PLeaS — Merging Models with Permutations and Least Squares

Anshul Nasery[†*] Jonathan Hayase[†*] Pang Wei Koh[†◇] Sewoong Oh[†]

[†] University of Washington [◇] Allen Institute for AI

Abstract

The democratization of machine learning systems has made the process of fine-tuning accessible to practitioners, leading to a wide range of open-source models fine-tuned on specialized tasks and datasets. Recent work has proposed to merge such models to combine their functionalities. However, prior approaches are usually restricted to models that are fine-tuned from the same base model. Furthermore, the final merged model is typically required to be of the same size as the original models. In this work, we propose a new two-step algorithm to merge models—termed PLeaS—which relaxes these constraints. First, leveraging the Permutation symmetries inherent in the two models, PLeaS partially matches nodes in each layer by maximizing alignment. Next, PLeaS computes the weights of the merged model as a layer-wise Least Squares solution to minimize the approximation error between the features of the merged model and the permuted features of the original models. PLeaS allows a practitioner to merge two models sharing the same architecture into a single performant model of a desired size, even when the two original models are fine-tuned from different base models. We also demonstrate how our method can be extended to address a challenging scenario where no data is available from the fine-tuning domains. We demonstrate our method to merge ResNet and ViT models trained with shared and different label spaces, and show improvement over the state-of-the-art merging methods of up to 15 percentage points for the same target compute while merging models trained on DomainNet and fine-grained classification tasks[1].

1. Introduction

With the widespread democratization of machine learning, there has been a rapid increase in the availability of open-source models trained by the community on specific tasks and datasets. Such specialized models exhibit unique strengths and weaknesses. For example, Code Llama [25] (fine-tuned from Llama-2) is specialized for coding, while Vicuña 1.3 [3] (fine-tuned from Llama-1) is specialized for chat. They have the same architecture but are fine-tuned starting from different pre-trained models: Llama-1 and Llama-2. Such diversity in the combination of pre-training data and fine-tuning tasks will only increase as decentralized marketplaces for models become increasingly more common, *e.g.*, [24], providing practitioners with more options.

This presents an opportunity to combine such specialized models in order to create a single general-purpose model that can handle multiple tasks. Traditional approaches for combining trained models, such as ensembling [7] or domain-specific mixture-of-experts (*e.g.* [13]), take a step towards this goal. However, these methods need to store all the component models at inference time, leading to an increased memory footprint. Practitioners with limited memory capacity cannot use such approaches with high and fixed memory costs, especially when combining large models, deploying to resource-constrained environments, or for applications demanding a memory-performance trade-off.

To this end, recent works [12, 33, 35, 36] have proposed new algorithms tackling this problem of *model merging*. However, their scope is limited to merging models fine-tuned from the *same* pretrained model. Further, some recent works [28] also need access to the *training data* used to fine-tune the component models, which limits their applicability in situations where such data is not available due to, for example, privacy or legal reasons [5]. In this paper, we address the problem of merging models (sharing the same architecture) trained on different datasets starting from *different initializations*. This is motivated by prior work (*e.g.*, [10, 28, 34]), which we compare with in Section 5 for merging ResNet and ViT models. To address the above limitations of prior work in this space, we present PLeaS—an algorithm which adaptively merges models for different inference compute budgets, and can work without using the fine-tuning data of the component models.

PLeaS (short for **P**ermutations and **Le**ast **S**quares) is a two-stage algorithm which works with models having the

*Equal Contribution. Correspondence to anasery@cs.washington.edu

[1]Code open-sourced at https://github.com/SewoongLab/PLeaS-Merging

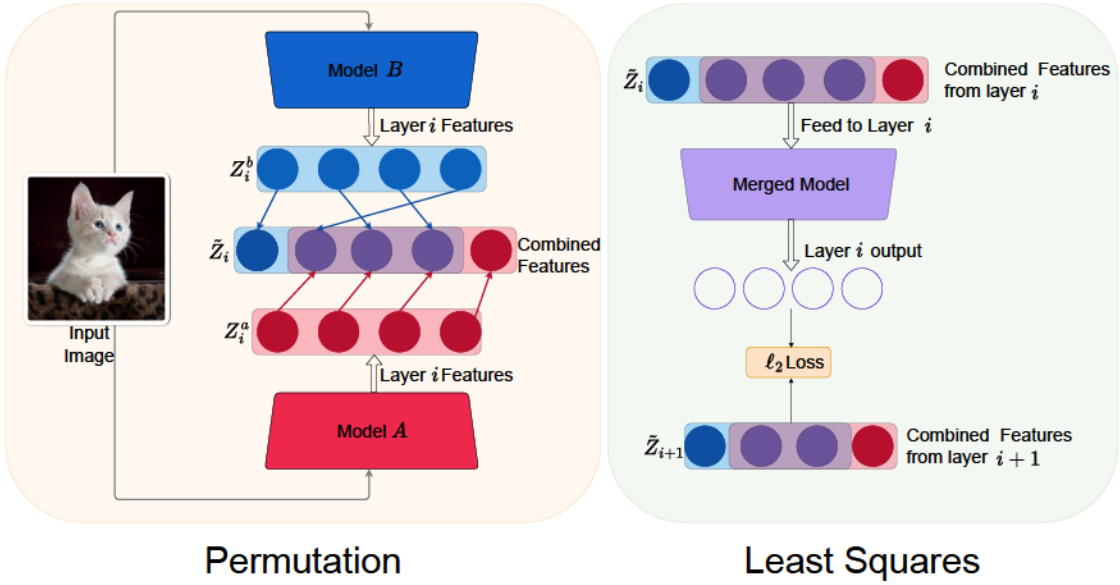

Figure 1. **PLeaS is a two-step algorithm for merging models**: The first step (left) finds layer-wise **P**ermutations to match features across models to compute combined features $\tilde{Z}_i$. Features which are similar are merged, while those which are dis-similar are kept separate. The number of features to be merged depends on the target compute budget, and can be different for each layer. The second step of PLeaS (right) aims to find weights of the merged model which can map the combined features of layer i (i.e., $\tilde{Z}_i$) to those of layer $i+1$ (i.e., $\tilde{Z}_{i+1}$) appropriately by solving layer-wise **Le**ast **S**quares problems for each layer.

same architecture. The first step consists of matching features across the models. We harness the idea of permutation invariance in neural networks to find an appropriate pairing of features. Inspired by the Git Re-Basin [1] algorithm, which is designed for merging two models that are trained on the same data, we introduce a matching algorithm that finds permutations between similar features across models, while keeping dissimilar features separate in the final merged model. This is critical when merging models trained on widely different tasks, since it prevents interference between features while still merging overlapping features. This also gives PLeaS fine-grained control over the width of each layer of the merged model, improving performance over prior work such as ZipIt! [28]. PLeaS can hence flexibly trade-off inference memory/compute and performance according to the deployment requirements.

It has been observed that permutation matching alone suffers from significant performance loss when merging vastly different models, *e.g.* those trained on disparate data [34]. We hypothesize that while permuted features are powerful when ensembled, simply averaging the permuted weights degrades the features of the merged model. This results in the observed decline in performance. Hence, in the second step of PLeaS, we solve a layer-wise Least Squares problem, so that each layer of the merged model mimics the permuted ensemble of features from the corresponding layer of the original models. This produces better representations and superior down-stream performance.

Apart from the target compute budget, PLeaS is hyper-parameter free, making it easy for practitioners to use. A schematic of PLeaS is depicted in Fig. 1.

We empirically demonstrate that PLeaS can outperform prior work in the challenging setting of merging differently initialized models which have been trained on different datasets. We merge ResNet and ViT models fine-tuned on different datasets in Secs. 5.2, 5.3 and 5.5, and find that PLeaS improves upon the state-of-the-art up to 15% with the same merged model size. Our empirical results are on subsets of DomainNet, and on fine-grained classification tasks. PLeaS can also approach the performance of ensemble methods with significantly lower FLOPs (Sec. 5.5).

The proposed approach can be seamlessly extended to the scenario where data from the fine-tuning domains is unavailable. We call this variant PLeaS-free. This variant uses data from publicly available datasets (like ImageNet) to merge models. We demonstrate in Sec. 5.4 that PLeaS-free is competitive with PLeaS, which uses the actual data from the training domains of the component models. This is highly encouraging, as it demonstrates the applicability of PLeaS-free in scenarios where data from the training domains is unavailable due to privacy or commercial reasons.

In summary, our contributions are the following:

- We generalize Git Re-Basin [1] to support partial merging

of corresponding layers of two models (Sec. 4.1). This gives practitioners the freedom to choose the size of the final merged model as per resources available at inference. Investigating this tradeoff is one of the goals in this work, *e.g.*, Fig. 3.

- Motivated by the success of ensemble methods, we propose to assign weights to the merged model's parameters by solving a least squares problem attempting to mimic ensemble methods at each layer (Sec. 4.2). Ablation study for this step is in Fig. 3.
- On a test-bed of multiple datasets, we showcase that `PLeaS` outperforms recent merging methods up to 15 percentage points (Sec. 5) at the same model size. Further, `PLeaS` approaches the ensemble accuracy while using 40% fewer parameters. Finally, even with no data from the training domains, `PLeaS-free` remains competitive with `PLeaS` (Fig. 4).

2. Related works

There has been growing interest in merging models with minimal data and compute overhead. Here, we focus on methods which merge models with the *same* architecture.

Merging models fine-tuned from the same initialization. Several methods aim to merge models in the weight space. Ilharco et al. [12] simply add up *task vectors*, the weight differences of fine-tuned models from the pretrained model, and demonstrate a strong baseline for merging fine-tuned models. Other approaches edit the task vectors based on magnitude of the weights [33, 37] to resolve interference while merging. Some methods aim to find layer-wise [2, 36] or parameter-wise [19] coefficients for merging different task vectors. However, methods that work with task vectors assume that the base pretrained model is shared across the fine-tuned models, and hence they cannot be easily extended to settings where models are fine-tuned from different starting points. A different line of work [14] proposes layer wise distillation, aiming to minimize the sum of the ℓ_2 distances between the activations of the merged model and the original models. However, naively applying this to vastly different models leads to degraded performance, as we show in Sec. 5. Further, these methods do not provide a way to control the size of the merged model. Although not designed for this scenario of merging fine-tunes of a common pre-trained model, `PLeaS` still allows us to achieve significant performance gains when the merged network is slightly larger than the original model (*e.g.* by 20%) as demonstrated in Tab. 2.

Merging from two different initializations. We consider a less restrictive setting, where the models being merged can have different initializations. This has been studied in the literature, and existing works propose weight or activation matching algorithms for this task. Git Re-Basin [1] proposes an algorithm to compute permutation matrices to match the weights of the hidden layers of two or more neural networks. Yamada et al. [34] investigate the usage of permutations to merge models trained on different datasets, however, their study is limited to wide ResNet models on MNIST and CIFAR datasets. These permutation symmetries have also been studied in [4, 20, 27, 34]. A recent work – MuDSC [32] leverages permutation symmetries both in weight space and activation space to merge models better, however, we show that `PLeaS` outperforms this work empirically. Another recent work – ZipIt! [28], tackles a similar problem of merging models fine-tuned on different datasets from different initializations. This work also supports merging models partially by "zipping" some layers of the component models. While this can provide a knob for controlling the size-performance trade-off of the merged model, the empirical performance of their proposed scheme can be improved upon, as we show in Sec. 5.3. On the other hand, our work describes a merging formulation which is more expressive and allows for partial merges with expanded layers to minimize feature interference. Finally, [10] also proposes a method to merge networks layer-wise in a progressive manner, which involves light-weight retraining. However, their method requires domain labeled data at both training and inference time, while we only require unlabeled data and also propose a method using no data from the train domain at all.

Other merging paradigms. Other model merging approaches include mixture of experts [26, 29], selecting experts using test data [18], and sparse expert ensembles [9]. These come with larger compute or memory over-heads, both at inference and training time.

3. Preliminaries

Notation. For simplicity, we describe our method for two L-layered MLPs. However, it can be readily extended to convolutional and residual networks, as we demonstrate in our experiments. Let $\Theta^A = \{W_1^A, W_2^A, \cdots, W_L^A\}, \Theta^B = \{W_1^B, W_2^B, \cdots, W_L^B\}$ be the parameters of two MLPs A, B having the same architecture. We omit the layer-wise bias here for simplicity. Let z_i^A, z_i^B denote the input activations to the i^{th} layer of each network respectively, and d_i denote the dimension of z_i^A, z_i^B. We also define $Z_i^A, Z_i^B \in \mathbb{R}^{d_i \times n}$ to be the activations of a batch of n inputs. Note that $z_1^A = z_1^B = x$, and $z_{L+1}^A = y^A, z_{L+1}^B = y^B$. Finally, let $\{W_i^M : i \in \{1, 2, \cdots, L\}\}$ be the weights of the merged model. We allow the merged model to have varying widths (which can be different from the widths of the base model), depending on the inference resources available.

Background on Git Re-Basin. Our method is inspired by Git Re-Basin [1], which aims to find permutation matrices $\pi = \{P_1, P_2, \cdots, P_L\}$ to permute the weights of model B. The merged model is formed by permuting and averaging the weights, i.e., $W_i^M = (1/2)(W_i^A + P_i W_i^B P_{i-1}^T)$.

Ainsworth et al. [1] propose to estimate the permutation matrices by directly optimizing the average similarity between the permuted weights of model B and the original weights of model A. This *weight matching* greedily finds a solution to the following sum of bilinear assignment problems,

$$\underset{\pi=\{P_i\}_{i=1}^{L}}{\arg\max} \sum_{i=1}^{L} \langle W_i^A, P_i W_i^B P_{i-1}^T \rangle \ ,$$

where P_0 is defined to be the identity matrix. This has an advantage of not requiring any data to solve the optimization, but an optimal solution is computationally intractable. Instead, when some samples are available to the optimizer, Ainsworth et al. [1] propose a computationally efficient alternative called *activation matching*, which solves the following optimization problem:

$$P_i \ \in \ \underset{P \in S_{d_i}}{\arg\min} \ \|Z_i^A - P Z_i^B\|_F^2 \ .$$

Here, S_{d_i} refers to the set of permutation matrices of size $d_i \times d_i$. Computing the activations Z's require samples from the data. However, this optimization can be efficiently solved separately for each layer.

4. Method: `PLeaS`

We call our approach to model merging `PLeaS`. We harness permutation symmetries to match features between two models, inspired by Git Re-Basin [1]. We extend this method to allow for partial merging of models, where each layer can have a different number of merged neurons.

We then compute the weights of the final merged model by solving layer wise least squares problems to ensure that activations of the merged model resemble the permuted activations of the original models.

4.1. Extending Git Re-Basin to partial merging

Note that in Git Re-Basin, two models are averaged (after permuting one model) and hence the dimension of the parameters of the merged model is the same as the corresponding parameters of the base models. However, when the networks A, B are trained on different datasets, not all features might be compatible across models. These features may interfere destructively if merged, leading to degraded performance. Further, these incompatible features may need to be retained in the merged model in order to make accurate predictions on both tasks. Merging all nodes in every layer discounts this possibility, leading to performance degradation, as we show in Fig. 4b. To this end, we aim to merge features which are similar across the two models, while keeping those which are very different as separate features in the merged model. We hence propose a framework for partially merging model features by leveraging permutations.

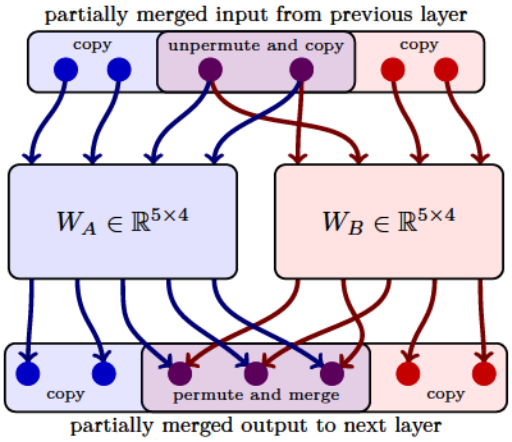

Figure 2. **Partial merging with permutations**: We show the construction of the 7×6 weight matrix W_i^m from two weights of size 5×4 in the first step of `PLeaS`. The merged inputs are copied and unpermuted to approximate the original inputs. Then we apply both weight matrices separately. Finally, we pair up the merged outputs and average the pairs. Since all operations used are linear, we can fuse them to construct W_i^m using a single linear layer.

Given a permutation matrix P_i, we select k_i indices from $[d_i]$ for which the distance between the features of model A and the permuted features of model B for layer i is the smallest. These k_i features are merged, while other features are retained separately in the final model. In particular, we find a subset J_i satisfying

$$J_i \in \underset{\{J: J \subseteq [d_i], |J| = k_i\}}{\arg\min} \ ||Z_{J,i}^A - (P_i Z_{:,i}^B)_J||_F^2 \ .$$

This is simple to implement: retain the indices with the smallest k_i distances between the (permuted) activations. For weight matching, we can retain the indices with the largest similarity between the (permuted) weights for each layer. The size of W_i^M is then increased to $(2d_i - k_i) \times (2d_{i+1} - k_{i+1})$ in exchange for improved performance. This partial merging scheme is illustrated in Fig. 2. One goal of this paper is to investigate this trade-off between size and the performance of the merged model.

Note that the ratio k_i/d_i can be chosen independently for each layer. In Appendix A.3, we propose a scheme to find a configuration of these ratios subject to a target compute/memory budget B; this optimizes a proxy of the downstream performance without using any validation data from the target domain, and is used in all our experiments. The permutation matrices P_i are computed using the activation matching strategy from Git Re-Basin. In Sec. 5.4 and the Appendix, we compare this with using the weight matching strategy, which we call `PLeaS-Weight` .

We would like to emphasize that our partial merging formulation is different from ZipIt! [28], since we can assign any amount of compute between $1\times$ and $2\times$ (relative to

the original layer's compute) to each layer independently. ZipIt! on the other hand, assigns exactly $1\times$ compute to a prefix of layers and $2\times$ to the rest. In Sec. 5.3, we show that this flexibility in our formulation leads to better empirical performance.

4.2. Permuted least squares

Suppose, for example, that the target merged model has the same architecture and size as each of the base models. Once the permutations, P_i, have been computed, we propose optimizing the weight matrices of the merged model by solving the following least-squares problem:

$$W_i^M \in \arg\min_W \|(Z_i^A + P_i Z_i^B)W - (Z_{i+1}^A + P_{i+1} Z_{i+1}^B)\|^2, \tag{1}$$

independently for each layer $i \in [L]$. This is motivated by the impressive performance of the ensemble method (*e.g.*, [34] and Sec. 5), which retains two separate models and only averages the (permuted) activations at the last layer (pre-softmax): $\tilde{z}_{L+1} = z_{L+1}^A + z_{L+1}^B$. We aim to have our merged model approximate such activations. We inductively assume that the first $i-1$ layers are properly merged. Hence, the ensemble of the permuted features (of the i^{th} layer) of the component models can be well approximated by the activations at the input of the i^{th} layer of the merged model. We denote the ensembled features by $\tilde{Z}_i = (Z_i^A + P_i Z_i^B) \in \mathbb{R}^{d_i \times n}$. The goal of the above optimization is to match the ensembled activation of the next layer, $\tilde{Z}_{i+1} = (Z_{i+1}^A + P_{i+1} Z_{i+1}^B)$, with a linear transform of the input ensemble: $\tilde{Z}_i W$. We empirically validate this choice to use a permuted ensemble of features to optimize the weights of the merged model in Tab. 4 in Appendix B.1, where we compare with alternatives to Eq. (1).

This second step of `PLeaS` is similar to feature distillation. However, the key novelty arises from averaging the permuted features for transferring knowledge from multiple models. This is critical for accurate prediction. To show this, we compare `PLeaS` against RegMean [14], which optimizes an objective similar to Eq. (1) without the permutations and averaging, i.e. this method merges models by minimizing $\|Z_i^A W - Z_{i+1}^A\|^2 + \|Z_i^B W - Z_{i+1}^B\|^2$. As we show in Sec. 5, RegMean performs poorly compared to `PLeaS`. Apart from the inference computation budget for the final model, `PLeaS` is completely hyperparameter free.

Note that the second step of `PLeaS` is fully compatible with the partial merging of Sec. 4.1 as well: we can directly set the values of W_i^M corresponding to the unmerged features to be the respective values of W_i^A and W_i^B.

While the objective in Eq. (1) can be minimized in closed form using Ordinary Least Squares (OLS), we practically implement it using gradient descent for ease of use with convolution layers. Given that the objective is convex if computed layer-wise, the weight matrices W_i^M converge in relatively few (less than 100) steps of gradient descent. Further, we solve this optimization independently for each layer, so it can be efficiently parallelized.

4.3. Data requirements of `PLeaS`

`PLeaS` has two steps – the first step finds permutations to match features using weight or activation matching and the second step computes weight matrices to mimic the ensemble of the merged features more closely. In order to compute these features, one could use the data from the training domains, however, this may not be feasible for privacy or commercial reasons. Hence, we propose an alternative scheme–dubbed `PLeaS-free`–which uses a general vision dataset, like ImageNet, to compute the activations of the component models. These activations are then used to merge domain specific models without requiring any data from the training domains. In Sec. 5.4, we show that `PLeaS-free` suffers a minimal performance penalty compared to `PLeaS`, suggesting wider applicability in low/no data settings.

5. Experiments

We show the effectiveness of our method in merging models fine-tuned on different datasets with different initializations. We investigate the following research questions:

1. How does `PLeaS` compare with prior work in merging models to produce a model of the same size (Sec. 5.2)?
2. What is the trade-off between size of the merged model and its performance for `PLeaS` (Sec. 5.3)?
3. How does `PLeaS` perform if one does not have access to the training data of the models being merged (Sec. 5.4)?
4. How does `PLeaS` perform while merging models fine-tuned from the same initial model (Sec. 5.5), or different models trained on the same data (Appendix B.2)?
5. What is the impact of varying the objective in Eq. (1) on the performance of `PLeaS` (Appendix B.1)?

5.1. Experimental Setup

To obtain models for merging, we fine-tune ImageNet pre-trained ResNet models on other smaller datasets. We merge models trained (from *different* initializations) on different data domains in a pair-wise fashion, and compute the accuracy of the merged model on both the data domains. For each domain, we average the accuracy across all such pairs.

5.1.1. Datasets

Since we are dealing with classification models, we consider two sets of datasets (with shared and different label spaces) for training and merging models.

Datasets with a shared label space. We fine-tune ImageNet pre-trained ResNet-50 models on four different domains of the DomainNet [23] dataset: Clipart, Infograph, Painting and Real. These datasets share a label space with 345 classes and comprise of images in various styles.

Method	FLOPs	Memory	Same Label Space					Different Label Spaces				
			Clip	Info	Paint	Real	Avg	CUB	Pets	Dogs	NABird	Avg
MoE*	1.1×	2.1×	69.1	36.1	65.7	78.0	62.2	81.1	92.7	83.1	75.8	83.2
Ensemble*	2×	2×	63.6	30.3	61.0	74.7	57.4	80.5	92.8	82.1	76.1	82.9
Simple Avg [12]	1×	1×	1.2	0.8	1.9	2.1	1.5	7.1	19.2	9.2	4.7	10.1
RegMean [14]	1×	1×	16.6	5.8	10.1	15.8	12.1	42.5	45.1	20.2	37.1	36.2
ZipIt! [28]	1×	1×	26.9	12.2	27.1	37.4	25.9	67.5	83.6	60.0	56.3	66.9
Git Re-Basin [1]	1×	1×	18.2	7.8	18.8	26.5	17.8	66.2	80.2	62.6	59.4	67.1
MuDSC [32]	1×	1×	34.0	14.3	29.5	45.1	30.7	70.1	82.5	63.2	58.2	68.5
PLeaS (Ours)	1×	1×	**41.7**	**16.9**	**40.8**	**55.1**	**38.6**	**75.2**	**85.0**	**69.6**	**69.7**	**74.9**

Table 1. **Merging pairs of models trained on different datasets**: For each pair of datasets, we merge models and compute the final accuracy on the pair. To compute the final accuracy for a dataset, we average the accuracies across the pairs that the dataset is a part of. We report the accuracies of the merged models for the Same Label Space setting, and a linear probe accuracy on the representations of the merged model for the Different Label Space setting. * Note that here the merged models (bottom six) have the same size as the original, but the MoE and ensemble have a size twice the original.

Datasets with different label spaces. We fine-tune models on CUB [31], NABirds [30], Oxford-IIIT Pets [21] and Stanford Dogs [16] datasets, and merge them up to the penultimate layer. Since the label spaces of these datasets are different, we aim to evaluate the representations of the penultimate layer of these merged models by training a linear probe on top of the representations. We average the results in the same manner as for DomainNet, and report the performance of different methods in Tab. 1. In Appendix B.4, we follow the setting of [28], using task specific heads from the original models to compute the accuracy of the merged model, which requires knowing the domain of each test data point.

5.1.2. Baselines

We compare our method against prior works including Git Re-Basin [1], Simple Averaging [12], RegMean [14], ZipIt! [28] and MuDSC [32]. Note that RegMean has similar data and compute requirements as PLeaS, and ZipIt! also supports partial merging of models like PLeaS. We also consider two practical upper bounds — training a router model based Mixture of Experts model (MoE), and ensembling the predictions (or activations) of the original models. The former requires storing both models and running one of them at inference in addition to the overhead of the router, hence having 1.1× FLOPs and 2.1× memory requirements, while the latter requires running both the models in parallel, and hence has 2× FLOPs and memory requirements. We find that the performance of the ensemble and MoE models is close to the best performance of a single model on its training dataset.

For each task, we also report results for Permutations, which is the model obtained by weight averaging the component models after applying the permutations obtained from the first step of PLeaS. Following the recommendation of REPAIR [15], we recompute the batch-norm parameters of the model after merging for all methods. We run each merging experiment for three different seeds, and across two different initial models. We find that inter-run variation in performance is low, with the standard deviation usually being less than 1%. We report disaggregated results along with these standard deviations in Appendix B.3.

5.2. Merging for the same size

In Tab. 1, we report the domain-wise performance for merged models at 1× size of the original model for different methods. On DomainNet, we observe that PLeaS outperforms the previous state-of-the-art method, MuDSC by over 8% on average. Similarly, On datasets with different labels spaces, PLeaS is better than MuDSC by almost 6.5% on average. Further, PLeaS vastly outperforms RegMean, a method which which performs Least Squares without permutations and has a similar computation overhead as PLeaS. This indicates that permuting the features before performing Least Squares is a crucial step which helps PLeaS produce better models. Finally, These results also show the utility of the second step of PLeaS, as it boosts the performance over Git Re-Basin by a large amount, nearly doubling the accuracy on DomainNet.

5.3. Exploring the model size-accuracy tradeoff

We seek to demonstrate the effect of partial merging on the performance of the merged models. To do this, we merge models pairwise as above, and report the average of the performance of these merged models on their respective component domains for different sized ResNet models in Fig. 3.

We find that PLeaS consistently outperforms ZipIt! at various compute/memory budgets and for all model scales. The gains are particularly striking for lower memory bud-

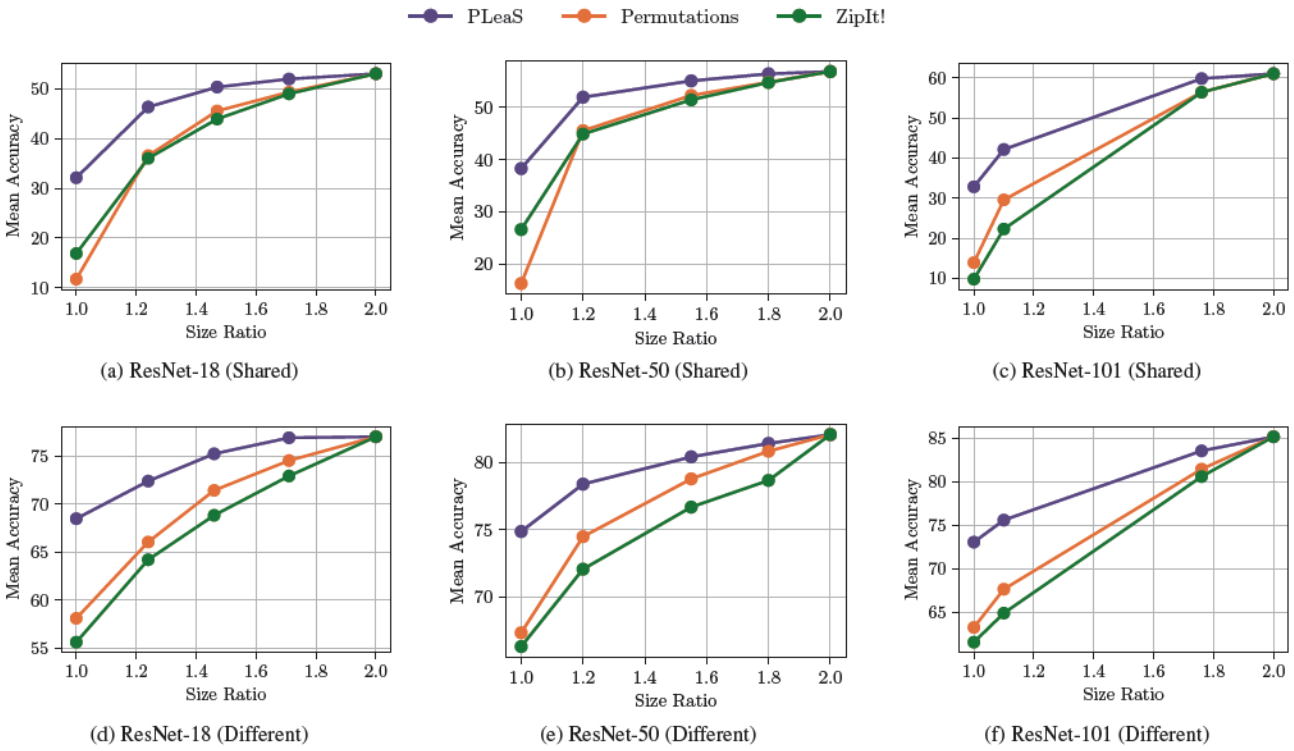

Figure 3. **Memory-Performance trade-off for merged models**: We merge pairs of models fine-tuned on different datasets, and compute the average performance across all four datasets for two settings: datasets with a shared label space (top) and datasets with different label spaces (bottom). Plotting average accuracy against the final merged model size, we find that PLeaS dominates the state-of-the-art methods.

gets, where PLeaS outperforms ZipIt! by up to 10% for ResNet-50 (Fig. 3b). The power of partial merging is also observed from these results, as one can see that increasing the flops by just 20% leads to massive improvements in the accuracies in the shared label settings. These results also provide evidence of the effectiveness of our partial permutation scheme — permutations can outperform ZipIt! at intermediate model budgets by up to 6% (*e.g.* for ResNet-101 with shared label spaces in Fig. 3c). We posit that this is because we can assign a non-uniform width multiplier across the layers of the merged model, which is important for larger models and those which are trained on disparate domains. As expected, the performance gap closes as the relative size of the merged models increases.

5.4. Does PLeaS need data from the training domains?

To investigate the data requirements of our method, we compare the performance of PLeaS and PLeaS-free when merging ResNet-50 models. We also compare the effect of using weight matching to find the permutations for PLeaS, and we term this variant as PLeaS-Weight. Note that this variant uses data only for the Least Squares step. The performance-model size tradeoff is reported in Figs. 4a and 4b for shared and different label spaces.

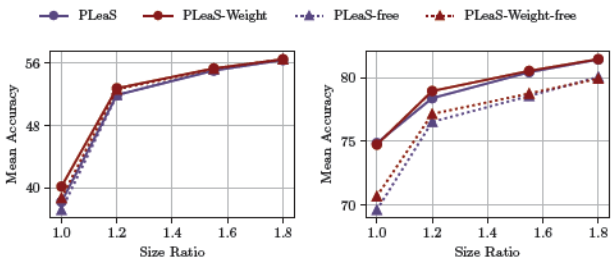

Figure 4. **Investigating the data requirement of PLeaS** : We run PLeaS and PLeaS-Weight using data from the actual domains or ImageNet (indicated by the suffix free) for both the Shared label space (Fig. 4a) and Different label spaces (Fig. 4b) settings for ResNet-50. We plot the average accuracy across all datasets against the relative size of the output model. We find minimal performance drops for PLeaS-free .

We find that PLeaS-free retains a similar performance when using ImageNet instead of the actual domain data for merging models on DomainNet, achieving a drop of less than 1% in accuracy at 1× model size. There is al-

Method	Size	Clipart	Infograph	Painting	Real
Simple Avg	1.0	58.2	28.9	55.7	70.2
RegMean	1.0	58.9	29.0	57.4	71.8
MuDSC	1.0	57.8	55.8	55.1	67.6
PLeaS	1.0	59.6	29.5	58.0	72.0
PLeaS	1.2	64.2	31.9	61.8	75.9
Ensemble	2.0	64.4	32.0	62.0	76.1

Table 2. **Merging models fine-tuned from the same initialization**: We merge pairs of models fine-tuned from the same base model and compute the average accuracy across all pairs of domains for each dataset in the Shared Label Setting. We find that PLeaS can approach the performance of the ensemble while having a 1.2× sized merged model.

most no drop at higher sizes of the merged model. Notably, even on the more difficult task of merging models with different label spaces, using ImageNet data for computing activations can perform competitively to using the actual data: linear probing on the representations from PLeaS-free performs within 2% of the PLeaS at 1.2× model size, and the gap is less than 4% at 1× model size. This result is particularly encouraging, since it extends the practical applicability of PLeaS-free to scenarios where data from the training domains may not be available. Note that while we must use data from the actual domains for linear probing, i.e. to assess the quality of the representations, we do not use it for actually merging the models. We also find that PLeaS-Weight performs similarly to PLeaS in the both the shared and different label space settings for ResNet-50. Further, PLeaS-Weight is less affected when ImageNet data is used, since the permutations computed by PLeaS-Weight do not depend on the data.

5.5. Merging models with the same initialization

In Tab 2, we evaluate the performance of our method for merging ResNet-50 models fine-tuned from the same starting model. We compare against simple average (Task Vectors), MuDSC, and RegMean and find that the performance is similar across methods, with PLeaS being slightly better than the baselines. In fact, task vectors is an effective baseline here. However, we note that 20% extra parameters in the merged model can lead to closing the gap between the ensemble and the merged model produced by PLeaS, demonstrating the need for flexible merging methods.

5.5.1. Merging ViTs

In Tab. 3, we present results of merging CLIP style ViT models. In accordance with prior work [32], we merge models starting from the same initialization. We consider two settings – the different label space setting described in Sec. 5.1 and CIFAR50+50 from [28, 32]. In the latter, the

Method	CIFAR-50+50		Other Datasets			
	Joint	Avg	NABirds	CUB	Pets	Dogs
Simple Avg	72.6	84.5	7.8	65.9	86.1	60.4
RegMean	72.7	84.7	7.7	66.2	85.1	58.5
MuDSC	72.8	84.9	8.0	66.1	86.1	60.6
PLeaS	73.3	85.1	8.3	66.7	86.5	61.6

Table 3. **Merging ViT models:** We merge pairs of ViT models fine-tuned from the same initialization and report the performance. PLeaS can outperform baselines across datasets.

100 labels from CIFAR-100 are partitioned into 2 sets of 50 labels each, and a ViT is trained on each of these sets (using a CLIP-like loss). More details on the setup are described in Appendix A. The accuracy of the merged model is reported on both these partitions separately by considering only 50 classes at a time (denoted by Avg in the table), as well as on the Joint CIFAR-100 dataset (by considering all 100 classes together). Note that since we use CLIP-like models, we can use the language head directly for classification despite different label spaces. For the other datasets, we follow the protocol from Sec. 5.1.

We observe that PLeaS boosts the performance slightly over the baselines. For example, it increases performance by 0.7% on CIFAR-100 over Task Vectors. PLeaS also out-performs MuDSC by around 0.6% on the larger datasets. While these gains are small, they are non-trivial and are more than the boost reported by the previous state-of-the-art method (MuDSC). We believe this presents an exciting opportunity to further study the symmetries and invariances in transformers in order to merge them better.

6. Limitations

The scope of this study is limited to merging models with the *same* architecture, and applying PLeaS to merge different architectures could be an interesting future direction. Since PLeaS is a two-stage algorithm, its running time is greater than some existing works [12, 14, 28]. However, since the second step can be computed in parallel for all layers, as discussed in Appendix A.1.

7. Conclusion

In this work, we present PLeaS, an algorithm to merge models trained on different datasets starting from different initializations. We demonstrate that PLeaS can effectively produce merged models at different points on the compute-performance trade-off curve. We also propose PLeaS-free, a variant which can merge models without needing any data from the training domains of the component models, and empirically validate that its performance is comparable to running PLeaS with data, which widens its applicability to data-scarce regimes.

Acknowledgement

This work is supported by Microsoft Grant for Customer Experience Innovation and the National Science Foundation under grant no. 2019844, 2112471, and 2229876. JH is supported by the NSF Graduate Research Fellowship Program. PWK is supported by the Singapore National Research Foundation and the National AI Group in the Singapore Ministry of Digital Development and Innovation under the AI Visiting Professorship Programme (award number AIVP-2024-001).

References

[1] Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2022. 2, 3, 4, 6, 11, 12, 14

[2] Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes, 2024. arXiv preprint: 2403.13187. 3

[3] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90% ChatGPT quality, 2023. 1

[4] Dong Kyu Cho, Jinseok Yang, Jun Seo, Seohui Bae, Dongwan Kang, Soyeon Park, Hyeokjun Choe, and Woohyung Lim. ShERPA: Leveraging neuron alignment for knowledge-preserving fine-tuning. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. 3

[5] Muhammed Demircan. The dma and the gdpr: Making sense of data accumulation, cross-use and data sharing provisions. In *IFIP International Summer School on Privacy and Identity Management*, pages 148–164. Springer, 2022. 1

[6] Manel Baradad et al. Procedural image programs for representation learning. In *NeurIPS*, 2022. 12

[7] M.A. Ganaie, Minghui Hu, A.K. Malik, M. Tanveer, and P.N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022. 1

[8] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. 12

[9] Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Scaling expert language models with unsupervised domain discovery. *arXiv preprint arXiv:2303.14177*, 2023. 3

[10] Xiaoxi He, Zimu Zhou, and Lothar Thiele. Multi-task zipping via layer-wise neuron sharing. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018. 1, 3, 12

[11] 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. 11

[12] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 3, 6, 8

[13] Yash Jain, Harkirat Behl, Zsolt Kira, and Vibhav Vineet. Damex: Dataset-aware mixture-of-experts for visual understanding of mixture-of-datasets. In *Advances in Neural Information Processing Systems*, 2023. 1

[14] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *The Eleventh International Conference on Learning Representations*, 2023. 3, 5, 6, 8, 11, 12

[15] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. Repair: Renormalizing permuted activations for interpolation repair, 2023. 6

[16] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, 2011. 6

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 11

[18] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022. 3

[19] Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging, 2022. 3

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

[21] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 6

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 11

[23] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1406–1415, 2019. 5

[24] Yuma Rao, Jacob Steeves, Ala Shaabana, Daniel Attevelt, and Matthew McAteer. Bittensor: A peer-to-peer intelligence market. *arXiv preprint arXiv:2003.03917*, 2020. 1

[25] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. 3

[27] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020. 3

[28] George Stoica, Daniel Bolya, Jakob Brandt Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 2, 3, 4, 6, 8, 11, 12, 14

[29] Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, et al. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *arXiv preprint arXiv:2403.07816*, 2024. 3

[30] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 595–604, 2015. 6

[31] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 6

[32] Zhengqi Xu, Ke Yuan, Huiqiong Wang, Yong Wang, Mingli Song, and Jie Song. Training-free pretrained model merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5915–5925, 2024. 3, 6, 8, 11

[33] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, 2023. 1, 3

[34] Masanori Yamada, Tomoya Yamashita, Shin'ya Yamaguchi, and Daiki Chijiwa. Revisiting permutation symmetry for merging models between different datasets. *arXiv preprint arXiv:2306.05641*, 2023. 1, 2, 3, 5

[35] Enneng Yang, Li Shen, Zhenyi Wang, Guibing Guo, Xiaojun Chen, Xingwei Wang, and Dacheng Tao. Representation surgery for multi-task model merging. *arXiv preprint arXiv:2402.02705*, 2024. 1

[36] Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 3

[37] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*, 2024. 3