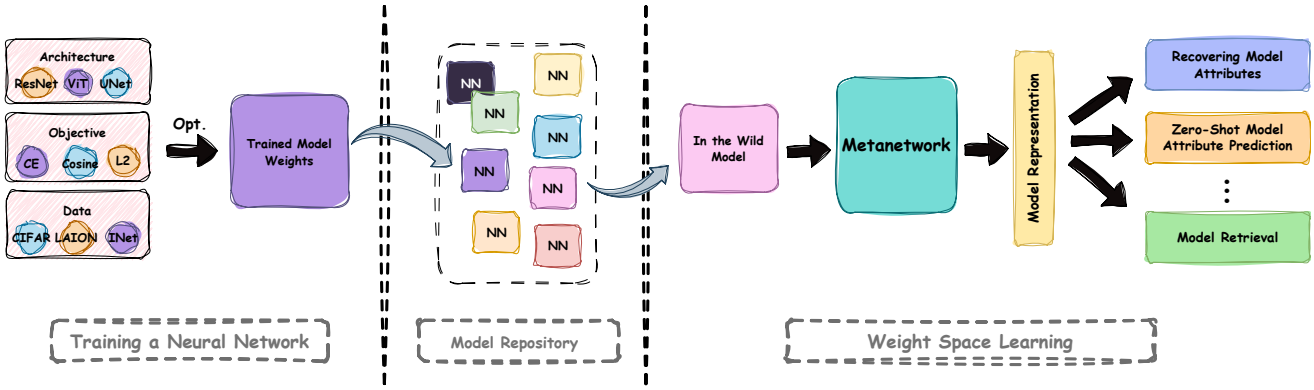

Figure 2. ***Weight Space Learning:*** *Left.* Model weights are a direct product of the objective, optimization procedure and training data. *Center.* Users upload models to public repositories, e.g., Hugging Face, which are often disorganized and lack metadata on model functionality, pre-training checkpoints, and training data. *Right.* Weight space learning treats model repositories as datasets where each model is a data point. It studies how to design metanetworks, a special kind of neural networks that process the weights of other models. Here, we explore metanetworks that embed model weights into a joint space with language, enabling tasks such as zero-shot model classification and retrieval

as pre-training for another model within the same tree. For example, the Llama3 Model Tree [7] includes all models that have been fine-tuned from Llama3 or any of its descendants. We hypothesize that learning on models from the same Model Tree is a much simpler task than learning on models from many different Model Trees. This setting is practical as most models are members of a small set of trees. To illustrate, we analyze the Hugging Face model hub, and find that less than 20 Model Trees already account for 50% of all models, with each tree containing over 1,000 models [1]. We validate our hypothesis in a motivating experiment, where we compare between learning on intra-tree and inter-tree model populations. While a simple linear model performs well on models within the same tree, it fails when applied to a similar population that came from many different trees. It also uses an infeasible number of parameters.

To address the limitations of the linear baseline, we propose a Mixture-of-Experts (MoE) approach that extends within-tree learning to handle multiple trees in a diverse model population. Our approach consists of two key components: i) a model router and ii) ***Prob****ing* ***eX****perts* (ProbeX). The model router identifies the tree to which a model belongs and directs it to the dedicated tree expert. While linear classifiers perform well within individual trees, they become impractical for large models due to the high dimensionality of model weights, often requiring hundreds of millions of parameters. To address this, we introduce ProbeX, a probing-based expert architecture that is significantly more efficient. Unlike conventional probing methods, ProbeX operates effectively on a single layer. Notably, ProbeX can learn on weights of models containing hundreds of millions of parameters, requiring less than 10 minutes to train. We also establish a theoretical connection between ProbeX and Tucker tensor decomposition [32]. Furthermore, we demonstrate the effectiveness of ProbeX by aligning its representation with a CLIP text encoder, creating a joint embedding space of model weights and language.

To evaluate our method, we present three datasets that simulate real-world model repositories, comprising over 12,000 models across five disjoint Model Trees. These datasets include both generative and discriminative models spanning multiple architectures and tasks. We demonstrate the efficacy of ProbeX on various tasks, such as predicting the presence of specific classes within a model's training data and identifying the personalization concept a model was fine-tuned on. To assess generalization, we align the ProbeX weight representations of models fine-tuned from Stable Diffusion with a CLIP text encoder. Using the resulting shared embedding space, we showcase the generalization of these aligned representations on tasks such as model one-class classification and retrieval. We also introduce the task of zero-shot model classification, where we successfully classify models from unseen classes based on a text prompt.

To summarize, our main contributions are:

1. Identifying that most public models belong to a small set of Model Trees and suggesting that learning within trees can increase accuracy.
2. Introducing a Mixture-of-Experts (MoE) approach consisting of lightweight, theoretically motivated Probing Experts (ProbeX) for weight-space learning from diverse model populations.
3. Proposing the alignment of model weights with language representation and introducing the task of zero-shot model weight classification, along with releasing an evaluation dataset.

[1]These figures are from the official hub-stats dataset, though some sampling bias may exist due to incomplete information.

2. Related Works

While neural networks are fully defined by their weights, little research has explored using weights with machine learning methods. Eilertsen et al. [8], Unterthiner et al. [33] were among the first to systematically analyze model weights of diverse populations to predict undocumented properties like the training dataset or generalization error. Some works aim to learn general representations [14, 28–30] for multiple properties, while others incorporate specific priors [19, 21–23, 36] to directly predict the property. A major challenge with model weights is the presence of many parameter space symmetries [13]. For instance, permuting neurons in hidden layers of an MLP doesn't change the network output. Thus, neural networks designed to take weights as inputs must account for these symmetries. In order to avoid the issue of weight symmetries, recent methods [14, 19] propose using *probing*. In this approach, a set of probes are optimized to serve as inputs to the model and the outputs act as the model representation. However, until now, this was limited to passing the probes through the entire model and did not apply to single layers.

Other weight-space learning applications include generating model weights [1, 6, 9, 10, 24, 31], predicting dataset size [27], and recovering the weights of unpublished models. This recovery can occur through an API to retrieve a single layer [3] or by utilizing multiple fine-tuned variants of the same foundation model to reconstruct the entire model [15].

3. Motivation

3.1. The Challenge

Representation learning aims to compress data points into vectors that have i) low-dimension, and ii) keep as much semantic information as possible. In recent years, effective representation learning methods for images, text, and audio have been developed by identifying and eliminating key non-semantic (nuisance) factors. However, representation learning for model weights is still in its infancy, and the key nuisance factors remain unclear. Many approaches focused on neuron permutations [19, 22, 35] as the core nuisance factor. However, permutations are not likely to describe all nuisance variation, particularly as neurons and layers can have different roles in different models and architectures. This paper focuses on another nuisance factor, Model Trees.

3.2. Seeing the Forest by Seeing the Trees

Background: Model Trees. Following Horwitz et al. [16], we represent model populations as a *Model Graph* composed of disjoint directed *Model Trees*. In this graph, each node represents a model, with directed edges connecting a model to those directly fine-tuned from it. Since each model has at most one parent, the graph forms a set of non-overlapping trees. Importantly, our approach does not require knowledge of the internal structure of these Model Trees; it is sufficient to know whether a model belongs to a particular tree.

Table 1. ***Intra vs. Inter Tree Model Populations:*** We investigate two model populations, i) models from different trees (F) and ii) models from the same tree (T). We compare the accuracy of identifying the classes that the models were trained on. As hypothesized, learning from models within the same Model Tree results in almost perfect accuracy while learning from many disjoint Model Trees results in random accuracy

	Random	**Inter-Tree** (F)	**Intra-Tree** (T)
Acc. ↑	0.5	0.502	0.940

Divide-and-Conquer. Current weight-space methods generally rely on a single metanetwork to learn from a diverse model population spanning multiple Model Trees. We hypothesize that Model Trees are a significant nuisance factor in model weights, complicating the learning of meaningful representations. We expect that dividing the population into distinct groups based on Model Trees and learning within each tree, can greatly simplify model representation learning.

Nuisance factor: Tree Membership. To illustrate the benefits of Model Trees, we conduct a simple experiment. We randomly choose 50 classes from CIFAR100 (without replacement), denoting this dataset A. By randomly selecting 25 of the remaining classes, we create dataset B and pre-train a classifier model on it for a *single epoch*. The experiment compares two similar model populations T and F, each containing 500 ResNet9 models. For each model, we randomly selected 25 classes from A, and learned to classify each image into the correct class out of 25. The models in T and F differ in one aspect only, models in F (model forest) were initialized randomly, while the models in T (Model Tree) were initialized from the model pre-trained on B. Therefore, models in T all lie within the same tree, while those in F lie in different, disjoint trees.

The task is to take each model as input and predict which classes from A it was trained on. We thus train a metanetwork on all models in T, and another one for all models in F. Both metanetworks have the same simple architecture, a *single linear layer* which takes as input a single weight matrix of the input model. The results are in Tab. 1. In line with our hypothesis, we observe an extreme performance gap between the two settings. While learning on models within the same tree achieved excellent results (0.940), learning on models from heterogeneous trees achieved near random accuracy (0.502).

This simple experiment demonstrates that even a shared pre-training of just a single epoch can be enough to eliminate significant non-semantic variations in neural weights. Moreover, it shows that models within a single tree have a simple, even linear, mapping from weights to semantics.

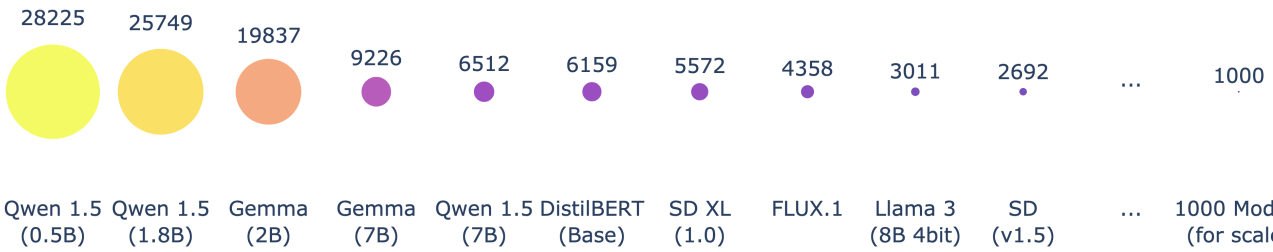

Figure 3. ***Largest Model Trees on Hugging Face:*** We show the sizes of the ten largest Model Trees of Hugging Face. Our insight is that learning a Mixture-of-Experts for these trees greatly simplifies learning from model weights. This setting is practical since as shown here, most real-world, public models belong to a small number of large Model Trees

A Few Large Trees Dominate the Landscape. While we demonstrated that learning within trees is effective, this insight is only useful if real models lie in a small number of trees. To explore the broader landscape, we analyzed approximately $250k$ models from the Hugging Face model hub[2]. Our analysis reveals that most public models belong to a small number of large Model Trees. For instance, less than 20 Model Trees account for 50% of the total population. Furthermore, 196 Model Trees have at least 100 models, collectively covering over 70% of all models. This breakdown is illustrated in Fig. 3. We conclude that learning metanetworks on Model Trees is both effective and practical.

4. Mixture of Experts

Motivated by Sec. 3, we propose a mixture-of-experts (MoE) approach for classifying heterogeneous model populations. This approach has two components: i) a routing function R, that maps each model $\mathcal{F}$, to its relevant expert. This effectively divides the model population into non-overlapping subsets. ii) A set of experts that classify model weights within their subset of expertise.

Notation. Consider a model $\mathcal{F}$ with s layers and denote the dimension of each layer by d_H and d_W. Let $X^{(1)}, \ldots, X^{(s)}$ denote the weight matrices of the layers. For brevity, we omit the layer index superscript in the notation, although in practice, we apply the described method to each layer of the model individually. In case the model uses LoRA, we can multiply the decomposed matrices $X = BA$ and work with the full matrix.

4.1. Routing

Differently from recent MoE methods [34] that learn the router and experts end-to-end, we decouple the two; first learning the routing function and then the experts. For the routing function, we opt for a fast and simple clustering algorithm. Specifically, we cluster the set of models using hierarchical clustering. After completing the clustering step, we compute the center of each cluster $\hat{X}_1, \hat{X}_2, \cdots, \hat{X}_K$. The routing function assigns models to the nearest cluster in ℓ_1:

$$R(X) = arg\min_{k=1}^{K} \|X - \hat{X}_k\|_1 \tag{1}$$

4.2. Problem Formulation and Dense Experts

We train a dedicated expert model for each cluster. The prediction task is mapping a model weight matrix $X \in \mathbb{R}^{d_W \times d_H}$ to an output vector $\mathbf{y} \in \mathbb{R}^{d_Y}$. The output vector can be logits or a representation.

Dense Expert. A simple choice for the expert architecture is a linear function. As the input is a 2D weight matrix $X \in \mathbb{R}^{d_W \times d_H}$, the linear function is a 3D tensor $W \in \mathbb{R}^{d_H \times d_W \times d_Y}$. The output of an expert is simply:

$$y_k = \sum_{ij} W_{ijk} X_{ij} \tag{2}$$

Although such experts can achieve surprisingly good performance, they often have an extremely high number of parameters resulting in two main limitations: i) overfitting due to overparameterization ii) high memory cost due to the extreme dimension of the weight matrix, in our experiments they often require over a billion parameters.

4.3. Probing

Probing-based methods [14, 19] have emerged as a promising approach for processing neural networks. Instead of learning a metanetwork that directly processes the weights of the target model, probing methods pass *probes* (input vectors) through the model and use the model outputs to represent the model. As each probe provides partial information about the model, fusing information from a diverse set of probes leads to more informative representations. Passing probes through the model is usually cheaper than passing all network weights through a huge metanetwork, probing is therefore more compute and parameter efficient than the alternatives.

[2]We only consider models that include information about their pretraining model.

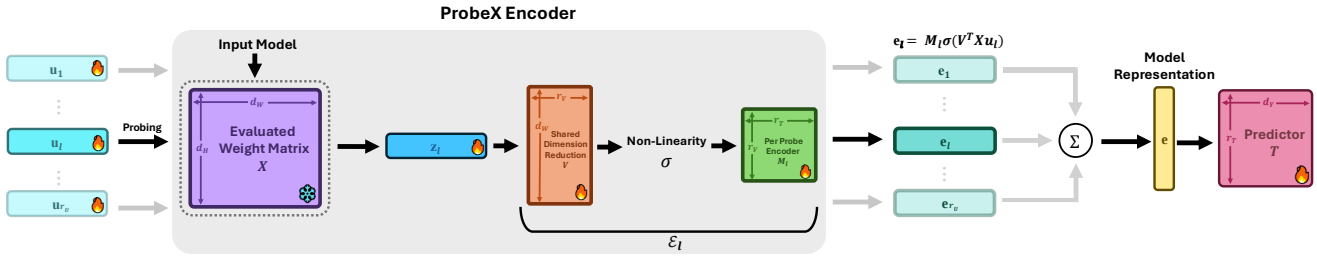

Figure 4. ***ProbeX Overview.*** An illustration of ProbeX, a lightweight probing-based metanetwork for weight space learning. We train ProbeX on a dataset of neural network models, where the input is the weight matrix of a single layer from the input model. (A) During training, we learn a set of probes $\mathbf{u}_1.\mathbf{u}_2, \cdots, \mathbf{u}_{r_U}$. (B) Each probe passes through the input weight matrix X. A projection matrix V, shared between all probes, reduces the dimension of the output. This is followed by a non-linear activation. (C) A per-probe encoder matrix M_l maps the activations to the probe representation $\mathbf{e}_l$. (D) We sum the probe representations to obtain the model representation $\mathbf{e}$. A predictor module maps the representation to the task output $\mathbf{y}$. We optimize all components of ProbeX end-to-end

Traditionally, probing requires passing probes through the entire model, circumventing many nuisance attributes of model weights (e.g., neuron permutations) by only using model inputs and outputs. However, as working within Model Trees already neutralizes one of the key nuisance factors (see Sec. 3), we expect probing to succeed even when applied just to hidden layers of the model. Our method adapts the idea of probing to operate directly on individual weight matrices.

Formally, let $f_X : \mathbb{R}^{d_W} \to \mathbb{R}^{d_H}$ be the function that we wish to analyze. To do so, we select a set of probes $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{r_U} \in \mathbb{R}^{d_W}$, passing each probe $\mathbf{u}_l$ through the function f_X, and recording its responses $\mathbf{z}_l = f_X(\mathbf{u}_l) \in \mathbb{R}^{d_H}$. A per-probe encoder $\mathcal{E}_l$ then maps the response $\mathbf{z}_l$ of each probe to a probe representation $\mathbf{e}_l \in \mathbb{R}^{d_V}$. The final representation of the model f_X is $\mathbf{e}$, which is the sum of representations $\mathbf{e}_l$ of the individual probes:

$$\mathbf{e} = \sum_l \mathcal{E}_l(f_X(\mathbf{u}_l)) \tag{3}$$

A prediction head $\mathcal{T} : \mathbb{R}^{d_V} \to \mathbb{R}^{d_Y}$, maps the model representation $\mathbf{e}$ to the final prediction:

$$\mathbf{y} = \mathcal{T}(\mathbf{e}) \tag{4}$$

In practice, we learn all the probes, the probe encoders, and the prediction head end-to-end. Note that we learn the probe values directly (via latent optimization [2]), and a different probe encoder $\mathcal{E}_l$ for each probe $\mathbf{u}_l$.

The probing network is very general. Indeed, its linear version is as expressive as the dense expert.

Proposition 1. *Assume we implement $\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_{r_U}$ linear operations, i.e., matrices, and use a sufficient number of probes. The dense expert (Eq. 2) and probing network (Eq. 4) have identical expressivity.*

Proof. App. A.1 □

4.4. Single Layer Probing Experts

As probing is effective and efficient, we want to adapt probing for the single layer case. Here, the function that we probe is the weights of a single layer of a model, $f_X = X$. If the weights are a higher dimensional tensor, we reshape it as a 2D matrix. We initially consider the case where the probing encoders and prediction head are all linear functions. We immediately face a challenge, there are r_U probes and each one has a dedicated encoder parameterized by a large matrix. This requires many parameters.

We therefore factorize each probing encoder as the product of 2 matrices. A dimensionality reduction matrix $V \in \mathbb{R}^{d_W \times r_V}$ that projects the high dimensional outputs of the input matrix $X \in \mathbb{R}^{d_W \times d_H}$ to a much lower dimension r_V. To reduce the parameter count, we share the matrix V between all encoders. The second matrix is a lower dimensional matrix $M[l] \in \mathbb{R}^{r_V \times r_T}$ (in practice, we use $r_T = r_V$), which is different for every probe encoder. Since only the smaller matrix $M[l]$ is specific to each probe, while all probes share the larger matrix V, this decomposition significantly reduces the number of parameters. Finally, the per-probe encoder is given by:

$$\mathcal{E}_l(\mathbf{z}_l) = M_l V^T \mathbf{z}_l \tag{5}$$

The prediction head $\mathcal{T}$ is simply the matrix $T \in \mathbb{R}^{r_T \times d_Y}$. Putting everything together, our entire single layer probing expert that we call ProbeX (**Prob**ing **eX**pert) is given by:

$$\mathbf{y} = T \sum_l M_l V^T X^T \mathbf{u}_l \tag{6}$$

While we justified using ProbeX intuitively, we can prove that the linear ProbeX has the same expressivity as using the Tucker low-rank tensor decomposition on the weight tensor of the dense expert (Eq. 2).

Proposition 2. *The Tucker low-rank tensor decomposition is defined as:*

$$W = \sum_{nml} M_{nml} \cdot \boldsymbol{t}_n \otimes \boldsymbol{v}_m \otimes \boldsymbol{u}_l \tag{7}$$

The linear ProbeX in Eq. 6 has identical expressivity as using the dense predictor in Eq. 2, with a weight tensor obeying the Tucker decomposition.

Proof. App. A.2 □

4.4.1 Non-Linear Single Layer Probing

Although Proposition 1 showed that linear ProbeX has identical expressivity to the dense expert, ideally we would like ProbeX to be more expressive than a linear function. We therefore suggest a simple modification to the probe encoder, adding a non-linearity σ between the two matrices. This change effectively transforms ProbeX into a factorized one hidden layer neural network:

$$\mathcal{E}_l(\mathbf{z}_l) = M_l \sigma(V^T \mathbf{z}_l) \tag{8}$$

We present an overview of ProbeX in Fig. 4. Unless otherwise stated, in all our experiments we chose σ to be the ReLU function. Note that while this paper uses a probe encoder with a single hidden layer, deeper architectures can also be utilized.

4.4.2 Training

For classification tasks, we use ProbeX to map model weights to logits and optimize it using cross entropy. For representation alignment, we use a contrastive loss. In all cases, we optimize the weights of $V, \mathbf{u}_1, \cdots, , \mathbf{u}_r, M_1, \cdots, M_r, T$ end-to-end. Note that while our formulation describes the case of a single layer, there is no loss of generality. Given multiple layers, we extract a representation from each layer using ProbeX. We then concatenate them and map them to the output y using a matrix T, training everything end-to-end.

5. Model Jungle Dataset

We construct the *Model Jungle* (Model-J) dataset for realistic evaluation of weight space learning methods. Importantly, our dataset simulates the structure of model repositories and contains large models that belong to a small set of *disjoint* Model Trees that vary in architecture, task, and size. Each fine-tuned model uses a set of randomly sampled hyperparameters. Collectively, Model-J contains over 12,000 models, divided into two primary splits. See Tab. 2 for an overview:

Discriminative. We fine-tune 2,500 models for image classification. These models belong to one of three Model Trees: i) *Sup. ViT.* A supervised pre-trained ViT-B/16 [5] (1,000 models), ii) *MAE.* A masked auto-encoder (MAE) [12] with a ViT-B/16 architecture (500 models), and iii) *ResNet.* A supervised pre-trained ResNet-101 [11] (1,000 models). Each of our models is fine-tuned (using "vanilla" full fine-tuning) to classify images from a random subset of 50 out of the 100 CIFAR100 classes.

Table 2. ***Model Jungle Dataset Summary.*** We train over 12,000 models, covering different architectures, tasks and model sizes. Each model uses randomly sampled hyper parameters

Name	Task	Fine-tuning	Split Size	# Classes
Sup. ViT	Full FT	Att. classification	1000	50/100
MAE	Full FT	Low-resource Att. classification	500	50/100
ResNet	Full FT	Att. classification	1000	50/100
SD_{200}	LoRA	Fine-grained	5000	200
SD_{1k}	LoRA	Low resource	5000	1000

Generative. We fine-tune $10,000$ personalized models Ruiz et al. [26]. All of the models in this split belong to the Stable Diffusion [25] Model Tree. Each model was fine-tuned on $5-10$ images, randomly sampled without replacement, originating from the same ImageNet [4] class. This split consists of 2 variants each with $5,000$ models: i) SD_{200}. A fine-grained variant consisting of 25 models per class for the first 200 ImageNet classes (mostly different animal breeds). ii) SD_{1k}. A low resource split containing 5 models per class for all $1,000$ ImageNet classes. To save compute and storage, we follow common practice and use LoRA [17] fine-tuning. We set aside a test subset of random *holdout* classes, $30 \in SD_{200}$ and $150 \in SD_{1k}$.

6. Experiments

6.1. Experimental Setting

We use Model-J presented in Sec. 5 with a $70/10/20$ train, val, test split. We train all of our methods for 500 epochs and choose the best epoch according to the validation set. As our method uses a single input layer, and since the results vary significantly between layers, we train ProbeX on each layer of the model and report the best layer according to the validation set. We set the number of probes to be $r_U = 0.1 \cdot d_W$ and the dimension of output dimension of the encoder $r_V = r_T = 0.1 \cdot d_H$. With this choice of hyperparameters, training ProbeX on a single layer takes under 10 minutes on a single GPU.

Baselines. Many state-of-the-art methods do not scale to large models with hundreds of millions of parameters. We therefore compare to the following baselines: i) *StatNN [33].* This permutation-invariant baseline extracts 7 simple statistics (e.g. mean, variance, different quantiles.) for the weights and biases of each layer. It then trains a gradient

Table 3. ***Mixture of Experts Results.*** In this challenging task, each model is trained on 50 randomly selected CIFAR100 classes (out of a total of 100). We train ProbeX tree experts to predict which of the 100 classes was used in training. While the dense expert performs moderately well, ProbeX achieves better accuracy with two orders of magnitude fewer parameters, highlighting its effectiveness for fine-grained tasks

	ResNet		MAE		Sup. ViT		MoE	
Method	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓
Random	0.5	-	0.5	-	0.5	-	0.5	-
StatNN	0.622	-	0.501	-	0.519	-	0.547	-
Dense	0.693	105m (×286)	0.609	59m (×85)	0.660	59m (×85)	0.654	223m (×111)
ProbeX	**0.800**	366k	**0.653**	694k	**0.867**	694k	**0.773**	2m

boosted tree on the concatenation of the statistics from all layers. ii) *Dense Expert.* Training a single linear layer on the flattened raw weights. Note that this baseline results in extremely large classifiers. For instance, a single layer of Stable often has $1.6m$ parameters. Therefore, even a single linear layer trained to classify SD_{1k} would consist of $1.4B$ parameters, twice the number of parameters of the entire Stable Diffusion model.

Metrics. We use accuracy as the evaluation metric. For ProbeX and the dense baseline, we also report the parameter count of the model. Finally, for one-class classification we use the area under the ROC curve.

6.2. Training Dataset Class Prediction for Discriminative Models

In this experiment, we train a metanetwork to predict the training dataset classes for models in the discriminative split of Model-J. As each model was trained on 50 randomly selected classes out of 100, we treat the output as a set of 100 binary labels indicating whether each class was included in the model's fine-tuning data. A random prediction strategy yields 50% accuracy on this task. Concretely, we train Eq. 5 with 100 jointly optimized binary classification heads. This is an especially difficult task as each class is only 2% of the training data and so its signature is likely to be quite weak.

This task is quite practical; consider a model repository such as Hugging Face, which currently relies on the model metadata (e.g., model card) when searching for a model. These model cards are often poorly documented and missing details regarding the precise classes the model was trained on. Therefore, using model cards to understand what classes a model can classify is unlikely to be effective. Conversely, our metanetwork would allow users to search for suitable models effectively.

We train ProbeX for each discriminative Model Trees, the results are shown in Tab. 3. We can see that while the dense expert achieves some level of accuracy. ProbeX performs better with roughly two orders of magnitude fewer parameters. This demonstrates the power of our method for very challenging, fine-grained tasks.

6.3. Aligning Weight Representations to Text Embedding

In this section, we test the ability of our method to align model weight representations to text embeddings. We learn a mapping between layer weights of models in the generative split of Model-J and the CLIP embedding of the concept. This effectively creates a shared space between text prompts and weight matrices of models from the same Model Tree. We evaluate the aligned representation on a variety of tasks, and show that it generalizes well. Strikingly, we believe ProbeX is the first method that learns weight representations with zero-shot capabilities.

Representation Alignment. We train ProbeX to map representations of model weights to pre-trained text embeddings. This mapping is supervised, as we have paired data consisting of model weights and the text embedding on the class name of their fine-tuning dataset. Our dataset comprises of both SD_{200} and SD_{1k}, as described in Sec. 5. We train the representation similarly to the way that CLIP was trained, i.e., we compute the representation of the input layer using ProbeX (or one of the baselines). We then compute the cosine similarity to the embedding of each of the potentially target classes. The optimization objective is that the cosine similarity to the ground truth class will be high, and all other classes lower.

6.3.1 Text-Weights Zero-Shot Capabilities

We begin by testing the zero-shot capabilities of our aligned representation on the held-out splits of Model-J (see Sec. 5). Specifically, given a weights-to-text embedding mapping function, we compute the similarity between the model representation and all possible classes. The similarity score is calculated for all held-out classes (unseen during ProbeX's training), and the model is labeled with the class that has the highest matching score. See Fig. 5 for an overview of this setting. We perform a similar experiment for in distribution data (concepts seen in training time), i.e., a standard classification setting. In Tab. 4, we show the top-1 accuracy of our method compared to the dense expert. Importantly, our method generalizes not only to unseen models trained

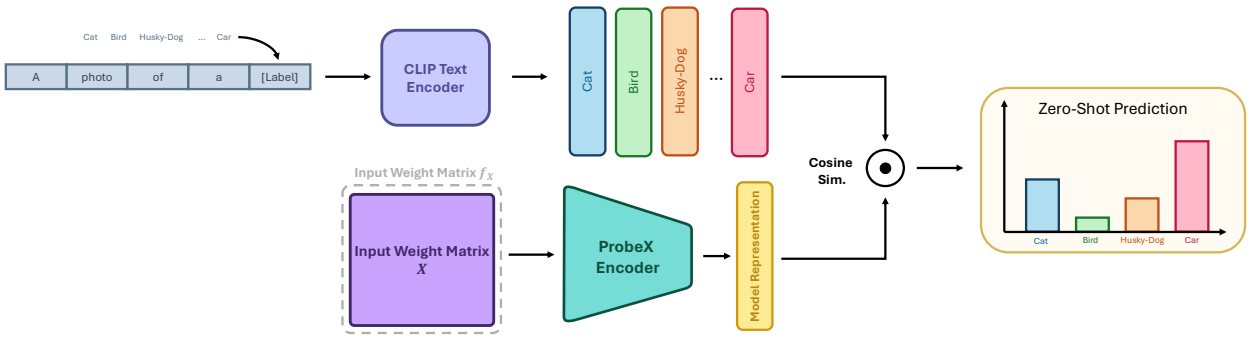

Figure 5. ***Zero-Shot Inference Overview.*** We align model weights with a pre-trained text encoder, creating a shared text-weight embedding space for zero-shot model classification. For each class, we extract the CLIP text embedding of the class name and use ProbeX to encode the weight matrix X into the shared space. Classification follows by selecting the text prompt nearest to the model weight representation e using cosine similarity. This creates a CLIP-like zero-shot setting, where model weights from unseen classes are classified via text prompts

on the same classes (i.e., in distribution samples) but also to entirely new object categories. ProbeX can detect classes unseen during training with over 50% accuracy when there are 150 held-out classes, and nearly 90% accuracy with 30 held-out classes. This demonstrates that ProbeX successfully aligns model representations with CLIP's, generalizing effectively to new concepts.

6.3.2 Unsupervised Downstream Tasks

Motivated by the promising zero-shot results, we explore our model representations in other downstream tasks. In all experiments, we use our trained ProbeX model to extract model representations.

Model Retrieval. Given a model, we search for the models that were trained on the most similar datasets. We use the cosine distance between the ProbeX representations of the input model as the similarity metric. Fig. 6 shows the 3 nearest-neighbors for 3 query models, each fine-tuned using a different dataset. For visualization purposes, we present a model by showing 2 images from its training set. We see that indeed query models are related to models within the same concept, showing our representation captures highly semantic attributes even in fine-grained cases. For instance, while SD_{200} contains many different dog and cat breed classes, our retrieval accurately returns the breed that the query model was trained on.

kNN classification. In a related but different task, we show that kNN can correctly classify the training dataset class. We randomly split our samples into train and test sets. For each test model, we label it by its nearest class. The score is the average kNN distances between the ProbeX representation of the test model and the training models from this class. Tab. 6 compares our learned representation with simply using raw weights, and shows that our representation performs much better.

Table 4. ***Aligned Weight-Text Representation Results:*** We report the text guided classification accuracy on both the in distribution and holdout splits. Our method generalizes not only to unseen models trained on the same classes (in-distribution samples) but also to entirely new object categories. ProbeX detects classes unseen during training with over 50% accuracy when 150 held-out classes are used, and nearly 90% accuracy when there are 30 held-out classes. This demonstrates that ProbeX successfully aligns model representations with CLIP's, generalizing effectively to new concepts. In the 1k case, where each class has only 5 training samples, the dense method overfits the in-distribution classes but fails to generalize compared to ProbeX

	Method	In Dist. ↑ Acc.	Zero-shot. ↑ Acc.	# Params ↓
SD_{200}	Random	0.006	0.033	-
	$StatNN_{MLP}$	0.018	0.075	2.6m
	$StatNN_{Linear}$	0.030	0.147	689k
	Dense	0.801	0.706	32m (×13)
	ProbeX	**0.973**	**0.898**	2.5m
SD_{1k}	Random	0.001	0.006	-
	$StatNN_{MLP}$	0.001	0.029	2.6m
	$StatNN_{Linear}$	0.01	0.045	689k
	Dense	**0.382**	0.343	210m (×84)
	ProbeX	0.296	**0.505**	2.5m

One-Class-Classification. We further examine the effectiveness of our representations at discriminating in-distribution and out-of-distribution classes. We designated each of our classes as "normal" and computed the average kNN distance between all test models and the training set of the normal class. In Tab. 6 we report the mean ROC AUC score, using the kNN similarity score for separating the normal samples from the anomalous ones. Indeed, the results show that our method can detect out-of-distribution models much more accurately than other methods.

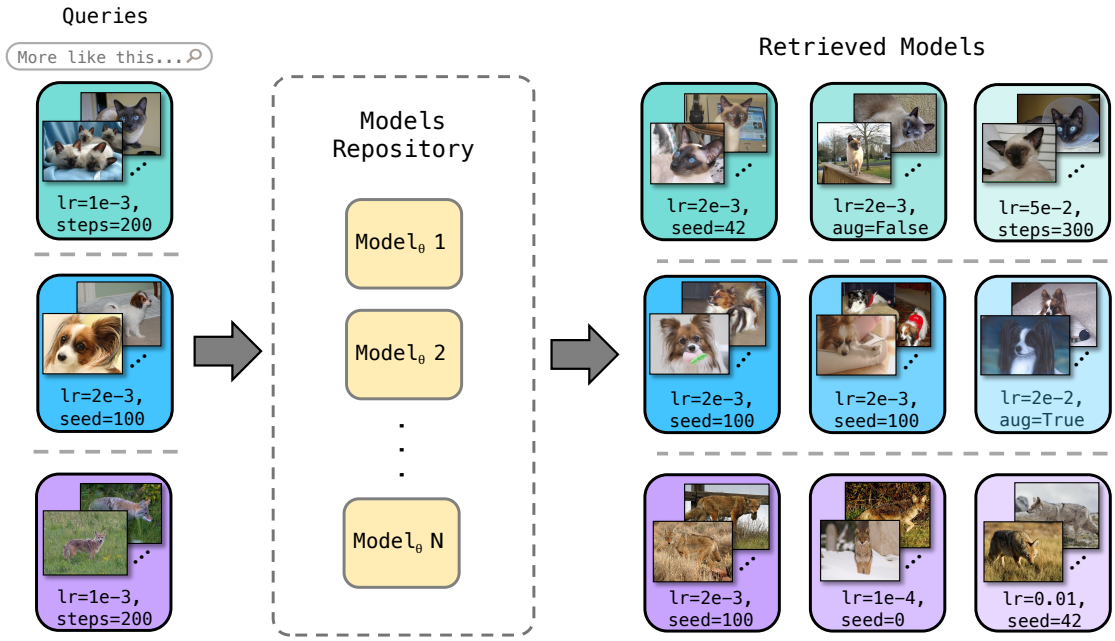

Figure 6. ***Qualitative Retrieval Results:*** For each query model, we search for the models trained on the most similar concepts, measuring similarity by cosine distance between ProbeX representations. We present the three nearest neighbors for three query models, each fine-tuned on a different concept. For visualization, we show two of the images used to train the model. Indeed, the retrieved models have similar concepts to the query indicating our representations accurately align with fine-grained, semantic text embeddings. For example, in SD_{200} there are models of many breeds, but the retrieved models still succeed in having the same breed as the query

Table 5. ***Activation Ablation on*** SD_{200}***:*** Using ReLU slightly improves in-distribution classification, but significantly improves zero-shot classification. This suggests that while linear ProbeX represents training samples well, ReLU enhances generalization. The best layer contained the same number of parameters in all cases

	In Dist. Acc. ↑	**Zero-shot Acc ↑.**
No ReLU	0.953	0.564
ReLU	0.973	**0.898**

Table 6. ***kNN and OCC Results on a Single Expert:*** We report mean accuracy for kNN and mean AUC (mAUC) for OCC, averaging across all 30 holdout classes of SD_{200}. ProbeX achieves the highest results for both. Interestingly, using many neighbors is beneficial for OCC but not for supervised kNN classification

	kNN (Acc. ↑)			**OCC (mAUC ↑)**		
k	**Raw**	**Dense**	**ProbeX**	**Raw**	**Dense**	**ProbeX**
1	0.833	0.502	**0.913**	0.501	0.561	**0.698**
2	0.389	0.525	**0.933**	0.502	0.573	**0.702**
5	0.417	0.477	**0.872**	0.504	0.610	**0.720**
All	0.033	0.294	**0.428**	0.507	0.681	**0.792**

7. Ablations

Activation Function. We ablate the need for the non-linear version of ProbeX using the SD_{200} dataset; results are shown in Tab. 5. Interestingly, while the use of ReLU slightly improves in-distribution classification performance (0.953 without ReLU vs. 0.973 with ReLU), the main benefit is in zero-shot classification (0.564 without ReLU vs. 0.898 with ReLU). This significant difference in zero-shot performance suggests that, while the linear version of ProbeX can effectively represent the training samples, generalizing to other classes requires a deeper model.

Text Encoder. We ablate the sensitivity of our method to the precise language encoder used. We test CLIP, OPEN-CLIP [18], and BLIP2 [20] in our zero-shot experiment. The results in Tab. 7, suggest that while CLIP performs best (which is expected, as Stable Diffusion was trained using CLIP), our approach remains effective across different text encoders. This shows robustness to the choice of text backbone.

Dataset Size. We examined the effect of dataset size on accuracy. This ablation tested both the dense expert and ProbeX on the supervised ViT Model Tree when 350 and 700 training models. The results in Tab. 8 show that the dense method only improves slightly while ProbeX improves by over 10%, this indicates that ProbeX can use extra samples more effectively but may be more prone to overfitting.

Table 7. ***Text-Encoder Ablation on*** SD_{200}***:*** We ablate the sensitivity of the representation alignment to different text encoders using the zero-shot experiment. While CLIP performs best, as expected due to Stable Diffusion's training, our approach remains effective across various text encoders, demonstrating robustness to the choice of text backbone

Encoder	Acc. ↑
BLIP2	0.564
OPENCLIP	0.860
CLIP	**0.898**

Table 8. ***Dataset Size Ablation on Sup. ViT:*** Doubling the dataset size slightly improved the dense expert, but significantly improved ProbeX (by over 10%). This indicates that ProbeX uses additional samples more effectively but may be more susceptible to overfitting

Size	Method	Acc. ↑	# Params ↓
350	Dense	0.643	59m
	ProbeX	**0.757**	694k
700	Dense	0.660	59m
	ProbeX	**0.867**	694k

8. Discussion

Generalizing to Unseen Model Trees. In this paper, we focus on learning within a closed set of Model Trees. However, new Model Trees are continually added to public repositories. A primary limitation of ProbeX is its inability to generalize to these new Model Trees, requiring training new experts for new trees. Despite this drawback, ProbeX's lightweight design, combined with the fact that each expert is trained independently, allows for quick integration of new experts into the MoE as new Model Trees emerge. A promising avenue for future research is the development of a zero-shot metanetwork capable of generating new experts for previously unseen Model Trees.

Self Supervised Learning vs. Aligning Representations. In this work, we focus on aligning weight-space representations with existing representations. While self-supervised (SSL) weight-space learning [28–30] could reduce dependence on external representations, such methods typically require carefully crafted augmentations and priors to make the representations invariant to nuisance factors. For model weights, designing such augmentations is not straightforward, and many key nuisance factors are still being identified. Identifying the Model Tree nuisance attribute may hopefully accelerate the development of new SSL methods for weight-space learning.

Mechanistic vs. Functional Weight-Space Learning. Herrmann et al. [14] distinguished between two approaches to weight-space learning. The mechanistic approach treats the weights as input data and learns directly from them, while the functionalist approach (e.g., probing) interacts only with a model's inputs and outputs. Although the functionalist approach bypasses weight-space-related nuisance factors such as permutations or Model Trees, it treats the entire model as a black box, limiting its scope. ProbeX can be seen as a blend of both approaches, enabling us to operate at the weight level while engaging with the function defined by the weight matrix. This approach may facilitate the study of different model layers' functionalities. For instance, in the case of the MAE and Sup. ViT Model Trees, which share the same architecture, the most effective layer for our task differed between the two. This suggests that, despite having the same architecture, the two models utilize their layers for different functions.

Deeper ProbeX Encoders. In this work, we used encoders with a single hidden layer. In preliminary experiments, we observed that adding more layers to the encoder reduced performance, probably due to overfitting. An intriguing direction for future research would be to design deeper encoders that improve generalization or handle more complex tasks.

9. Conclusion

In this paper, we identified Model Trees as a major source of nuisance variation in model weights. We empirically showed that learning from a diverse population of models spanning multiple Model Trees is significantly more difficult than learning from models within the same tree. Building on this insight, and based on our analysis of real-world model repositories, which are dominated by a few large Model Trees, we proposed a Mixture-of-Experts approach. Specifically, for a diverse model population, we trained our Probing Expert (ProbeX) for each Model Tree. ProbeX is a theoretically grounded architecture that scales weight-space learning to large models, requiring only a few minutes to train. We validated the effectiveness of our approach using a dataset of over $12,000$ fine-tuned models that we created. Furthermore, we demonstrated the powerful generalization of ProbeX by embedding model weights into a shared space with language representations. In this space, we successfully performed model retrieval, one-class classification, and text-guided zero-shot model classification—all directly from the weights of a single model layer.

References

[1] Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. Nern–learning neural representations for neural networks. *arXiv preprint arXiv:2212.13554*, 2022. 3

[2] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017. 5

[3] Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, et al. Stealing part of a production language model. *arXiv preprint arXiv:2403.06634*, 2024. 3

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[5] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 6

[6] Amil Dravid, Yossi Gandelsman, Kuan-Chieh Wang, Rameen Abdal, Gordon Wetzstein, Alexei A Efros, and Kfir Aberman. Interpreting the weight space of customized diffusion models. *arXiv preprint arXiv:2406.09413*, 2024. 1, 3

[7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 2

[8] Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. *arXiv:2002.05688*, 2020. 3

[9] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14300–14310, 2023. 3

[10] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 3

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[12] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022. 6

[13] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier, 1990. 1, 3

[14] Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning useful representations of recurrent neural network weight matrices. In *Forty-first International Conference on Machine Learning*, 2024. 1, 3, 4, 10

[15] Eliahu Horwitz, Jonathan Kahana, and Yedid Hoshen. Recovering the pre-fine-tuning weights of generative models. In *Forty-first International Conference on Machine Learning*, 2024. 3

[16] Eliahu Horwitz, Asaf Shul, and Yedid Hoshen. On the origin of llamas: Model tree heritage recovery. *arXiv preprint arXiv:2405.18432*, 2024. 1, 3

[17] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 6

[18] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, 2021. If you use this software, please cite it as below. 9

[19] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, Efstratios Gavves, Cees GM Snoek, and David W Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024. 1, 3, 4

[20] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023. 9

[21] Derek Lim, Haggai Maron, Marc T Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. *arXiv preprint arXiv:2312.04501*, 2023. 1, 3

[22] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pages 25790–25816. PMLR, 2023. 1, 3

[23] Aviv Navon, Aviv Shamsian, Ethan Fetaya, Gal Chechik, Nadav Dym, and Haggai Maron. Equivariant deep weight space alignment. *arXiv preprint arXiv:2310.13397*, 2023. 1, 3

[24] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022. 1, 3

[25] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 6

[26] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 22500–22510, 2023. 6

[27] Mohammad Salama, Jonathan Kahana, Eliahu Horwitz, and Yedid Hoshen. Dataset size recovery from lora weights. *arXiv preprint arXiv:2406.19395*, 2024. 3

[28] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021. 1, 3, 10

[29] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.

[30] Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Forty-first International Conference on Machine Learning*, 2024. 1, 3, 10

[31] Viraj Shah, Nataniel Ruiz, Forrester Cole, Erika Lu, Svetlana Lazebnik, Yuanzhen Li, and Varun Jampani. Ziplora: Any subject in any style by effectively merging loras. *arXiv preprint arXiv:2311.13600*, 2023. 3

[32] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 2

[33] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020. 3, 6

[34] Seniha Esen Yüksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23: 1177–1193, 2012. 4

[35] Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36, 2024. 3

[36] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *Advances in neural information processing systems*, 36, 2024. 1, 3

Representing Model Weights with Language using Tree Experts

Supplementary Material

A. Proofs

A.1. Proposition 1

Proposition 1. Assume we implement $\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_{r_U}$ linear operations, i.e., matrices, and use a sufficient number of probes. The dense expert (Eq. 2) and probing network (Eq. 4) have identical expressivity.

Proof. We will prove both that the dense expert entails linear probing (1), and that probing entails linear experts (2).

Direction (1) is trivial, as linear probing is a composition of linear operations, it follows that the operation is a linear operation from $\mathbb{R}^{d_W \times d_H} \rightarrow \mathbb{R}^{d_Y}$. As the dense expert, parameterized as $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$, can express all linear operations in $\mathbb{R}^{d_W \times d_H} \rightarrow \mathbb{R}^{d_Y}$, it clearly entails linear probing.

Direction (2) requires us to prove that we can find a set of matrices $U, E[1], E[2], \cdots, E[r_U], T$ such that $\mathbf{y} = T \sum_l E[l] X \mathbf{u}_l = \sum_{ij} W_{ijk} X_{ij}$ for every $X \in \mathbb{R}^{d_W \times d_H}$ and any $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$. We show a proof by construction. Let $T = I$ (the identity matrix), $U = I$ and $E[l]_{ik} = W_{ilk}$. We have:

$$y_k = (T \sum_l E[l] X \mathbf{u}_l)_k = \sum_{ijl} W_{ilk} X_{ij} \delta_{jl} \tag{9}$$

Where δ_{jl} is 1 in the diagonal and 0 otherwise, the T is the identity matrix and cancels out. Summing over l, we obtain:

$$y_k = \sum_{ij} W_{ijk} X_{ij} \tag{10}$$

This proves that linear probing can express any dense expert. □

A.2. Proposition 2

Proposition 2. Let the Tucker low-rank tensor decomposition be given by:

$$W = \sum_{nml} M_{nml} \cdot \mathbf{t}_n \otimes \mathbf{v}_m \otimes \mathbf{u}_l \tag{11}$$

The linear ProbeX in Eq. 6 has identical expressivity as using the dense predictor in Eq. 2, with a weight tensor obeying the Tucker decomposition.

Proof. The Tucker decomposition expresses a 3D tensor $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$ by the product of a smaller tensor $M \in \mathbb{R}^{r_T \times r_V \times r_U}$ and three matrices $U \in \mathbb{R}^{d_H \times r_U}, V \in \mathbb{R}^{d_W \times r_V}, T \in \mathbb{R}^{d_Y \times r_T}$ as follows:

$$W = \sum_{nml} M_{nml} \cdot \mathbf{t}_n \otimes \mathbf{v}_m \otimes \mathbf{u}_l \tag{12}$$

Where $\otimes$ is the tensor product, and $\mathbf{u}_q, \mathbf{v}_q, \mathbf{t}_q$ are the q^{th} column vectors of matrices U, V, T respectively.

The expression for the Tucker decomposition in index notation is:

$$W_{ijk} = \sum_{nml} T_{kn} M_{nml} V_{im} U_{jl} \tag{13}$$

By linearity, we can reorder the sums as:

$$W_{ijk} = \sum_{n} T_{kn} \sum_{ml} M_{nml} V_{im} U_{jl} \tag{14}$$

We can equivalently split tensor M into r matrices $M[1], M[2], \cdots, M[r]$, so that:

$$W_{ijk} = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{15}$$

Multiplying tensor W by input matrix $X \in \mathbb{R}^{d_W \times d_H}$, the result is:

$$\tilde{y}_k = \sum_{ij} X_{ij} W_{ijk} = \sum_{ij} X_{ij} \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{16}$$

By linearity, we can reorder the sums:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} \sum_{ij} V_{im} X_{ij} U_{jl} \tag{17}$$

Rewriting U using its column vectors this becomes:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} \sum_{i} V_{im} (X \mathbf{u}_l)_i \tag{18}$$

Rewriting the sum over i as a matrix multiplication:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} (V^T X \mathbf{u}_l)_m \tag{19}$$

Rewriting the sum over m as a matrix multiplication:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{l} (M[l] V^T X \mathbf{u}_l)_n \tag{20}$$

Rewriting the sum over n as a matrix multiplication, we finally obtain:

$$\tilde{y} = T \sum_{l} M[l] V^T X \mathbf{u}_l \tag{21}$$

□