Learning on Model Weights using Tree Experts

Eliahu Horwitz* Bar Cavia* Jonathan Kahana* Yedid Hoshen
The Hebrew University of Jerusalem, Israel
https://horwitz.ai/probex/
{eliahu.horwitz, bar.cavia, jonathan.kahana, yedid.hoshen}@mail.huji.ac.il

Abstract

The number of publicly available models is rapidly increasing, yet most remain undocumented. Users looking for suitable models for their tasks must first determine what each model does. Training machine learning models to infer missing documentation directly from model weights is challenging, as these weights often contain significant variation unrelated to model functionality (denoted nuisance). Here, 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. Concretely, while learning across Model Trees requires complex architectures, even a linear classifier trained on a single model layer often works within trees. While effective, these linear classifiers are computationally expensive, especially when dealing with larger models that have many parameters. To address this, we introduce Probing Experts (ProbeX), a theoretically motivated and lightweight method. Notably, ProbeX is the first probing method specifically designed to learn from the weights of a single hidden model layer. We demonstrate the effectiveness of ProbeX by predicting the categories in a model's training dataset based only on its weights. Excitingly, ProbeX can map the weights of Stable Diffusion into a weight-language embedding space, enabling model search via text, i.e., zero-shot model classification.

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. Ideally, this abundance would allow users to simply download the most suitable model for their task, thereby saving resources, reducing training time, and potentially improving accuracy. However, the lack of adequate documentation for most models makes it challenging for users to determine a model's suitability for specific tasks. This motivates developing machine learning methods that can infer model functionality and missing documentation directly from model weights. The emerging field of weight-space learning [31, 34, 40, 52, 54] studies how to design and train 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 [29, 43], model performance [31], and even generate new model parameters [9, 35]. In this work, we focus on predicting the categories in a model's training dataset as a proxy for the concepts it can recognize or generate. As a initial step towards model search-by-text, we demonstrate that it is sometimes possible to align weights with language, enabling zero-shot model classification.

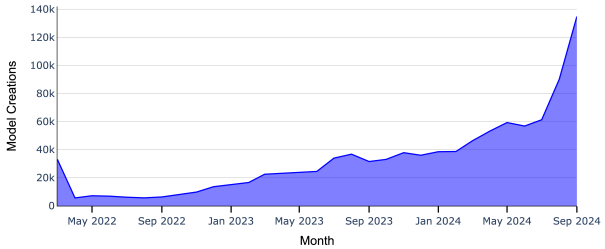


Figure 1. ***Growth in Hugging Face models:*** The number of public models is growing fast, but they are mostly undocumented. Fully benefiting from them requires effective model search.

However, extracting meaningful information from model weights is challenging. While the weights of a neural network are a function of its training data, they are also affected by the optimization process, which may introduce nuisance variation unrelated to attributes of interest. Neuron permutation [17] is perhaps the most studied nuisance factor and has driven research into permutation-invariant architectures [27, 29, 31, 32, 54] and carefully designed augmentations [18, 40, 41, 43]. In this paper, we highlight another important nuisance factor, the weights at the beginning of optimization.

The key insight in this paper is that models within a Model Tree [21] have reduced nuisance variation. A Model Tree describes a set of models that share a common ancestor (root), with each model derived by fine-tuning either from

*Equal contribution

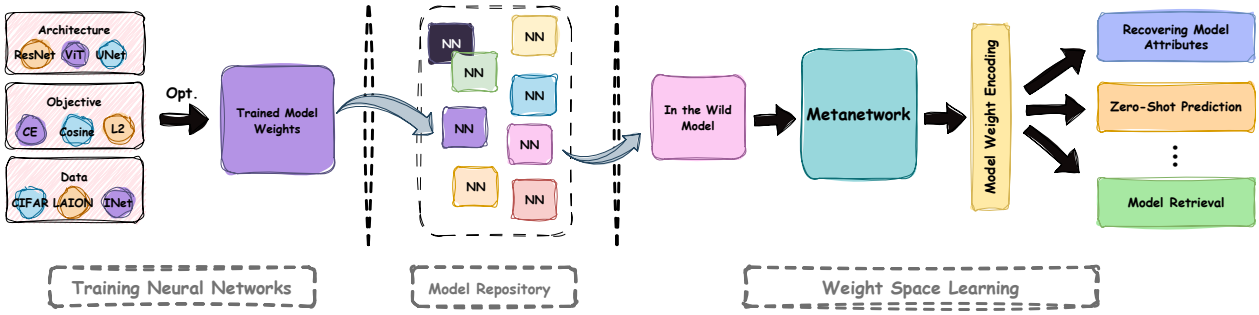


Figure 2. ***Weight-space learning:*** *Left.* Model weights are a direct product of the optimization process and training data. *Center.* Models are uploaded to public repositories, e.g., Hugging Face, typically lacking documentation. This prevents model search, making models hard to discover and reuse. *Right.* Weight-space learning treats each model as a data point and designs metanetworks: networks that process weights of other models as inputs. We train metanetworks that predict categories in a model's training dataset. As a first step towards model search-by-text, we also align weights with language, enabling zero-shot model classification.

the root or from one of its descendants. For example, the Llama3 [10] Model Tree includes all models fine-tuned from Llama3 or any of its descendants. In practice, most public models belong to a relatively small number of Model Trees, for instance, on Hugging Face, fewer than 20 Model Trees cover most models (see Fig. 4). *We hypothesize that learning on models from the same tree is significantly simpler than learning from models across different trees.* We demonstrate this empirically, showing that standard linear models perform well on models within the same tree but fail to learn when applied to models from many different trees.

However, standard linear classifiers require too many parameters for learning on large models. To address this, we present single layer ***Probing eXperts*** (ProbeX), a theoretically grounded architecture that scales weight-space learning to large models. Unlike conventional probing methods, ProbeX operates on hidden model layers. Remarkably, ProbeX can handle models with hundreds of millions of parameters, requiring under 10 minutes to train. When working with multiple Model Trees, we use a Mixture-of-Experts.

To evaluate our method, we introduce a dataset of 14,000 models across 5 disjoint Model Trees spanning multiple architectures and functionalities. ProbeX achieves state-of-the-art results on the task of training category prediction, accurately identifying the specific classes within a model's training dataset. Excitingly, ProbeX can also align fine-tuned Stable Diffusion weights with language representations. This capability enables a new task: zero-shot model classification, where models are classified via a text prompt describing their training data, ProbeX achieves 89.8% accuracy on this.

To summarize, our main contributions are:

1. Identifying that learning within Model Trees is much simpler than learning across trees.
2. Introducing Probing Experts (ProbeX), a lightweight, theoretically motivated method for weight-space learning.
3. Proposing the task of zero-shot model weight classification and tackling it by aligning model weights with language representations, achieving strong results.

2. Related works

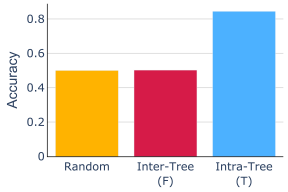
Despite the popularity of neural networks, little research has explored using their weights as inputs for machine learning methods. Eilertsen et al. [11], Unterthiner et al. [47] were among the first to systematically analyze model weights to predict undocumented properties like the training dataset or generalization error. Some works aim to learn general representations [18, 40, 41, 43] for multiple properties, while others incorporate specific priors [27, 29, 31, 32, 39, 54] to directly predict the property. A major challenge with model weights is the presence of many parameter space symmetries [17]. 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 need to account for these symmetries. To avoid the issue of weight symmetries, recent methods [5, 18, 26, 27, 30] 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. Probes are also used in other fields such as [2, 8, 23, 45, 46] mechanistic interpretability for solving different tasks. However, until now, this was limited to passing the probes through the entire model and did not apply to single layers. Concurrent to our work, Putterman et al. [36] propose a method for linearly classifying LoRA [22] models to infer their functionality. While they do not describe it this way, it can be viewed as probing.

Other applications include generating weights [1, 9, 12, 14, 35], merging models [25, 33, 44, 49, 50], and recovering the weights of unpublished models [3, 20].

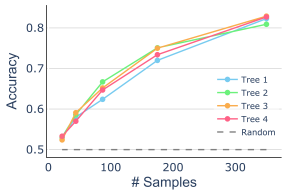
3. Motivation

3.1. The challenge

While machine learning on images, text, and audio is fairly advanced, learning from model weights is still in its infancy and the key nuisance factors remain unclear. Many approaches focused on neuron permutations [27, 31, 53] as



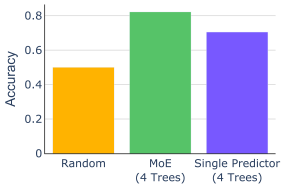
(a) ***Intra vs. Inter tree learning***



(b) ***Positive transfer within trees***

(c) ***Negative transfer between trees***

# Trees	# Models	Acc.
1	350	0.844
2	700	0.752
3	1050	0.686
4	1400	0.724



(d) ***MoE vs. Shared predictor***

Figure 3. ***Tree membership and weight-space learning:*** We conduct 4 motivating experiments that illustrate the benefits of learning within Model Trees. In each experiment, we train a linear classifier to predict the classes a ViT model was fine-tuned on. First, we show that *learning within Model Trees is significantly simpler (a)* by comparing a metanetwork trained on models from the same tree (T) with one trained on models from different trees (F). Next, we demonstrate *positive transfer within the same tree (b)* by showing that adding more models from the same tree improves the performance. Surprisingly, we observe *negative transfer between Model Trees (c)*, where adding samples from other trees degrades performance on a single tree. Finally, we find that *expert learning is preferable when learning from multiple trees (d)*, as a single shared metanetwork performs worse than an expert metanetwork per tree (MoE).

the core nuisance factor. However, permutations are not likely to describe all nuisance variation, as neurons and layers can serve different roles across models and architectures. This paper highlights that learning within Model Trees [21] reduces nuisance variation, making learning simpler.

3.2. Seeing the forest by seeing the trees

Background: Model Trees. Following Horwitz et al. [21], we represent model populations as a *Model Graph* comprising disjoint directed *Model Trees*. In this graph, each node is a model, with directed edges connecting each model to those directly fine-tuned from it. Since a model has at most one parent, the graph forms a set of non-overlapping trees. Importantly, while all the models within a Model Tree share the same architecture, two models with the same architecture but different roots belong to different Model Trees. E.g., DINO [4] and MAE [16] both use the same ViT-B/16 [7] architecture but form disjoint trees. Note that while Model Trees were originally proposed for model attribution, here, we use them differently: to group models with shared initial weights, thereby reducing nuisance variation. Consequently, we do not need to recover the precise tree structure, but only to map each model into a particular tree.

Tree membership and weight-space learning. Current weight-space methods generally rely on a single metanetwork to learn from a diverse model population spanning multiple trees. We hypothesize that learning within a single Model Tree is significantly simpler than learning across multiple Model Trees. I.e., we expect that dividing the population into distinct Model Trees and learning within each tree, can greatly simplify weight-space learning. To test this hypothesis, we simulate various model populations.

First, we create dataset A by randomly selecting 50 classes from CIFAR100 and dataset B by randomly choosing 25 of the remaining classes. Second, we *pre-train* a classifier on B for a *single epoch*. We then train two different model populations, T (Model Tree) and F (Model Forest), each with 500 ResNet9 models. All models are trained to classify between 25 randomly selected classes from A. The populations differ in one aspect only: models in F are initialized randomly, while models in T are all initialized from the same model pre-trained on B. Therefore, *all models in T belong to the same Model Tree, while each model in F belongs to a different tree.* Given a model, the task is to predict which 25 out of the 50 classes from A it was trained on. Using T and F, we can analyze learning within and across Model Trees.

Is learning within the same Model Tree beneficial? We begin with a simple experiment, training a *linear* metanetwork for models in T and another one for models in F. In line with our hypothesis, there is a large performance gap between the two settings (see Fig. 3a). While learning on models within the same tree achieved good results (0.844), learning on models from many different trees achieved near random accuracy (0.502). This demonstrates that i) Model Tree membership introduces significant non-semantic variations in model weights, and ii) even a single epoch of shared pre-training might be enough to eliminate the variation.

Which models have positive transfer? Next, we investigate whether models have positive transfer. I.e., whether increasing the size of a training set helps learning. To this end, we pre-train 4 different models on B and use them to fine-tune populations $T_1, \ldots, T_4$ similarly to the way T was created. First, we train different metanetworks on increasingly larger population subsets. We find that when all models belong to the same tree, increasing the size of the training set results in better performance (see Fig. 3b). I.e., models in the same tree have positive transfer.

Next, we test whether adding models from different trees is helpful. We start by training and evaluating a metanetwork on models from T_1. We gradually add to the training set models from other trees ($T_2, \ldots, T_4$) and check whether training different metanetworks on these larger datasets improves the classification of models from T_1. Surprisingly, we find that adding models from different trees *decreases*

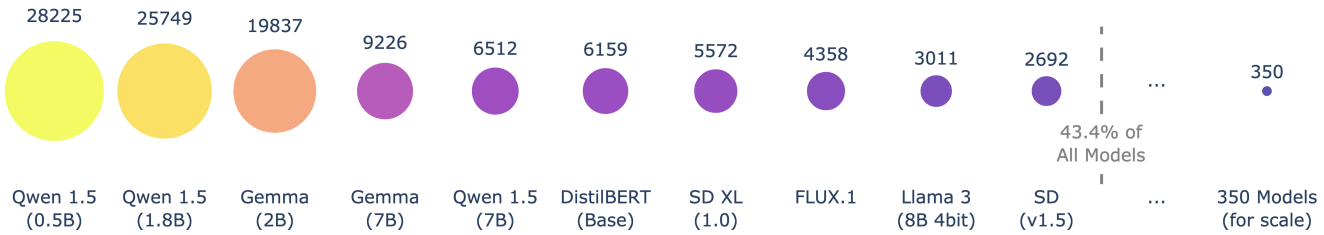


Figure 4. ***Largest Model Trees on Hugging Face:*** We show the 10 largest Model Trees on Hugging Face. Our insight is that learning an expert for each tree greatly simplifies weight-space learning. This is a practical setting as a few large Model Trees dominate the landscape.

the accuracy on T_1 (see Fig. 3c), demonstrating that learning from multiple trees has a negative transfer effect.

How to learn from multiple Model Trees? Finally, we compare learning a separate expert model per tree and combine them via a Mixture-of-Experts (MoE) approach vs. learning a single shared metanetwork for all trees. We find that the MoE approach outperforms joint training (see Fig. 3d), motivating us to learn an expert per tree.

A Few Large Trees Dominate the Landscape. To explore the practicality of working within Model Trees, we analyzed approximately $250k$ models from the Hugging Face model hub[1], for more details see App. F. We find that most public models belong to a small number of large Model Trees. For instance, 20 Model Trees already cover 50% of the models. Moreover, the 196 trees which contain 100 or more models, collectively cover over 70% of all models. Fig. 4 shows a breakdown of the top 10 Model Trees on Hugging Face. *We conclude that learning metanetworks within Model Trees is both effective and practical.*

4. Probing Expert

Notation and task definition. Consider a model $\mathcal{F}$ with s layers and denote the dimension of each layer by d_H and d_W[2]. 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 models use LoRA [22], we multiply the decomposed matrices $X = BA$ and work with the full matrix. Our task is to map a weight matrix $X \in \mathbb{R}^{d_W \times d_H}$ to an output vector $\mathbf{y} \in \mathbb{R}^{d_Y}$, where $\mathbf{y}$ is a logits vector in classification tasks or an external semantic representation in text alignment tasks.

4.1. Dense experts

Building on the motivation discussed in Sec. 3, we learn a separate expert metanetwork for each Model Tree. 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 $W \in \mathbb{R}^{d_H \times d_W \times d_Y}$. Formally,

$$y_k = \sum_{ij} W_{ijk} X_{ij} \tag{1}$$

Although such an expert can achieve impressive performance, its high parameter count (often exceeding 1 billion) makes it impractical due to excessive memory requirements.

4.2. Probing

Probing recently emerged as a promising approach for processing neural networks [18, 26, 27]. Instead of directly processing the weights of the target model, it passes *probes* (input vectors) through the model and represents the model by its outputs. As each probe provides partial information about the model, fusing information from a diverse set of probes improves representations. Passing probes through the model is typically cheaper than passing all the network weights through a metanetwork, making probing much more parameter efficient than the alternatives.

Formally, let $f_X : \mathbb{R}^{d_W} \to \mathbb{R}^{d_H}$ be the input function (e.g., a neural network). Probing methods first select a set of probes $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{r_U} \in \mathbb{R}^{d_W}$ and pass each probe $\mathbf{u}_l$ through the function f_X, resulting in a probe response $\mathbf{z}_l = f_X(\mathbf{u}_l) \in \mathbb{R}^{d_H}$. A per-probe encoder $\mathcal{E}_l$ then maps the response $\mathbf{z}_l$ of each probe to encoding $\mathbf{e}_l \in \mathbb{R}^{d_V}$. The final model encoding $\mathbf{e}$ is the sum of the encodings of all probes:

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

Finally, a prediction head $\mathcal{T} : \mathbb{R}^{d_V} \to \mathbb{R}^{d_Y}$, maps the model encoding to the final prediction:

$$\mathbf{y} = \mathcal{T}(\mathbf{e}) \tag{2}$$

We begin with a linear probe encoder, where we can theoretically motivate our architectural design choices. We later extend our architecture to the non-linear case.

4.3. Single layer probing experts

Dense vs. linear probing experts. Traditionally, probing methods focus only on model inputs and outputs, thus avoiding many nuisance factors (e.g., neuron permutations).

[1] We only consider models with information about their pre-training.

[2] We reshape higher-dimensional weight tensors (e.g., convolutional layers) into 2D matrices, with the first dimension being the output channels.

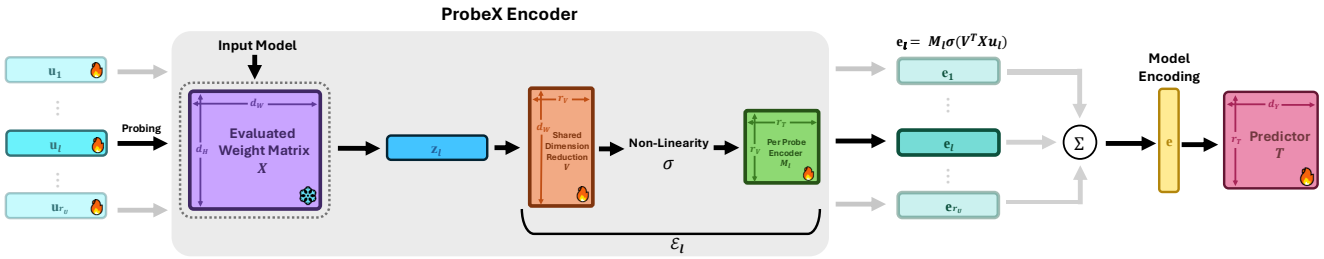


Figure 5. ***ProbeX overview.*** Unlike conventional probing methods that operate only on inputs and outputs, our lightweight architecture scales weight-space learning to large models by probing hidden model layers. ProbeX begins by passing a set of learned probes, $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{r_U}$, through the input weight matrix X. A projection matrix V, shared between all probes, reduces the dimensionality of the probe responses, followed by a non-linear activation. Each probe response is then mapped to a probe encoding $\mathbf{e}_l$ via a per-probe encoder matrix M_l. We sum the probe encodings to obtain the final model encoding $\mathbf{e}$, which the predictor head maps to the task output $\mathbf{y}$.

However, as working within Model Trees reduces nuisance variation (see Sec. 3), we hypothesize that probing can succeed even when applied to hidden layers. We focus on the case where $f_X(u) = X(u)$ and probing encoders are linear.

Proposition 1. *Assume $\mathcal{E}_1, \ldots, \mathcal{E}_{r_U}$ are all linear operations and a sufficient number of probes. The dense expert (Eq. 1) and linear probing network (Eq. 2) have identical expressivity.*

Proof. App. A.1 □

Deriving a single layer probing architecture. Having demonstrated that the linear probing framework can match the expressivity of the dense expert, we now address the primary issue of dense experts: their high parameter count. Recall that each of the r_U probes has a dedicated encoder, parameterized by a large matrix. We can therefore factorize each probe encoder into a product of two matrices. The first is a dimensionality reduction matrix $V \in \mathbb{R}^{d_W \times r_V}$, *shared* across probes. This matrix projects the high-dimensional outputs of $X \in \mathbb{R}^{d_W \times d_H}$ into a lower dimension r_V. The second matrix, $M_l \in \mathbb{R}^{r_V \times r_T}$ is *unique* to each probe encoder and can be much smaller. By sharing the larger matrix V among all probes and using a smaller, probe-specific matrix M_l, we significantly reduce the overall parameter count. Finally, the per-probe encoder is given by:

$$\mathcal{E}_l(\mathbf{z}_l) = M_l V^T \mathbf{z}_l \tag{3}$$

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

$$\mathbf{y} = T \sum_l M_l V^T X^T \mathbf{u}_l \tag{4}$$

In Prop. 2 we derive our linear ProbeX (Eq. 4) from the dense expert (Eq. 1) using the Tucker tensor decomposition.

Proposition 2. *The linear ProbeX (Eq. 4) has identical expressivity as using the dense predictor (Eq. 1), when the weight tensor W obeys the Tucker decomposition:*

$$W_{Tucker} = \sum_{nml} M_{nml} \cdot \boldsymbol{t}_n \otimes \boldsymbol{v}_m \otimes \boldsymbol{u}_l$$

Proof. App. A.2 □

Non-linear probing experts In Props. 1 and 2 we establish the relation between linear ProbeX and the dense expert. To make ProbeX more expressive, we add a non-linearity σ between the two matrices V, M_l making ProbeX a factorized one hidden layer neural network:

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

In our experiments we chose σ to be the ReLU function. Note the approach can easily be extended to deeper probe encoders. We present an overview of ProbeX in Fig. 5.

Training ProbeX. For classification tasks, we use ProbeX to map model weights to logits via the cross-entropy loss (Sec. 6.1). For representation alignment, we use a contrastive loss (Sec. 6.2). In all cases, we optimize $V, \mathbf{u}_1, \cdots, , \mathbf{u}_{r_U}, M_1, \cdots, M_{r_U}, T$ 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 can extract an encoding from each layer using ProbeX and concatenate them. Finally, we map the concatenated encoding to the output y using a matrix T, training everything end-to-end. Notably, training ProbeX on a single layer takes under 10 minutes on a single small GPU (e.g., $10GB$ of VRAM).

4.4. Handling multiple Model Trees

In practice, models may belong to multiple Model Trees. We therefore propose a mixture-of-tree-experts approach, consisting of a router metanetwork that maps models to their tree and a per-tree expert metanetwork. Differently from recent MoE methods [51] that learn the router and experts end-to-end, we decouple the two; first learning the

Table 1. ***Training dataset class prediction 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 were used during training. While the dense expert performs moderately well, ProbeX achieves better accuracy with roughly $\times 30$ fewer parameters.

	ResNet		DINO		MAE		Sup. ViT		MoE	
Method	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓	**Acc.** ↑	**# Params** ↓
Random	0.5	-	0.5	-	0.5	-	0.5	-	0.5	-
StatNN	0.631	-	0.511	-	0.502	-	0.522	-	0.541	-
Dense	0.713	105m (×45)	0.614	59m (×25)	0.666	59m (×25)	0.663	59m (×25)	0.664	282m (×30)
ProbeX	**0.842**	2.3m	**0.705**	2.3m	**0.765**	2.3m	**0.885**	2.3m	**0.799**	9.2m

routing function and then the ProbeX experts. For the routing function, we opt for a fast and simple clustering algorithm. Specifically, we cluster the set of models into trees 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_2:

$$R(X) = arg\min_k \|X - \hat{X}_k\|_2 \tag{5}$$

In practice, we find that these clusters perfectly match the division into Model Trees [13, 21] (see App. C).

5. Model jungle dataset

We construct *Model Jungle* (Model-J), a dataset that simulates the structure of real-world model repositories, with models organized into a small set of *disjoint* Model Trees. These trees consist of large models that vary in architecture, task, and size, with each fine-tuned model using a set of randomly sampled hyperparameters. Model-J includes 14,000 models, divided into two main splits. We shortly describe the splits here, for further details see App. G.

Discriminative. We fine-tune 4,000 models for image classification. These models belong to one of 4 Model Trees: i) Supervised ViT (Sup. ViT) [7], ii) DINO [4], iii) MAE [16], and iv) ResNet-101 [15]. Each model is fine-tuned (using "vanilla" full fine-tuning) to classify images from a random subset of 50 out of the 100 CIFAR100 classes.

Generative. We fine-tune $10,000$ Stable Diffusion (SD) [37] personalized models [38]. Each model fine-tuned on $5-10$ images, randomly sampled without replacement, originating from the same ImageNet [6] class. This split consists of 2 variants each with $5,000$ models: i) SD_{200}. A fine-grained variant with 25 models from each class using the first 200 ImageNet classes (mostly different animal breeds). ii) SD_{1k}. A low resource variant with 5 models per class for all ImageNet classes. To save compute and storage, we follow common practice and use LoRA [22] fine-tuning. We set aside an additional test subset of models trained on randomly selected *holdout* classes, $30 \in SD_{200}$ and $150 \in SD_{1k}$.

6. Experiments

We train ProbeX and the baselines on each layer and choose the best layer according to the validation set. For more implementation details see App. H. We use accuracy as the evaluation metric, we also report the parameter count.

Baselines. Most 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 [47].* This permutation-invariant baseline extracts 7 simple statistics (mean, variance, and 5 different quantiles) for the weights and biases of each layer. It then trains a gradient-boosted tree on the concatenated statistics. ii) *Dense Expert.* Training a single linear layer on the flattened raw weights. Note that this baseline produces impractically large classifiers. E.g., a single layer classifier trained to classify SD_{1k} typically has $1.4B$ parameters, twice the size of the entire SD model. We also attempted to run Neural Graphs [27], but it struggled to scale to the ViT architecture even when adapted to the single-layer case (see App. H). Since all attempts yielded near-random results, we did not report it.

6.1. Predicting training dataset classes

Here, 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 predict a set of 100 binary labels, each indicating whether a specific class was included in the model's fine-tuning data. Concretely, we train Eq. 3 with 100 jointly optimized binary classification heads. This task is particularly challenging, as each class represents only 2% of the model's training data, making its signature relatively 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. However, these model cards are often poorly documented and lack details about the specific classes a model was trained on. In contrast, our metanetwork could allow users to filter for suitable models more effectively.

In Tab. 1, we present the results of ProbeX for each Model Tree in the discriminative split. While dense expert performs better than random, ProbeX performs significantly better,

Table 2. ***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) but also to entirely new object categories in a zero-shot manner, without requiring additional training. This suggests that ProbeX successfully aligns model encodings with CLIP representations.

	Method	In Dist. ↑ Acc.	Zero-shot. ↑ Acc.	# Params ↓
SD_{200}	Random	0.006	0.033	-
	$\text{StatNN}_{\text{MLP}}$	0.018	0.075	2.6m
	$\text{StatNN}_{\text{Linear}}$	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	-
	$\text{StatNN}_{\text{MLP}}$	0.001	0.029	2.6m
	$\text{StatNN}_{\text{Linear}}$	0.01	0.045	689k
	Dense	**0.382**	0.343	210m (×84)
	ProbeX	0.296	**0.505**	2.5m

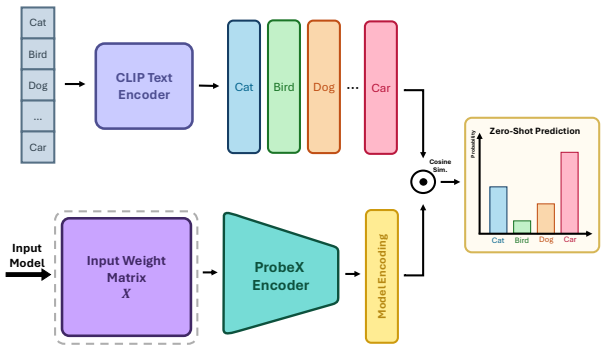


Figure 6. ***Zero-shot inference overview.*** We align model weights with a pre-trained text encoder for zero-shot model classification. We extract the CLIP text embedding of each class name and use ProbeX to encode the weight matrix X into a shared weight-text embedding space. Classification follows 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.

improving accuracy by more than 10% on average with roughly $\times 30$ fewer parameters. The MoE router (Eq. 5) achieves perfect accuracy, for more details see App. C.

6.2. Aligning weights to text representations

We hypothesize that the weights of models conditioned on text can be aligned with a text representation. We therefore learn a mapping between the weights of models in the generative split (see Sec. 5) and the CLIP text embeddings of the model's training dataset categories. This process creates a shared weight-text embedding space. We evaluate these aligned representations across various tasks and demonstrate strong generalization. To the best of our knowledge, ProbeX is the first method that learns weight representations with zero-shot capabilities.

Representation alignment. We train ProbeX to map model encodings to pre-trained text embeddings (e.g., CLIP). This mapping is supervised, as we have paired data consisting of i) model weights and ii) text embedding of the category of their fine-tuning dataset. Our training loss is similar to CLIP, i.e., the optimization objective is that the cosine similarity between the ProbeX model encoding to the ground truth class text embedding will be high, and all other classes lower.

6.2.1. Zero-shot classification

We begin by testing the zero-shot capabilities of our aligned representation on the holdout splits of Model-J (see Sec. 5). Specifically, given a weights-to-text mapping function, we compute the similarity between the model encoding 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. 6). We perform a similar experiment for in-distribution data (categories seen during training), i.e., a standard classification setting. In Tab. 2, we show the top-1 accuracy of our method compared to the dense expert. Importantly, our method generalizes not only to unseen models trained on the same classes (i.e., in-distribution) but also to entirely new object categories (i.e., zero-shot). ProbeX detects classes unseen during training with 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 encodings with CLIP's representations.

kNN classification. Similarly to the zero-shot setting, using the aligned representations, kNN can correctly classify the training dataset class. The score is the average kNN distances between the text aligned ProbeX representation of the test model and the training models from this class. Tab. 3 compares our aligned representation with simply using raw weights, our representation performs much better.

6.2.2. Unsupervised downstream tasks

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 text-aligned model representations as the similarity metric. Fig. 7 shows the 3 nearest-neighbors for 3 query models, each fine-tuned using a different dataset. We visualize each model by showing 2 images from its training set. Indeed, the retrieved models are closely related to the query models, showing our representation captures highly semantic attributes even in fine-grained cases. For instance, while SD_{200} contains many different dog and cat breeds, our retrieval accurately returns the breed that the query model was trained on.

One-class-classification. We further examine our text-

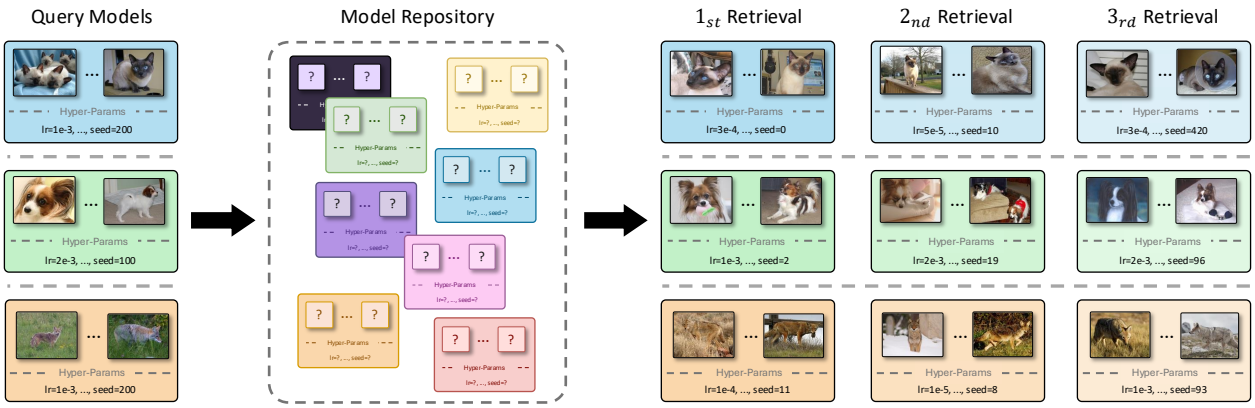


Figure 7. ***Qualitative retrieval results:*** For each query model, we search for the models trained on the most similar categories, measuring similarity via the cosine distance between the text-aligned ProbeX model representations. We present the 4 nearest neighbors for three query models, each fine-tuned on a different category. For visualization, we show two of the images used to train the model. Indeed, the retrieved models are of similar animal breeds to the query models, indicating our representations accurately capture fine-grained semantics.

aligned representations for detecting out-of-distribution models (OOD). For each held-out class, we label it as "normal" and compute the average kNN distance between all test models and the training set of the normal class. Samples near the normal distribution are considered normal while others are labeled as OOD. We average the results over all classes. In Tab. 3 we report the mean ROC AUC score, using the kNN similarity score for separating normal and OOD models. Indeed, the results show that our method can detect OOD models much more accurately than other methods. This result remains consistent across varying numbers of neighbors, clearly demonstrating that the representation extracted by ProbeX captures more semantic relations.

7. Ablations

Activation function. We ablate the need for a non-linear ProbeX using the SD_{200} dataset; results are shown in Tab. 4. Interestingly, while the use of ReLU slightly improves in-distribution classification performance (0.953 without vs. 0.973 with), the main benefit is in zero-shot capabilities (0.564 without vs. 0.898 with). This significant difference in zero-shot performance suggests that, while the linear version of ProbeX can effectively represent the training classes, generalizing to unseen classes requires a deeper model.

Text encoder. We use SD_{200} to ablate the sensitivity of our method to the precise language encoder used. While CLIP performs best (0.898), our approach remains effective across different text encoders (e.g., 0.860 with OpenCLIP [24], and 0.564 with BLIP2 [28]), see App. E for more details.

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.

Table 3. ***kNN and OCC results:*** Average results over all 30 heldout classes of SD_{200}. ProbeX achieves the highest results for both.

	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**

Table 4. ***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 categories well, ReLU enhances generalization.

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

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 and the ability to train experts independently in under 10 minutes allows for quick integration of new experts.

Aligning representations of other models. When aligning model weights with text representations, we focused on SD models. In preliminary experiments, we found that aligning models from the discriminative split performs well on in-distribution classes but does not generalize to unseen (zero-shot) classes. We hypothesize that the cross-attention layers in SD models facilitate alignment between model weights and text representations. Extending this alignment to other model architectures is left for future work.

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 is designing deeper encoders that improve generalization or handle more complex tasks.

9. Conclusion

In this paper, we take the first step toward model search based solely on model weights. We identify that learning from models within the same Model Tree is significantly simpler than learning across different trees. This setting is practical as most public models belong to a few large Model Trees. We therefore introduce Probing Expert (ProbeX), a theoretically grounded architecture that scales weight-space learning to large models. As public repositories consist of multiple trees, we propose a Mixture-of-Experts approach. We demonstrate that ProbeX can embed model weights into a shared representation space alongside language embeddings, enabling text-guided zero-shot model classification.

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. 2

[2] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017. 2

[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. 2

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 3, 6

[5] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023. 2

[6] 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

[7] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3, 6

[8] Amil Dravid, Yossi Gandelsman, Alexei A Efros, and Assaf Shocher. Rosetta neurons: Mining the common units in a model zoo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1934–1943, 2023. 2

[9] 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, 2

[10] 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

[11] 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. 2

[12] 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. 2

[13] Almog Gueta, Elad Venezian, Colin Raffel, Noam Slonim, Yoav Katz, and Leshem Choshen. Knowledge is a region in weight space for fine-tuned language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. 6

[14] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 2

[15] 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

[16] 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. 3, 6

[17] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier, 1990. 1, 2

[18] 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, 2, 4, 13

[19] Dominik Honegger, Konstantin Schürholt, and Damian Borth. Sparsified model zoo twins: Investigating populations of sparsified neural network models. *arXiv preprint arXiv:2304.13718*, 2023. 16

[20] 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. 2

[21] Eliahu Horwitz, Asaf Shul, and Yedid Hoshen. On the origin of llamas: Model tree heritage recovery. *arXiv preprint arXiv:2405.18432*, 2024. 1, 3, 6

[22] 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. 2, 4, 6

[23] Qihan Huang, Jie Song, Mengqi Xue, Haofei Zhang, Bingde Hu, Huiqiong Wang, Hao Jiang, Xingen Wang, and Mingli Song. Lg-cav: Train any concept activation vector with language guidance. *arXiv preprint arXiv:2410.10308*, 2024. 2

[24] 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. 8

[25] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022. 2

[26] Jonathan Kahana, Eliahu Horwitz, Imri Shuval, and Yedid Hoshen. Deep linear probe generators for weight space learning. *arXiv preprint arXiv:2410.10811*, 2024. 2, 4

[27] 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, 2, 4, 6, 17

[28] 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. 8

[29] 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, 2

[30] Daohan Lu, Sheng-Yu Wang, Nupur Kumari, Rohan Agarwal, Mia Tang, David Bau, and Jun-Yan Zhu. Content-based search for deep generative models. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–12, 2023. 2

[31] 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, 2

[32] 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, 2

[33] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023. 2

[34] Koyena Pal, David Bau, and Renée J Miller. Model lakes. *arXiv preprint arXiv:2403.02327*, 2024. 1

[35] 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, 2

[36] Theo Putterman, Derek Lim, Yoav Gelberg, Stefanie Jegelka, and Haggai Maron. Learning on loras: Gl-equivariant processing of low-rank weight spaces for large finetuned models. *arXiv preprint arXiv:2410.04207*, 2024. 2

[37] 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

[38] 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

[39] Mohammad Salama, Jonathan Kahana, Eliahu Horwitz, and Yedid Hoshen. Dataset size recovery from lora weights. *arXiv preprint arXiv:2406.19395*, 2024. 2

[40] 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, 2, 13

[41] 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. 1, 2, 13

[42] Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Model zoos:

A dataset of diverse populations of neural network models. *Advances in Neural Information Processing Systems*, 35:38134–38148, 2022. 16

[43] 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, 2, 13

[44] 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. 2

[45] Tamar Rott Shaham, Sarah Schwettmann, Franklin Wang, Achyuta Rajaram, Evan Hernandez, Jacob Andreas, and Antonio Torralba. A multimodal automated interpretability agent. In *Forty-first International Conference on Machine Learning*, 2024. 2

[46] Shir Ashury Tahan, Ariel Gera, Benjamin Sznajder, Leshem Choshen, Liat Ein Dor, and Eyal Shnarch. Label-efficient model selection for text generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8384–8402, 2024. 2

[47] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020. 2, 6

[48] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 14

[49] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022. 2

[50] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[51] 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. 5

[52] Allan Zhou, Chelsea Finn, and James Harrison. Universal neural functionals. *arXiv preprint arXiv:2402.05232*, 2024. 1

[53] 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. 2

[54] 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, 2

A. Proofs

A.1. Proposition 1

Proposition 1. Assume $\mathcal{E}_1, \ldots, \mathcal{E}_{r_U}$ are all linear operations and a sufficient number of probes. The dense expert (Eq. 1) and linear probing network (Eq. 2) 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 $\mathbb{R}^{d_W \times d_H} \to \mathbb{R}^{d_Y}$. As the dense expert, parameterized as $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$, can express all linear operations in $\mathbb{R}^{d_W \times d_H} \to \mathbb{R}^{d_Y}$, it clearly entails linear probing.

Direction (2) requires us to prove that we can find a set of matrices $U, \mathcal{E}[1], \mathcal{E}[2], \cdots, \mathcal{E}[r_U], T$ such that $\mathbf{y} = T \sum_l \mathcal{E}[l] X \mathbf{u}_l = \sum_{ij} W_{ijk} X_{ij}$ 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 $\mathcal{E}[l]_{ik} = W_{ilk}$. We have:

$$y_k = (T \sum_l \mathcal{E}[l] X \mathbf{u}_l)_k = \sum_{ijl} W_{ilk} X_{ij} \delta_{jl} \tag{6}$$

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:

$$y_k = \sum_{ij} W_{ijk} X_{ij} \tag{7}$$

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

A.2. Proposition 2

Proposition 2. The linear ProbeX (Eq. 4) has identical expressivity as using the dense predictor (Eq. 1), when the weight tensor W obeys the Tucker decomposition:

$$W_{\text{Tucker}} = \sum_{nml} M_{nml} \cdot \mathbf{t}_n \otimes \mathbf{v}_m \otimes \mathbf{u}_l$$

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:

$$W = \sum_{nml} M_{nml} \cdot \mathbf{t}_n \otimes \mathbf{v}_m \otimes \mathbf{u}_l \tag{8}$$

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:

$$W_{ijk} = \sum_{nml} T_{kn} M_{nml} V_{im} U_{jl} \tag{9}$$

By linearity, we can reorder the sums as:

$$W_{ijk} = \sum_{n} T_{kn} \sum_{ml} M_{nml} V_{im} U_{jl} \tag{10}$$

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

$$W_{ijk} = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{11}$$

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

$$\tilde{y}_k = \sum_{ij} X_{ij} W_{ijk} = \sum_{ij} X_{ij} \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{12}$$

By linearity, we can reorder the sums:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} \sum_{ij} V_{im} X_{ij} U_{jl} \tag{13}$$

Rewriting U using its column vectors this becomes:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} \sum_{i} V_{im} (X \mathbf{u}_l)_i \tag{14}$$

Rewriting the sum over i as a matrix multiplication:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{ml} M[l]_{nm} (V^T X \mathbf{u}_l)_m \tag{15}$$

Rewriting the sum over m as a matrix multiplication:

$$\tilde{y}_k = \sum_{n} T_{kn} \sum_{l} (M[l] V^T X \mathbf{u}_l)_n \tag{16}$$

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

$$\tilde{y} = T \sum_{l} M[l] V^T X \mathbf{u}_l \tag{17}$$

□

B. Additional discussion

B.1. Mechanistic vs. functional weight learning

Herrmann et al. [18] 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.

Similarly, for our aligned representations, the best-performing layer is a "query" layer in the U-Net's encoder. However, examining the top 10 best-performing layers by in-distribution accuracy reveals that the specific "query" layer chosen is critical, resulting in a 6.6% difference in zero-shot accuracy between the best and second-best layers. Additionally, while two of the top 10 layers are "out" layers and perform well on in-distribution samples, their performance drops sharply on the zero-shot task, causing a rank decrease of five places. Table 5 lists the top 10 layers by in-distribution validation accuracy alongside their zero-shot task results.

Table 5. ***Best performing layers of*** SD_{200}***:*** Rankings differ significantly between in-distribution and zero-shot tasks. Numbers in $(\cdot)$ indicate the amount the layer moved up or down in rank.

Layer Name	**In Dist. ↑ Acc.**	**Zero-shot ↑ Acc.**
down.2.attentions.1.attn2.q	0.974	0.898 (0↑)
up.1.attentions.1.attn2.q	0.958	0.832 (2↓)
up.1.attentions.0.attn2.q	0.952	0.850 (1↑)
up.1.attentions.1.attn2.out.0	0.946	0.665 (5↓)
up.1.attentions.2.attn2.q	0.944	0.822 (1↓)
up.1.attentions.2.attn2.out.0	0.920	0.641 (5↓)
down.2.attentions.0.attn2.q	0.916	0.830 (2↑)
up.2.attentions.2.attn2.k	0.872	0.735 (0↑)
mid.attentions.0.attn2.q	0.866	0.848 (6↑)
up.2.attentions.1.attn2.k	0.830	0.760 (3↑)

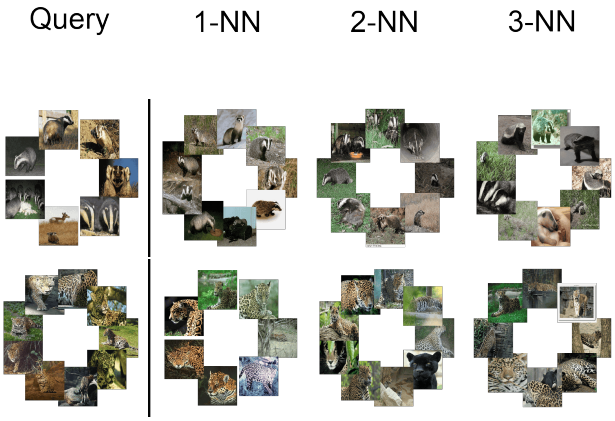


Figure 8. ***Additional model retrieval results:*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

B.2. Self-supervised learning vs. aligning representations

Here, we align model weights with existing representations. While weight-space self-supervised (SSL) learning [40, 41, 43] do not depend on external representations, they typically require carefully crafted augmentations and priors. Designing such augmentations for model weights is non-trivial as key nuisance factors are still being identified. We hope our work accelerates research on new weight-space SSL methods.

B.3. Other weight-space learning tasks

In this paper we focused on predicting the categories in a model's training dataset. However, many more weight-space learning tasks exists. As demonstrated in Prop. 1, our probing formulation is equivalent to the weight formulation, suggesting that ProbeX can potentially perform any task achievable by other mechanistic approaches. Since our focus has been on predicting the model's training dataset categories and their connection to text-based representations, extending ProbeX to these additional tasks is left for future work.

C. Mixture-of-Experts router

As described in Sec. 4.4, when handling Model Graphs with multiple Model Trees, we use a mixture-of-experts approach. This involves first learning a routing function and then training a separate ProbeX model for each Model Tree.

To implement the routing function, we perform hierarchical clustering on the ℓ_2 pairwise distances between models in the Model Graph. By calculating distances for a single model layer, this stage is significantly accelerated, enabling us to cluster Model Graphs with up to 10,000 models in under 5 minutes. Once clustering is complete, the routing function assigns each model to the nearest cluster based on ℓ_2 distance. The number of Model Trees is determined using the dendrograms produced by hierarchical clustering. We

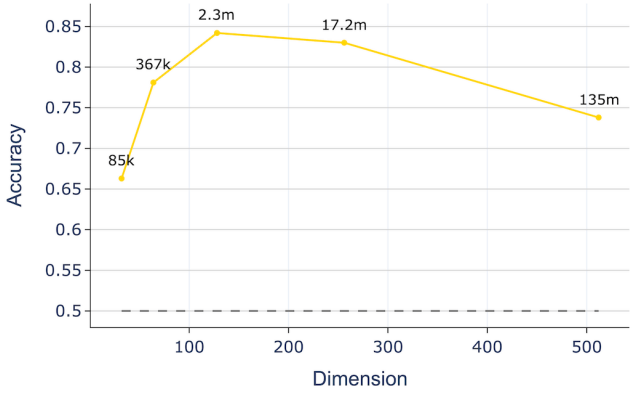


Figure 9. $r_{(\cdot)}$ ***dimension ablation:*** We ablate the effect of changing the dimension of all r_U, r_V and r_T jointly. We can see that beyond some point the performance drops.

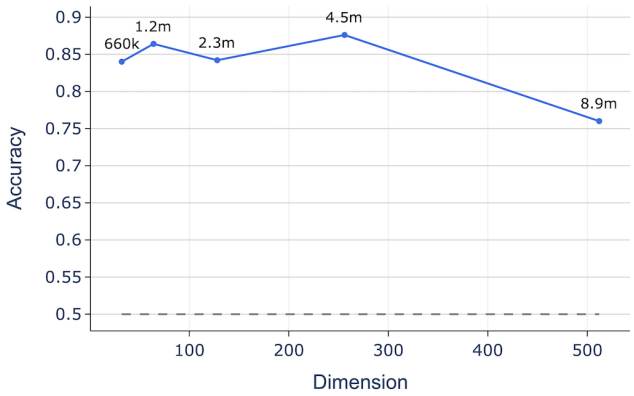


Figure 10. ***Number of probes (r_U) ablation:*** We fix r_V and r_T to 128 and change the number of probes (r_U). We can see that too many probes decreases the performance.

use the `scipy` [48] implementation with default hyperparameters.

In practice, this simple routing function achieved *perfect accuracy* every time.

D. Additional model retrieval results

In Sec. 6.2.2, we presented results for the task of model retrieval. Here, we provide results for *all* held-out models in SD_{200}. These results are not cherry-picked, and each model is visualized using the full set of images that were used for its fine-tuning. In Fig. 8, we display two additional results, in Figs. 25 to 27 present the remaining results.

E. Additional ablations

We provide additional ablations and expand on the ones from the manuscript.

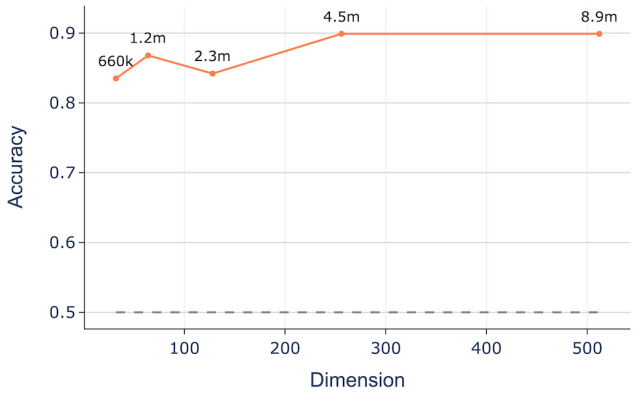


Figure 11. ***Probe dimension (r_V) ablation:*** We fix r_U and r_T to 128 and change the probe dimension (r_V). We can see that even a small probe dimension already results in good performance and that increasing it does not help beyond some point.

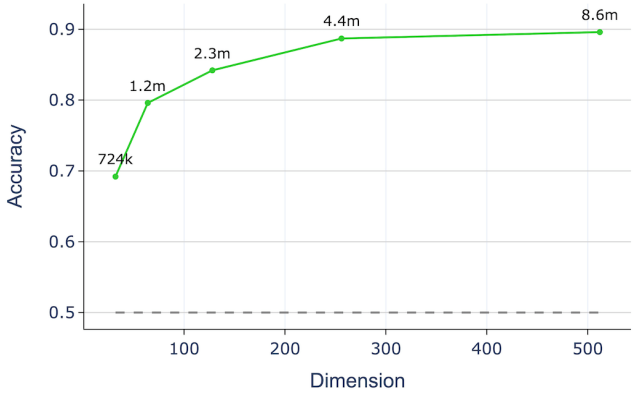


Figure 12. ***Encoding dimension (r_T) ablation:*** We fix r_U and r_V to 128 and change the encoding dimension (r_T). We can see that the size of the model encoding plays an important role in the performance of our method.

E.1. r_U, r_V, r_T size ablation

We ablate the effect of the dimensions r_U, r_V, and r_T using the Sup. ViT Model Tree. We begin by examining the impact of jointly increasing all dimensions. As shown in Fig. 9, increasing the size improves performance up to a point (128), after which performance begins to decline. When jointly adjusting all dimensions, the larger model size appears to be responsible for this drop. However, when we vary each dimension independently while fixing the other two at 128, we observe a different pattern.

Starting with the number of probes (r_U), as shown in Fig. 10, increasing the number of probes has minimal effect on performance until a threshold (256), beyond which performance drops significantly. This decline may explain the performance drop in Fig. 9, even without an extreme increase in the parameter size.

In Fig. 11, we observe that changing the dimension of the

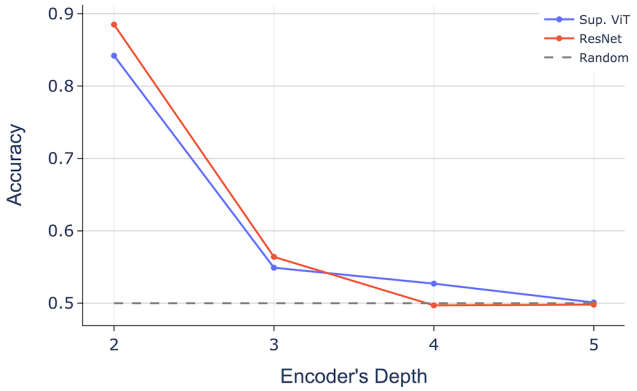


Figure 13. ***Deeper ProbeX encoder ablation - discriminative***

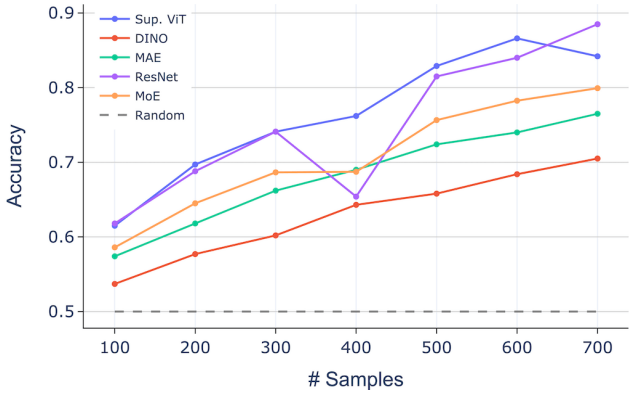


Figure 14. ***Dataset size ablation***

probes (r_V) has little impact on performance. Lastly, Fig. 12 shows that increasing the dimension of the encoding (r_T) has a dramatic effect, significantly improving performance.

E.2. Deeper ProbeX encoders

Here, we evaluate whether deeper, non-linear ProbeX encoders outperform our single hidden-layer encoder. Specifically, we stack additional dense layers followed by non-linear activations and assess their performance. This experiment is conducted for each architecture in the Model-J dataset (i.e., ViT, ResNet, and Stable Diffusion). As shown in Figs. 13 and 15, deeper encoders tend to overfit, leading to reduced performance.

E.3. Dataset size

We examined the effect of dataset size on accuracy. Indeed, in Fig. 14 we see that as discussed in the motivation, models that belong to the same Model Tree have positive transfer.

E.4. Text encoder

We ablate whether our success in aligning model weights with CLIP rerepresentations is due to the fact Stable Diffusion was originally trained with CLIP. We perform the

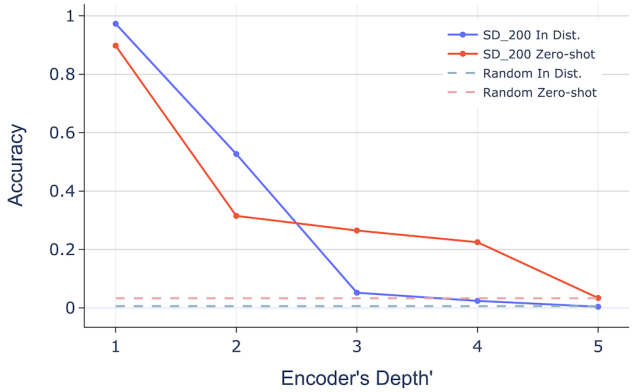


Figure 15. ***Deeper ProbeX encoder ablation -*** SD_{200}

Table 6. ***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**

zero-shot experiment using SD_{200} and the following text encoders. The results in Tab. 6, suggest that while CLIP performs best, our approach remains effective across different text encoders. This shows the robustness of ProbeX to the choice of text backbone.

F. Hugging Face Model Graph analysis

Our presented statistics regarding Model Trees are based on the "hub-stats" Hugging Face dataset[3]. This dataset, maintained by Hugging Face, is automatically updated daily with statistics about Hugging Face models, datasets, and more. We used a version from late September 2024, when there were "only" about 800,000 models hosted on Hugging Face. We utilized the `base_model` property from model cards and aggregated based on it. However, since not all models on Hugging Face use this property, these statistics are not 100% accurate and may contain some bias. Additionally, Fig. 1 also uses the "hub-stats" dataset and is based on the graphs shown at https://huggingface.co/spaces/cfahlgren1/hub-stats.

[3] https://huggingface.co/datasets/cfahlgren1/hub-stats

Table 7. ***Hyperparameters overview - Discriminative split***

Name	Value
lr	$[5e-4, 3e-4, 1e-4, 9e-5, 7e-5, 5e-5, 3e-5]$
lr_scheduler	linear, cosine, cosine-with-restarts, constant, constant-with-warmup
weight_decay	$[5e-2, 3e-2, 1e-2, 9e-3, 7e-3, 5e-3]$
epochs	$[2-9]$
random_crop	T,F
random_flip	T,F
batch_size	64
fine-tuning type	Full Fine-tuning

Table 8. ***Hyperparameters overview - generative Split***

Name	Value
# images	[5-10]
lr	$[5e-4, 3e-4, 1e-4, 9e-5, 7e-5, 5e-5, 3e-5]$
prompt_template	a photo of a, a picture of a, a photograph of a, an image of a, cropped photo of the, a rendering of a
base_model	[SD1.2, SD1.3, SD1.4, SD1.5]
steps	$[450, 500, 550, 600, 650, 700]$
fine-tuning type	LoRA
random_crop	T,F
rank	16
batch_size	1

G. Dataset details

Existing weight-space learning datasets and model zoos [19, 42] primarily consist of models that are randomly initialized. This means that each model in such datasets serves as the root of a distinct Model Tree containing only that model. As demonstrated in Sec. 3, learning from such Model Graphs is significantly more challenging, highlighting the need for our approach of learning within Model Trees. Furthermore, existing datasets primarily consist of small models, typically with only thousands of parameters per model. *As such, we cannot utilize the existing and established weight-space learning datasets.*

To address this, we introduce the Model Jungle dataset (Model-J), which simulates the structure of public model repositories. Each of our fine-tuned models is trained using a set of hyperparameters sampled uniformly at random. Discriminative models share the same set of possible hyperparameters, summarized in Tab. 7. Generative models, in contrast, use a different set of hyperparameters detailed in Tab. 8. Notably, in the generative split, our Model Trees have multiple levels of hierarchy, as models were fine-tuned from SD1.2 to SD1.5. This structure is designed to simulate public model repositories, where Model Trees often exhibit multiple levels of hierarchy. In Figs. 17 and 19 to 21 we provide a summary of the test accuracy the models in the discriminative split converged to. In Figs. 18 and 22 to 24 we plot these accuracies as a function of the model's learning rate.

For the discriminative split, we use the following models as the Model Tree roots taken from *Hugging Face*:

- https://huggingface.co/google/vit-base-patch16-224
- https://huggingface.co/facebook/vit-mae-base
- https://huggingface.co/facebook/dino-vitb16
- https://huggingface.co/microsoft/resnet-101

Table 9. ***Model Jungle dataset summary.*** We train over 14,000 models, covering different architectures, tasks and model sizes. Each model uses randomly sampled hyper parameters

Name	FT Type	Task	Size	# Classes
DINO	Full FT	Att. classification	1000	50/100
MAE	Full FT	Att. classification	1000	50/100
Sup. ViT	Full FT	Att. classification	1000	50/100
ResNet	Full FT	Att. classification	1000	50/100
SD_{200}	LoRA	Fine-grained	5000	200
SD_{1k}	LoRA	Few shot	5000	1000

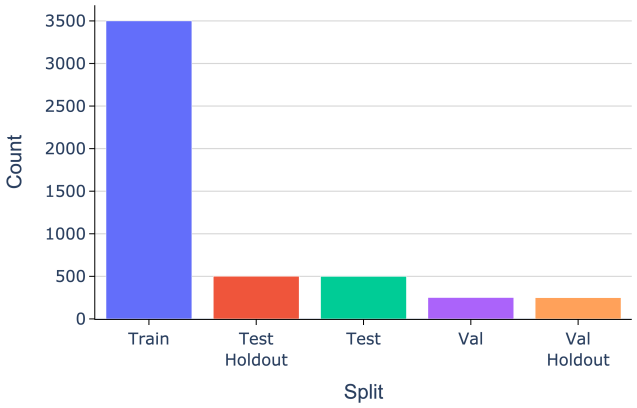


Figure 16. ***Distribution of splits in the generative split***

H. Implementation details

H.1. Experimental setup

We use the Model-J dataset presented in Sec. 5, split into 70/10/20 for training, validation, and testing. Given the significant variation in results between layers, we train ProbeX for 500 epochs on each layer and select the best layer and epoch based on the validation set. We use the Adam optimizer with a weight decay of $1e{-}5$ and a learning rate

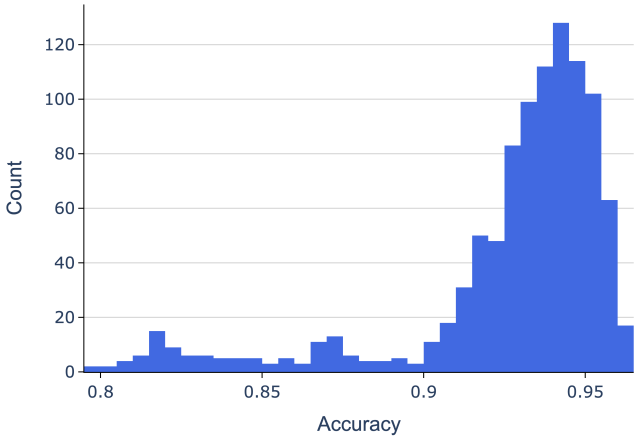


Figure 17. ***Sup. ViT - Test accuracy distribution***

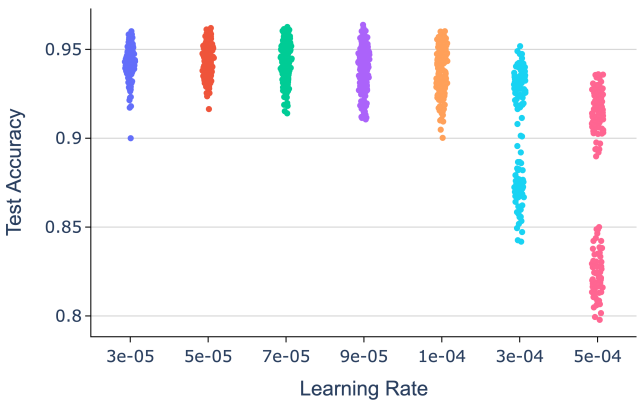


Figure 18. ***Sup. ViT - Effect of learning rate on test accuracy***

of $1e{-}3$. The number of probesm probe dimensions, and encoder dimension are set to $r_U = r_V = r_T = 128$.

H.2. Baselines

H.2.1. Neural Graphs baseline

As mentioned in the Sec. 6, we attempted to use [27] as a baseline but were unable to scale the method to models in our dataset. Here, we provide additional details about this attempt. Neural Graphs [27] is a graph-based approach that treats each bias in the network as a node and each weight as an edge. These methods scale quadratically with the number of neurons, leading to computational challenges when applied to larger models. To address this, we adapted the Neural Graphs approach to the single-layer case, but even in this simplified scenario, it required a relatively low hidden dimension to run on a 24GB GPU. Since this baseline yielded near-random results in our experiments with discriminative models, we chose not to include its results in the tables.

H.2.2. StatNN

For the discriminative split, we used StatNN as a baseline by training XGBoost on the StatNN features. To compare StatNN in the case of text-aligned representations, we replaced XGBoost with an MLP, allowing the baseline to be trained with the same contrastive objective used for ProbeX. We developed two StatNN variants: i) $StatNN_{Linear}$: A single linear layer trained on top of the StatNN features. ii) $StatNN_{MLP}$: A deeper architecture designed to match the parameter count of our method.

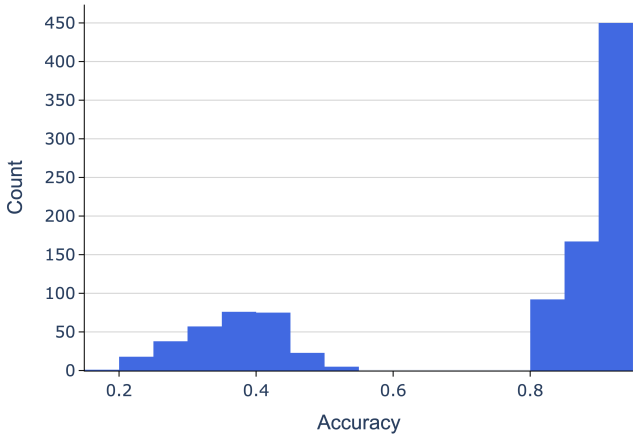


Figure 19. ***DINO - Test accuracy distribution***

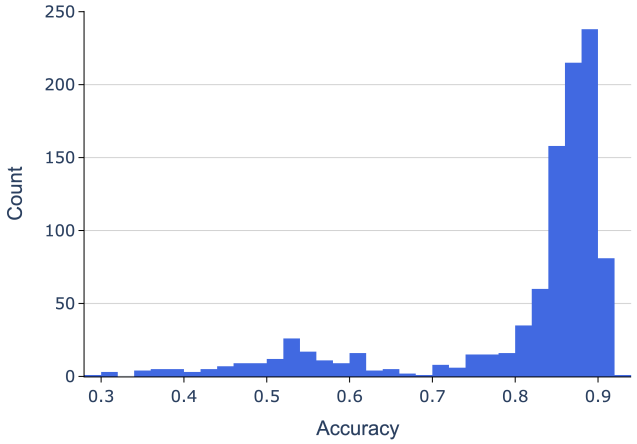


Figure 20. ***MAE - Test accuracy distribution***

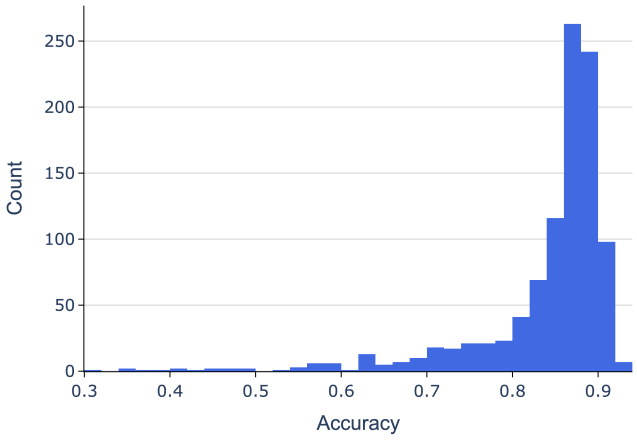


Figure 21. ***ResNet - Test accuracy distribution***

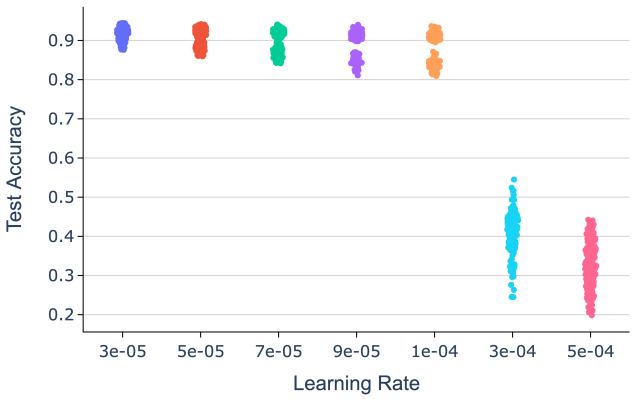


Figure 22. ***DINO - Effect of learning rate on test accuracy***

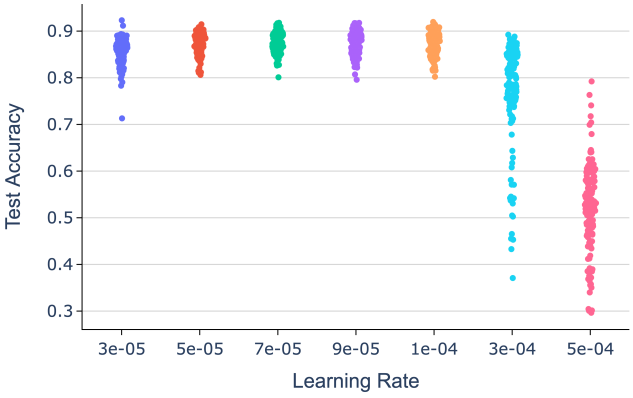


Figure 23. ***MAE - Effect of learning rate on test accuracy***

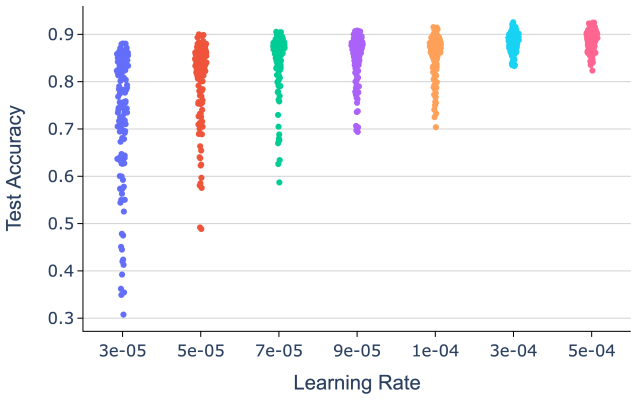


Figure 24. ***ResNet - Effect of learning rate on test accuracy***

Figure 25. ***Additional non-cherry picked retrieval results (1/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

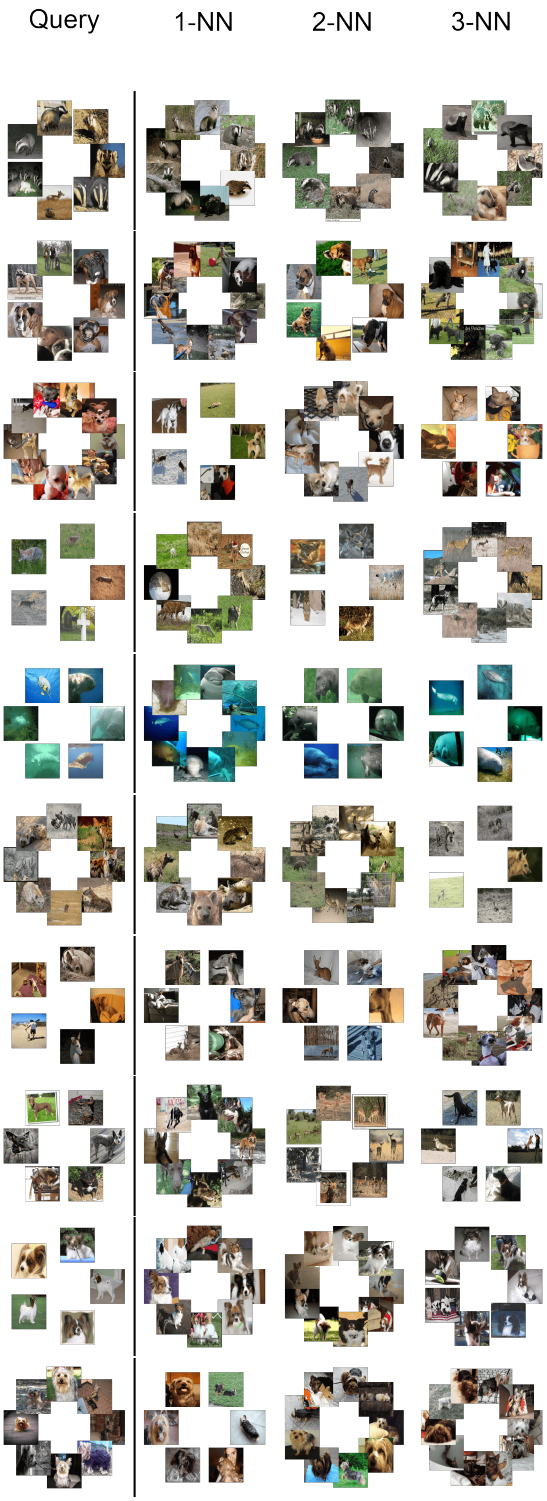


Figure 26. ***Additional non-cherry picked retrieval results (2/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

Figure 27. ***Additional non-cherry picked retrieval results (3/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.