# $S_2$: Layer Shuffling and Superposition for Better Multi-Model Compression

A Preprint

**Hangyu Zhou** *, **Aaron Gokaslan, Volodymyr Kuleshov, Bharath Hariharan**
Computer Science, Cornell University

October 28, 2024

## Abstract

We present two complementary random mechanisms to significantly reduce interference when eliminating cross-model redundancy for efficient multi-model serving: *Layer Shuffling* and *Task Vector Superposition*. They work together to increase the orthogonality among interfering task vectors, forcing them into self-destruction without requiring any post-training learning or optimization. *Layer Shuffling* randomly reorders layers of each individual models to reduce the alignment between interfering task vectors. While *Task Vector Superposition* leverages random orthogonal transformations to decorrelate task vectors further. Together, these techniques drastically minimize interference, yielding improved performance across multiple tasks with effectively zero incremental memory cost when incorporating new models. Their data and model-independent nature also allows for seamless on-the-fly addition or removal of models, without requiring any re-computation, making them highly practical for real-world deployment scenarios.

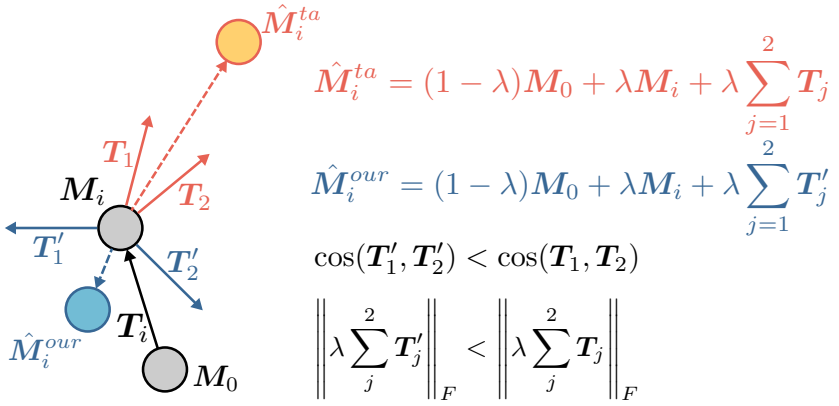



**Figure 1:** Illustration of interference reduction in multi-model compression. $M_0$ is the pre-trained checkpoint, and $M_i$ the $i$-th fine-tuned checkpoint, with task vectors $T_i$, $T_1$, and $T_2$. Standard task arithmetic ($\hat{M}_i^{\text{ta}}$, red) sums aligned task vectors, causing interference. With $S_2$ ($\hat{M}_i^{\text{our}}$, blue), layer shuffling and superposition decorrelate the interfering task vectors into $T'_1$ and $T'_2$, lowering the Frobenius norm of interference. This allows better retrieval of $M_i$ with a higher merging coefficient $\lambda$.

## 1 Introduction

Contemporary advances in machine learning are fueled by ever larger models. Language models and multimodal language models now run into billions of parameters (Cohen & Gokaslan, 2020; Radford et al., 2019, 2021; Workshop et al., 2022). These models are often finetuned into task-specific models to capture the intricacies of individual tasks. As the number of these large models proliferates, serving them becomes a challenge. It is no longer possible to

*Correspondence: hz477@cornell.edu

even store multiple models, even in high-end GPUs, significantly impacting downstream applications. Approaches to compress these models without losing accuracy are thus becoming increasingly important. When there are multiple models finetuned from the same pre-trained checkpoint on potentially related tasks, one would expect that the models have a lot of redundant information and can be compressed together. In fact, a line of work has shown that all these models can be *merged* together into a single model that can tackle all the tasks involved (Ainsworth et al., 2022; Frankle et al., 2020; Wortsman et al., 2020, 2022; Li et al., 2024; Yadav et al., 2024; Tang et al., 2024c; Ilharco et al., 2022; Yang et al., 2023). Of these, a popular framework is task arithmetic (Ilharco et al., 2022), which computes the difference between finetuned model weights and pre-trained model weights to produce task vectors for each task, and add these task vectors together with the pretrained model weights to yield a merged model.

However, the accuracy of these merged models still lags behind the accuracy of the original fine-tuned models. Prior work has identified as a potential reason the interference between the different tasks, which may not be perfectly correlated with each other (Yadav et al., 2024; Wang et al., 2024; Ortiz-Jimenez et al.). While many techniques have been proposed to limit interference, it has generally been difficult to reduce this interference.

In this paper, we take a renewed look at this interference, and find that the cause of this interference is not that the models involved are too different, but *that they are too similar*. Concretely, we find that task arithmetic works best when the task vectors are as orthogonal to each other as possible. Armed with this insight, we propose two new ways of improving upon task arithmetic. Our first approach is to *shuffle* task vectors across the layers of each model before combining them, with an inverse shuffle applied at test time. Our second approach is to apply a random sign flip or rotation to the task vectors before merging them, again inverting the transformation at test time. Both approaches significantly reduce interference between the task vectors. They also have the advantage of being simple, efficient and requiring no training or optimization.

We test our approach on three different benchmarks involving large models and their finetuned versions: CLIP-ViT-B/32 and CLIP-ViT-L/14 for zero-shot image classification (Radford et al., 2021), Flan-T5-base for text generation (Longpre et al., 2023), and GPT-2 for text classification (Radford et al., 2019). We find that across all of these benchmarks, our approach substantially improves in terms of accuracy over prior model merging-based approaches. When compared to the original fine-tuned models, in two of the three benchmarks our approach yields near-identical accuracy to the individual models while reducing the storage costs by $4\times$. In sum, our contributions are:

1. We provide an analysis of the interference between tasks in task arithmetic, which suggests that similarity between the task vectors may be a problem.

2. We propose two complementary strategies for reducing interference. Our first strategy randomly shuffles parameter matrices across layers. Our second strategy applies a random rotation or a sign flip to the task vectors before merging.

3. We demonstrate through experiments on three benchmarks that our approach compresses multiple models together and achieves much higher accuracy than prior model merging based approaches.

## 2 Problem setup

We are given $T$ models $\{\boldsymbol{\Theta}_i\}_{i=1}^T$ fine-tuned from a single pre-trained model $\boldsymbol{\Theta}_0$ on tasks $i = 1, \ldots, T$. Each model $\boldsymbol{\Theta}_i$ as a set of parameter matrices:

$$\boldsymbol{\Theta}_i = \left\{ \mathbb{I}_i, \left( \boldsymbol{M}_i^{k,1}, \boldsymbol{M}_i^{k,2}, \ldots, \boldsymbol{M}_i^{k,m_k} \right)_{k=1}^{K}, \mathbb{O}_i \right\},$$

Here $\mathbb{I}_i$ and $\mathbb{O}_i$ are the input and output layers. Each model has $K$ blocks, with the $k$-th block containing $m_k$ matrices $\boldsymbol{M}_i^{k,1}, \boldsymbol{M}_i^{k,2}, \ldots, \boldsymbol{M}_i^{k,m_k}$.

Our goal is to *compress* $\{\boldsymbol{\Theta}_i\}_{i=1}^T$ into a compact representation $\boldsymbol{\Theta}_* = \texttt{compress}(\{\boldsymbol{\Theta}_i\}_{i=1}^T)$ with minimal memory usage, so that at test time, given a task $i$, we can retrieve an *approximate* model $\hat{\boldsymbol{\Theta}}_i = \texttt{retrieve}(\boldsymbol{\Theta}_*, i)$ for the task that achieves high accuracy on this task.

One way to address this problem is obviously to compress each individual model using strategies such as pruning or quantization. However, here we are interested in techniques that can leverage the structure of the problem (namely, $T$ models finetuned from the same source) to yield storage that is *sub-linear* in $T$.

## 3 Task Arithmetic

A promising line of approaches for this problem derives from the observation that models fine-tuned for different tasks could be *merged* into a single model that gives reasonable accuracy for all tasks (Ilharco et al., 2022). Concretely, their framework, called task arithmetic, first computes the difference between the finetuned model weights for each layer $k$, $\boldsymbol{M}_i^k$, and the corresponding pre-trained model weights, $\boldsymbol{M}_0^k$, to produce *task vectors* $\boldsymbol{T}_i^k = \boldsymbol{M}_i^k - \boldsymbol{M}_0^k$. Task arithmetic then computes a weighted average of the pretrained model weights and the task vectors:

$$\boldsymbol{M}_\star^k \leftarrow \boldsymbol{M}_0^k + \lambda \sum_{i=1}^{T} \boldsymbol{T}_i^k = \boldsymbol{M}_0^k + \lambda \sum_{i=1}^{T} (\boldsymbol{M}_i^k - \boldsymbol{M}_0^k) \tag{1}$$

$$\boldsymbol{\Theta}_\star^{TA} \leftarrow \{\boldsymbol{M}_\star^k\}_{k=1}^K \tag{2}$$

where $\lambda \in \mathbb{R}^+$ is the merging coefficient. At test time, this compressed model is directly applied no matter what the task:

$$\hat{\boldsymbol{\Theta}}_i \leftarrow \boldsymbol{\Theta}_\star^{TA} = \{\boldsymbol{M}_\star^k\}_{k=1}^K \tag{3}$$

This approach can be used to reduce model storage by a factor of $T$ since we only need to store one model instead of $T$ different models. However, as we show later, this yields much lower accuracy than the original fine-tuned models.

One reason that has been put forward for the low accuracies offered by task arithmetic is *task interference*: different tasks may want to set particular parameters differently, and merging these parameters naively will cause one task to harm another task's accuracy(Yadav et al., 2024; Wang et al., 2024; Tang et al., 2023). However, a mathematical analysis of this interference is missing. Below, we delve deeper into this interference, and find a counter-intuitive solution.

### 3.1 Interference in Task Arithmetic

To understand the interference term, let us consider what happens when we apply the merged model to task $i$.

$$\hat{\boldsymbol{M}}_i^k = \boldsymbol{M}_\star^k \;\; equation\ 3 \tag{4}$$

$$= \boldsymbol{M}_0^k + \lambda \sum_{i=1}^{T} (\boldsymbol{M}_i^k - \boldsymbol{M}_0^k) \tag{5}$$

$$= (1-\lambda)\boldsymbol{M}_0^k + \lambda \boldsymbol{M}_i^k + \lambda \sum_{j \neq i} \boldsymbol{T}_j^k. \tag{6}$$

The last line suggests that the model being applied to the $i$-th task is interpolating between the pretrained model $\boldsymbol{M}_0^k$ and the task-specific finetuned model $\boldsymbol{M}_i^k$, but with an additional interference coming from other merged models : $\lambda \sum_{j \neq i} \boldsymbol{T}_j^k$. To retrieve the finetuned model, the first two terms suggest that we should set $\lambda$ to 1. However this will *increase* the interference term, $\lambda \sum_{j \neq i} \boldsymbol{T}_j^k$. Some prior work has tried to achieve a good balance by optimizing $\lambda$ for each model and layer using test-time adaptation(Yang et al., 2023), but this balance has generally been tricky to achieve.

Instead of focusing on $\lambda$, let us look at this interference term in greater detail by analyzing its Frobenius norm:

$$\left\| \lambda \sum_{j \neq i} \boldsymbol{T}_j^k \right\|_F^2 = \lambda^2 \left( \sum_{j \neq i} \|\boldsymbol{T}_i^k\|_F^2 + 2 \sum_{\substack{1 \leq l < j \leq n \\ l, j \neq i}} \|\boldsymbol{T}_l^k\|_F \|\boldsymbol{T}_j^k\|_F \cos(\boldsymbol{T}_l^k, \boldsymbol{T}_j^k) \right). \tag{7}$$

We observe that the interference term is directly correlated with two quantities: the magnitude of the task vectors $\boldsymbol{T}_i^k$ (which is out of our control since it depends on the task-specific finetuning), and the cosine products between them. Interestingly, the interference term is maximum when the task vectors are very closely aligned with each other. Thus, the problem with task arithmetic is not that the individual task vectors are very different from each other, but that they are *too similar*.

Our goal, therefore, should be to make the task vectors as different from each other as possible. Below, we propose two strategies for doing this.

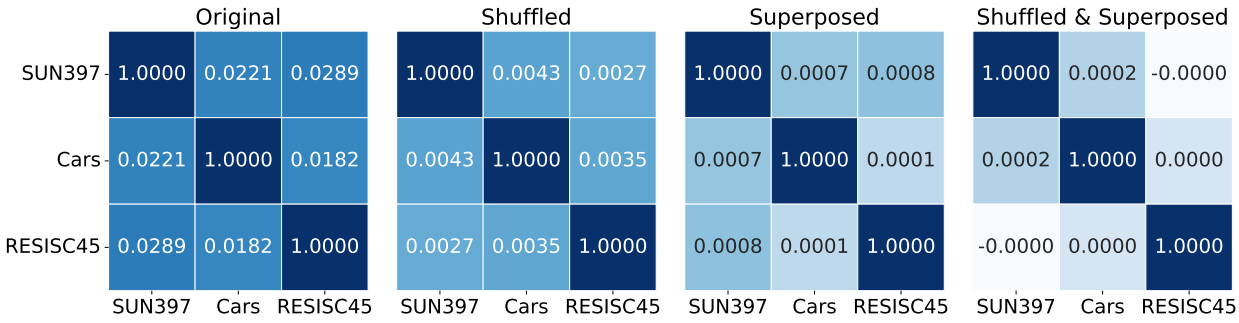



**Figure 2:** Average pairwise cosine similarity of three out of eight CLIP-ViT-B/32 task vectors during model retrieval for SUN397 across three repetitions. Both random layer shuffling and superposition increase mutual orthogonality, with an additive effect when combined.

## 4 Methodology

As described above, to minimize interference, we want the task vectors to be as orthogonal to each other as possible. We propose two complementary random algorithms to achieve this: *layer shuffling* and *task vector superposition*.

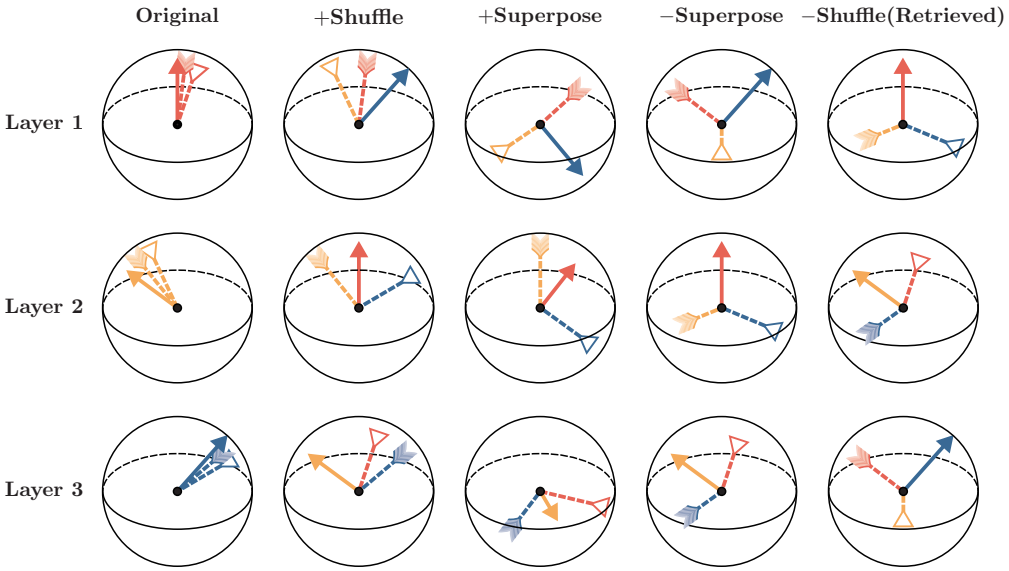



**Figure 3:** Overview of the $S_2$ approach in a three-layer model with three fine-tuned checkpoints. Different task vectors (distinct arrowheads) initially align across layers, causing interference when directly merged. Shuffling layers (column 2) and applying superposition with random binary diagonal matrices (column 3), followed by inverse transformations (columns 4 and 5), retain the target task's orientation (standard arrowhead) while reducing interference through increased orthogonality among other vectors.

### 4.1 Random Layer Shuffling

Across several model architectures (CLIP-ViT-B/32, CLIP-ViT-L/14 (Radford et al., 2021), Flan-T5 (Longpre et al., 2023), and GPT-2 (Radford et al., 2019)), we observed that task vector layers within the same model exhibit greater variability compared to corresponding layers across fine-tuned models. This insight suggests that by randomly shuffling layers across different task vectors, we can reduce the pairwise cosine similarity of interfering task vectors and thus minimize their contribution to the interference. To that end, we propose *layer shuffling* as a simple fix to the problem.

**Method Description.** The models we consider are made of multiple parameter blocks of similar structure. For example, transformers have multiple MLP layers and multiple attention layers. Many of the MLP and Attention layers have weight matrices of the same size.

Our proposal is that when merging the task vectors, for each model, we first *randomly permute* the task vectors across layers of the same type and with the same dimensions of parameter matrices. Then, when we want to perform inference on a particular task, we perform the inverse of the corresponding permutation to ontain the model for the task.

Concretely, for each task $i$, we produce a random permutation of the layers $\sigma_i$, taking care to only permute across layers of the same type and same dimensionality. We then produce merged task vectors by adding up these shuffled task vectors across models. The merged task vector for the $k$-th layer is:

$$\boldsymbol{T}_\star^k \leftarrow \sum_{i=1}^{T} \boldsymbol{T}_i^{\sigma_i(k)} \tag{8}$$

The number of such merged task vectors is equal to the total number of layers $K$. We then store both the pretrained model $\boldsymbol{\Theta}_0$ and the merged task vectors $\{\boldsymbol{T}_\star^k\}_{k=1}^K$:

$$\boldsymbol{\Theta}_*^{Shuffle} \leftarrow \left(\boldsymbol{\Theta}_0, \{\boldsymbol{T}_\star^k\}_{k=1}^K\right) \tag{9}$$

**Reduction in Interference:** Because parameter vectors from different layers are less likely to align, we effectively reduce the cosine product between the task vectors being merged: instead of the term $\cos(\boldsymbol{T}_i^k, \boldsymbol{T}_j^k)$ in the interference term (equation 7), we now have the product $\cos(\boldsymbol{T}_i^{\sigma_i(k)}, \boldsymbol{T}_j^{\sigma_j(k)})$ which we expect to be much lower. We thus expect smaller interference and thus more faithful retrieval of model weights for each task. The first two parts of Figure 3 shows this effect in action, with layer shuffling reducing the pairwise cosine similarity among interfering vectors unanimously.

### 4.2 Task Vector Superposition

We can also leverage the *blessing of dimensionality* (Gorban & Tyukin, 2018) to promote orthogonality among high dimensional vectors. We take inspiration from Cheung et al. (2019) on continual learning and introduce *superposition* as a complementary approach to increase the mutual orthogonality among interfering task vectors.

**Method Description.** Considering merging the parameters of layer $k$, we sample a random binary diagonal matrices whose diagonal entries have equal probability to be $+1$ or $-1$ to each of the $T$ task vectors and apply them to the vectors before summation:

$$\boldsymbol{T}_\star^k \leftarrow \sum_{i=1}^{T} \boldsymbol{T}_i^k \boldsymbol{C}_i^k. \tag{10}$$

We call them context matrices and $\forall i \in [1, \ldots, T]$, $\boldsymbol{C}_i^k \boldsymbol{C}_i^{k(T)} = \boldsymbol{C}_i^k \boldsymbol{C}_i^{k(-1)} = \boldsymbol{I}$.

When performing task $i$, we apply the inverse transformation $\boldsymbol{C}_i^{k(-1)}$ to retrieve task vector $\boldsymbol{T}_i^k$ from the superposition:

$$\hat{\boldsymbol{T}}_i^k = \boldsymbol{T}_\star^k \boldsymbol{C}_i^{k(-1)} \tag{11}$$

$$= \sum_{i=1}^{T} [\boldsymbol{T}_i^k \boldsymbol{C}_i^k] \boldsymbol{C}_i^{k(-1)} \tag{12}$$

$$= \boldsymbol{T}_i^k + \sum_{j \neq i} [\boldsymbol{T}_j^k \boldsymbol{C}_j^k \boldsymbol{C}_i^{k(-1)}] \tag{13}$$

We store both the pretrained model $\boldsymbol{\Theta}_0$, the merged task vectors $\{\boldsymbol{T}_\star^k\}_{k=1}^K$, as well as the context matrices $\{\boldsymbol{C}_\star^k\}_{k=1}^K$:

$$\boldsymbol{\Theta}_*^{Superpose} \leftarrow \left(\boldsymbol{\Theta}_0, \{\boldsymbol{T}_\star^k\}_{k=1}^K, \{\boldsymbol{C}_\star^k\}_{k=1}^K\right) \tag{14}$$

**Reduction in Interference.** Two random vectors in high dimensional space is very likely to be nearly orthogonal with each other. Now the cosine similarity in equation 7 changes from $\cos(\boldsymbol{T}_i^k, \boldsymbol{T}_j^k)$ to $\cos(\boldsymbol{T}_i^k \boldsymbol{C}_i \boldsymbol{C}_j^{l(-1)}, \boldsymbol{T}_j^k \boldsymbol{C}_i \boldsymbol{C}_j^{l(-1)})$ when performing task $l$. The randomly sampled diagonal binary matrices $\{\boldsymbol{C}_\star^k\}_{k=1}^K$ will randomize the task vectors, leading to more orthogonal task vectors, smaller interference, and thus better retrieval of model parameters for the task at hand. Again, Figure 3 confirmed the cosine similarity reduction when task vectors are superposed together.

**Table 1:** Performance comparison on eight image classification tasks with CLIP-ViT-B/32 models merging and compression, as well as memory footprint estimate. Three repetitions are done for methods with randomness. Variances smaller than $0.1\%$ are omitted.

| Method | Avg.(%) ↑ | Bits(Gb) ↓ | SUN397 | Cars | RESISC45 | EuroSAT | SVHN | GTSRB | MNIST | DTD |
|---|---|---|---|---|---|---|---|---|---|---|
| Pre-trained | 48.2 (53.4) | 0.564 (1.00) | 63.2 | 59.8 | 60.7 | 46.0 | 31.6 | 32.5 | 48.3 | 43.9 |
| Fine-tuned | 90.3 (100) | 2.84 (5.03) | 75.0 | 78.3 | 95.2 | 99.0 | 97.3 | 98.9 | 99.6 | 79.7 |
| Weight Averaging | 66.5 (73.6) | 0.564 (1.00) | 65.4 | 62.6 | 70.8 | 76.9 | 64.5 | 54.9 | 86.3 | 50.9 |
| Fisher Merging | 70.6 (78.2) | 0.564 (1.00) | 66.7 | 64.0 | 72.2 | 91.6 | 69.0 | 64.3 | 83.5 | 53.7 |
| RegMean | 80.5 (89.1) | 0.564 (1.00) | 67.8 | 68.9 | 82.5 | 94.4 | 90.6 | 79.2 | 94.7 | 63.2 |
| Task Arithmetic | 69.8 (77.2) | 0.564 (1.00) | 64.4 | 61.5 | 70.5 | 80.4 | 73.9 | 62.8 | 93.0 | 51.6 |
| Ties-Merging | 72.2 (80.0) | 0.564 (1.00) | 67.1 | 64.2 | 74.1 | 91.6 | 77.7 | 69.4 | 94.1 | 54.0 |
| AdaMerging | 82.6 (91.5) | 0.564 (1.00) | 67.9 | 71.3 | 83.5 | 92.7 | 87.4 | 92.9 | 98.2 | 67.0 |
| PSP | 4.5 (5.0) | 0.565 (1.00) | 0.3 | 0.5 | 1.9 | 10.4 | 8.8 | 2.3 | 9.8 | 1.9 |
| WEMoE | 89.2 (98.8) | 2.27 (4.03) | 73.7 | 76.8 | 93.4 | 98.2 | 96.8 | 98.2 | 99.6 | 76.6 |
| SMILE | 89.3 (98.9) | 1.23 (2.19) | 73.6 | 77.8 | 92.0 | 98.3 | 96.9 | 98.1 | 99.6 | 78.1 |
| TA+Shuffle (Ours) | 81.3 (90.0) | 1.13 (2.00) | 65.6 | 58.5 | 86.8 | 94.5 | 93.2 | 91.4 | 98.5 | 62.2 |
| STA (Ours) | 89.6 (99.2) | 1.13 (2.00) | 74.4 | 75.6 | 94.6 | 99.0 | 97.1 | 98.5 | 99.5 | 77.8 |
| STA+Shuffle (Ours) | 89.9 (99.6) | 1.13 (2.00) | 74.8 | 76.7 | 94.8 | 99.0 | 97.2 | 98.6 | 99.5 | 78.7 |

# 5 Experiments

In this section, we validate the proposed methods on a diverse set of model merging/compression scenarios from FusionBench (Tang et al., 2024a). We show comparable performance across discriminative and generative tasks in vision and language domains. An ablation and analysis section reveals the importance of each component. We seek to demonstrate how does the merging coefficient interact with the performance. We also conduct a a comparison between different context matrix design and target layer selection. We further demonstrate the ability of our method to be used in broader scenarios like PEFT model compression.

## 5.1 Experiment Setup

**Datasets and Models.** We follow Tang et al. (2024a) and select three representative scenarios to evaluate our methods. This includes i) CLIP-ViT-B/32 fine-tuned on eight image classification datasets; ii) Flan-T5-base fine-tuned on eight text generation datasets; and iii) GPT-2 fine-tuned on seven text classification datasets. Detailed information on the datasets and models is in Appendix B.

**Baselines and Metrics.** We have baselines from model merging literatures with various level of memory usage. "Pre-trained" and "fine-tuned" are using either the pre-trained model or each fine-tuned model individually, providing the lower and upper bound of performance. Detailed information on all the baselines can be found in Appendix B. By default, we use both layer shuffling and superposition on task arithmetic for optimal performance. It's denoted as *STA+Shuffle*. We use three repetitions per experiment setup to measure the variance of our random algorithms. Follow (Ilharco et al., 2022), the best merging coefficient $\lambda$ is determined by a grid search on the validation set. We use accuracy and memory footprint to evaluate the performance.

## 5.2 Performance Analysis

**Superior MTL Performance.** We see significant performance improvement in terms of average accuracy across all three benchmarks, per Tables 1 to 3. Especially for the image classification tasks (with CLIP-ViT-B/32) and the text generation tasks (with Flan-T5-base), where *STA+Shuffle* is approaching the performance of individual fine-tuned models. Among methods that use extra parameters, we surpass both WEMoE (Tang et al., 2024c) and SMILE (Tang et al., 2024b) on image classification despite having lower memory cost. We also outperform SMILE on text generation tasks with higher memory consumption. Meanwhile, Parameter Superposition (PSP) (Cheung et al., 2019) performs well in continual learning settings by directly superposing the entire model. But they perform poorly across all the benchmarks . This shows that superposing task vectors is a key ingredient in effective model compression/merging in offline settings.

**Amortizable Memory Overhead.** Note that our method only doubles the memory footprint, despite the large number of tasks learned. The extra storage cost comes from the merged task vectors $\{\boldsymbol{T}_\star^k\}_{k=1}^K$, as well as the context

**Table 2:** Performance comparison on eight GLUE text generation tasks with Flan-T5-base models merging and compression, as well as memory footprint estimate. Variances over three repetitions that are smaller than $0.1\%$ are omitted.

| Method | Avg.(%) ↑ | Bits(Gb) ↓ | CoLA | MNLI | MRPC | QNLI | QQP | RTE | SST2 | STSB |
|---|---|---|---|---|---|---|---|---|---|---|
| Pre-trained | 75.7 (87.6) | 1.19 (1.00) | 69.1 | 56.5 | 76.2 | 88.4 | 82.1 | 80.1 | 91.2 | 62.2 |
| Fine-tuned | 86.4 (100) | 9.52 (8.00) | 75.0 | 83.4 | 87.5 | 91.5 | 85.4 | 85.9 | 93.6 | 88.7 |
| Weight Averaging | 78.9 (91.3) | 1.19 (100) | 69.1 | 62.6 | 79.4 | 89.8 | 83.9 | 81.2 | 91.7 | 73.2 |
| Task Arithmetic | 79.6 (92.1) | 1.19 (1.00) | 69.7 | 64.1 | 79.2 | 90.2 | 83.9 | 81.6 | 92.1 | 76.4 |
| Ties-Merging | 79.9 (92.5) | 1.19 (1.00) | 70.3 | 65.0 | 78.9 | 90.2 | 83.5 | 81.6 | 91.7 | 78.3 |
| PSP | 0.0 (0.0) | 1.19 (1.00) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | N/A |
| SMILE | 85.5 (99.0) | 1.81 (1.52) | 73.2 | 84.2 | 85.0 | 91.3 | 84.9 | 84.8 | 93.5 | 87.3 |
| TA+Shuffle (Ours) | 85.7 (99.0) | 2.38 (2.00) | 75.5 | 82.0 | 87.5 | 91.1 | 83.9 | 83.8 | 93.6 | 88.4 |
| STA (Ours) | 86.5 (100) | 2.38 (2.00) | 77.2 | 82.1 | 87.6 | 91.6 | 85.3 | 85.7 | 93.2 | 89.0 |
| STA+Shuffle (Ours) | 86.4 (100) | 2.38 (2.00) | 75.6 | 82.8 | 88.2 | 91.7 | 85.3 | 85.7 | 93.5 | 88.9 |

**Table 3:** Performance comparison on seven GLUE text classification tasks with GPT-2 models merging and compression, as well as memory footprint estimate. Variances over three repetitions that are smaller than $0.1\%$ are omitted.

| Method | Avg.(%) ↑ | Bits(Gb) ↓ | CoLA | MNLI | MRPC | QNLI | QQP | RTE | SST-2 |
|---|---|---|---|---|---|---|---|---|---|
| Pre-trained | 44.5 (54.3) | 0.498 (1.00) | 30.9 | 33.0 | 31.4 | 49.2 | 63.2 | 52.7 | 50.9 |
| Fine-tuned | 82.0 (100) | 3.49 (7.00) | 76.8 | 82.1 | 80.4 | 88.3 | 89.6 | 65.3 | 91.2 |
| Weight Averaging | 56.1 (63.3) | 0.498 (1.00) | 55.0 | 55.1 | 51.0 | 57.6 | 76.7 | 44.8 | 52.5 |
| Fisher Merging | 58.7 (64.7) | 0.498 (1.00) | 54.8 | 58.0 | 39.5 | 63.3 | 81.5 | 49.1 | 64.7 |
| RegMean | 68.8 (79.7) | 0.498 (1.00) | 61.7 | 70.4 | 65.4 | 69.7 | 78.8 | 56.0 | 79.7 |
| Task Arithmetic | 70.0 (85.4) | 0.498 (1.00) | 68.7 | 68.6 | 69.6 | 70.5 | 81.8 | 47.3 | 83.6 |
| Ties-Merging | 70.0 (82.4) | 0.498 (1.00) | 68.4 | 71.4 | 68.4 | 69.6 | 82.4 | 47.7 | 81.8 |
| PSP | 44.5 (54.3) | 0.498 (1.00) | 30.9 | 33.6 | 31.6 | 49.5 | 63.2 | 52.5 | 50.3 |
| TA+Shuffle (Ours) | 76.7 (93.5) | 0.997 (2.00) | 71.6 | 80.3 | 73.9 | 85.8 | 88.5 | 47.5 | 89.3 |
| STA (Ours) | 71.3 $\pm$0.6 (87.0) | 0.997 (2.00) | 62.3 $\pm$0.3 | 78.2 | 46.1 $\pm$4 | 82.6 | 88.4 | 52.7 | 88.9 |
| STA+Shuffle (Ours) | 76.6 $\pm$0.2 (93.4) | 0.997 (2.00) | 70.3 $\pm$0.1 | 81.0 | 61.0 $\pm$1.3 | 87.2 | 89.3 | 57.5 $\pm$0.3 | 90.2 |

matrices $\{\boldsymbol{C}_\star^k\}_{k=1}^K$, as mentioned in equation 9 and 14. The merged task vectors contributes the majority of additional storage cost ($> 99\%$), as the context matrices are binary diagonal matrices, which can be stored very efficiently. Moreover, we can choose to store only the random seeds, and regenerate the corresponding context matrices on-the-fly. This will leads to effectively *zero* memory overhead for every additional model in compression. This process will amortize the memory footprint when more models are compressed together.

### 5.3 Key Components Ablation

In this section, we ablate both the *random layer shuffling* and *superposition* to show their individual contribution. Specifically, we derive two variants:

- **TA+Shuffle**: we randomly shuffle the layers before task arithmetic without performing superposition.
- **STA**: we superpose task vectors without layer shuffling.

Tables 1 to 3 shows that both methods can significantly improve upon task arithmetic in every benchmark. In some benchmarks, shuffling works better (Flan-T5-base and GPT-2) while superposition works better in others (CLIP). This difference may be because of the nature of task vectors themselves: how they vary across the model layers and across different models. We find that the combined approach is able to combine gains from both components, yielding consistently the best result across all the benchmarks. This complementary effect is further manifested in Figure 3, where introducing both mechanism gets the smallest pairwise cosine similarity among interfering task vectors.

### 5.4 Impact of Merging Coefficient $\lambda$

Here we examine the interplay between the merging coefficient $\lambda$ and the average performance across different setup. For each variant derived in section 4, we perform a grid search on $\lambda = \{0.1, 0.2, \cdots, 1.0\}$ when compressing eight CLIP-ViT-B/32 models for image classification. Figure 4 shows the change of optimal model performance and the coefficient $\lambda$ when *layer shuffling* and *superposition* are introduced to task arithmetic.

We observe that when shuffling and superposition are introduced, the best performance increases along with the value of $\lambda$. This shows the effectiveness of our method in reducing interference, allowing larger $\lambda$ to be selected for more authentic model retrieval as according to equation 4.

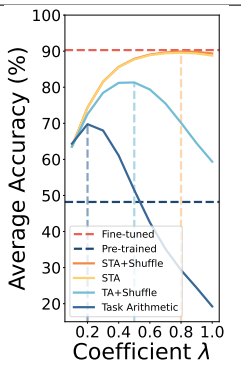
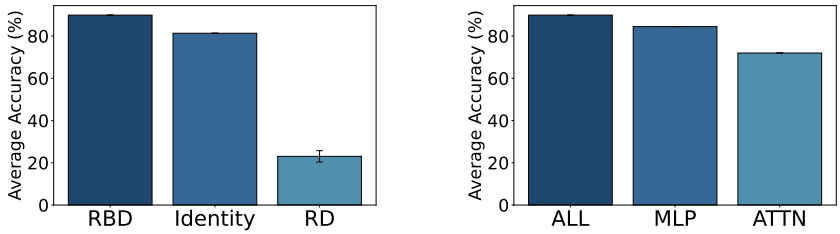



**Figure 4:** The impact of $\lambda$ on average accuracy over eight image classification tasks.

### 5.5 Impact of Context Matrix Design

To further examine how *task vector superposition* works and shad light on better context matrix design, we make a comparison between three types of context matrices: random binary diagonal matrix with $\{-1, +1\}$ entries (RBD), identity matrix (Identity), and random diagonal matrix with entries draw from Normal distribution (RD). We use random layer shuffling when compressing the 8 CLIP-ViT-B/32 models on the image classification tasks.

The average accuracy and its variance with the optimal merging coefficient is shown in Figure 5. RBD receives higher accuracy than Identity due to the randomness it introduces, which reduces interference as discussed in section 4.2. Despite being random, RD's accuracy is much lower than RBD. We think this happens because RD is not an orthogonal matrix. It fails to preserve the Frobenius norm of $\boldsymbol{T}_j^k \boldsymbol{C}_j^k \boldsymbol{C}_i^{k(-1)}$ and thus disturb this self-cancellation process.




(a) Context Matrix Design (b) Target Layer Selection

**Figure 5:** (a) Impact of context matrix design to the average accuracy. RBD stands for random binary diagonal; RD stands for random diagonal. (b) Impact of target layer selection to the average accuracy. ALL stands for choosing all layers; MLP means only MLP layers are selected; and ATTN stands for attention layers.

### 5.6 Target Layer Selection

By default we apply the random operations on all layers within the models. In this section, we evaluate the benefits of targeting specific types of layers. To do this, we create two variants: MLP (which selects only the MLP layers) and ATTN (which selects only the attention layers), in addition to the default setup (ALL). Figure 5 shows the average accuracy for each setup across eight image classification tasks using CLIP-ViT-B/32. The ALL configuration achieves the highest accuracy, followed by MLP and ATTN. Note that the total number of parameters in MLP is twice that of ATTN, explaining the gradual decline in performance as fewer parameters are selected.

### 5.7 Model Hot Swapping

The ability to hot-swap models in real-world applications is crucial, especially in dynamic environments like model serving, where new models need to be integrated into the system regularly, and deprecated ones need to be removed in a timely fashion. As mentioned, the *STA+Shuffle* method allows for this by shuffling layers and sampling diagonal binary matrices *independently* of data or model parameters, thus enabling the on-the-fly addition of new models without the need for recomputation. This provides our a method a big advantage over methods like WEMoE which require recomputation of the router when new models are added (Tang et al., 2024c). We dub this feature hot swapping or hot adding to borrow a termfrom the hardware literature.

**Table 4:** Comparison of selected methods with hot adding and recomputation requirements when new models are added to the pool.

| Method | Hot Swap | Recomputation |
|---|---|---|
| Task Arithmetic | ✓ | ✗ |
| WEMoE | ✗ | ✓ |
| STA+Shuffle | ✓ | ✗ |

**Table 5:** Multi-task performance when merging Flan-T5-base LoRA models on eight GLUE tasks (paraphrase). Variances over three repetitions that are smaller than $0.1\%$ are omitted. (STSB is Spearman Rho, others are accuracy?)

| Method | Avg.(%) ↑ | Bits(Gb) ↓ | CoLA | MNLI | MRPC | QNLI | QQP | RTE | SST2 | STSB |
|---|---|---|---|---|---|---|---|---|---|---|
| Pre-trained | 75.7 (87.6) | 1.19 (100) | 69.1 | 56.5 | 76.2 | 88.4 | 82.1 | 80.1 | 91.2 | 62.2 |
| Fine-tuned | 84.6 (100) | 1.25 (1.05) | 69.1 | 82.7 | 85.5 | 90.9 | 84.0 | 84.4 | 92.9 | 87.4 |
| Weight Averaging | 78.2 (92.4) | 1.19 (100) | 69.7 | 59.7 | 78.9 | 90.1 | 83.8 | 80.5 | 91.2 | 72.0 |
| Task Arithmetic | 77.4 (91.5) | 1.19 (100) | 68.8 | 55.2 | 78.7 | 89.8 | 83.7 | 79.1 | 91.5 | 72.4 |
| Ties-Merging | 77.5 (91.6) | 1.19 (100) | 68.3 | 56.3 | 79.4 | 89.8 | 83.7 | 79.4 | 91.6 | 71.2 |
| SMILE | 84.0 (99.3) | 1.21 (1.02) | 69.3 | 82.9 | 83.8 | 90.6 | 83.9 | 83.4 | 93.1 | 85.1 |
| STA (Ours) | 82.0 (96.9) | 1.20 (1.01) | 69.1 | 80.3 | 80.5 | 90.5 | 83.6 | 79.4 | 92.4 | 80.2 |
| STA+Shuffle (Ours) | 82.9 (98.0) | 1.20 (1.01) | 69.1 | 81.0 | 81.9 | 90.2 | 84.1 | 83.5 | 92.5 | 80.6 |

### 5.8 Parameter Efficient Finetuning (PEFT) Model Compression

We also apply our method on PEFT adapter weights. Consider a LoRA (Hu et al., 2021), where we have a fixed pre-trained model $\boldsymbol{\Theta}_0$, along with LoRA weights $\boldsymbol{L}_i$. We merge the LoRA weights to get the fine-tuned model: $\boldsymbol{\Theta}_i = \boldsymbol{\Theta}_0 + \boldsymbol{L}_i$. Similar to section 4.1 and 4.2, we apply *random layer shuffling* and *superposition* on these LoRA weight vectors before retrieval.

Experiments on Flan-T5-base LoRA fine-tunes (Longpre et al., 2023; Tang et al., 2024a,b) demonstrate that our method is performative in PEFT compression settings (Table 5). With 98% normalized average accuracy compare to the fine-tuned baseline, and 1.20 Gb memory usage, our method presents a new trade-off point between performance and storage usage.

## 6 Related Work

**Model Merging.** Model merging strives to reduce the cross-model redundancy to obtain a multi-task model out of a set of specialists. One prominent direction is linear mode connectivity, as discussed by Frankle et al. (2020), which investigates connectivity between different solutions in weight space. Additionally, permutation invariance plays a key role in many approaches, as noted in Ainsworth et al. (2022). Among classic methods, model soup (Wortsman et al., 2022; Li et al., 2024) and task arithmetic (Ilharco et al., 2022; Yang et al., 2023; Yadav et al., 2024; Tang et al., 2024c) offer effective strategies for model merging and interference reduction. As a natural extension, some recent works use various methods to separate preserved information from task-specific ones and receives good results. For example, Li et al. (2024) extract the common information using averaging, similar to task arithmetic. Similarly, Ostapenko et al. (2024) leverage singular value decomposition (SVD) to decompose LoRA weights, aligning with methods that aim to minimize interference through weight adjustments. We would like to also acknowledge the concurrent work, SMILE, which also explicitly addresses these challenges in a similar fashion, as seen in the latest research by Lu et al. (2024).

**Model Compression.** Model compression can take a variety of forms including quantization (Wortsman et al., 2022), pruning (Zhu & Gupta, 2017), and distillation (Sun et al., 2019). But it has largely been focus on compressing single models. Few works exist that try to reduce redundancies across models(Duan et al., 2022).

**Vector Symbolic Architecture (VSA)** Schlegel et al. (2022) refers to a computational framework that efficiently encodes and manipulates multiple pieces of information within shared high-dimensional vector spaces. These architec-

tures leverage algebraic operations such as binding, superposition, and permutation to represent and process structured symbolic information while preserving distributed representations.

## 7 Discussion and Future Work

In this work, we introduced two key strategies—random layer shuffling and task vector superposition—that significantly reduce task interference during multi-model compression. Our approach demonstrates that improving orthogonality between task vectors is critical to minimizing interference, which leads to enhanced performance when merging models. By introducing randomness in both the layer alignment and task vector manipulation, our method provides a simple yet effective solution to interference, requiring no additional training or optimization. This simplicity, combined with its ability to scale across different architectures and tasks, makes our approach highly efficient for real-world multi-model serving environments.

A key strength of this method lies in its scalability and adaptability. Unlike prior approaches that require computationally intensive recomputation for every model addition, our method allows seamless hot-swapping of models without the need to reprocess existing task vectors. This flexibility makes it particularly well-suited for dynamic environments, such as real-time model serving.

Future research will further refine these strategies, particularly exploring the role of specific layer subsets in model merging and optimizing the randomization processes for improved results. We also plan to evaluate our method on larger models or across models with different architectures to enable broader applications.

## References


Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

Brian Cheung, Alexander Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. *Advances in neural information processing systems*, 32, 2019.

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Vanya Cohen and Aaron Gokaslan. Opengpt-2: open language models and implications of generated text. *XRDS*, 27(1):26–30, September 2020. ISSN 1528-4972. doi: 10.1145/3416063. URL `https://doi.org/10.1145/3416063`.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Wenhong Duan, Zhenhua Liu, Chuanmin Jia, Shanshe Wang, Siwei Ma, and Wen Gao. Differential weight quantization for multi-model compression. *IEEE Transactions on Multimedia*, 25:6397–6410, 2022.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020.

Alexander N Gorban and Ivan Yu Tyukin. Blessing of dimensionality: mathematical foundations of the statistical physics of data. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2118):20170237, 2018.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

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.

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.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 554–561, 2013.

Tao Li, Weisen Jiang, Fanghui Liu, Xiaolin Huang, and James T Kwok. Scalable learned model soup on a single gpu: An efficient subspace training strategy. *arXiv preprint arXiv:2407.03641*, 2024.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pp. 22631–22648. PMLR, 2023.

Zhenyi Lu, Chenghao Fan, Wei Wei, Xiaoye Qu, Dangyang Chen, and Yu Cheng. Twin-merging: Dynamic integration of modular expertise in model merging. *arXiv preprint arXiv:2406.15479*, 2024.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models, 2023. *URL http://arxiv. org/abs/2305.12827*.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. Towards modular llms by building and reusing a library of loras. *arXiv preprint arXiv:2405.11157*, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Kenny Schlegel, Peer Neubert, and Peter Protzel. A comparison of vector symbolic architectures. *Artificial Intelligence Review*, 55(6):4523–4555, 2022.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Anke Tang, Li Shen, Yong Luo, Liang Ding, Han Hu, Bo Du, and Dacheng Tao. Concrete subspace learning based interference elimination for multi-task model fusion. *arXiv preprint arXiv:2312.06173*, 2023.

Anke Tang, Li Shen, Yong Luo, Han Hu, Bo Do, and Dacheng Tao. Fusionbench: A comprehensive benchmark of deep model fusion. *arXiv preprint arXiv:2406.03280*, 2024a.

Anke Tang, Li Shen, Yong Luo, Shuai Xie, Han Hu, Lefei Zhang, Bo Du, and Dacheng Tao. Smile: Zero-shot sparse mixture of low-rank experts construction from pre-trained foundation models. *arXiv preprint arXiv:2408.10174*, 2024b.

Anke Tang, Li Shen, Yong Luo, Nan Yin, Lefei Zhang, and Dacheng Tao. Merging multi-task models via weight-ensembling mixture of experts. *arXiv preprint arXiv:2402.00433*, 2024c.

Alex Wang. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Ke Wang, Nikolaos Dimitriadis, Guillermo Ortiz-Jimenez, François Fleuret, and Pascal Frossard. Localizing task information for improved model merging and compression. *arXiv preprint arXiv:2405.07813*, 2024.

BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

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*, pp. 23965–23998. PMLR, 2022.

Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492, 2010. doi: 10.1109/CVPR.2010.5539970.

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.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. *arXiv preprint arXiv:2310.02575*, 2023.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

## A Derivation of Equation 7

Here we derive the squared Frobenius norm of the interference $\lambda \sum_{i \neq k} \boldsymbol{T}_i^k$ in more details:

$$\left\| \lambda \sum_{i \neq k} \boldsymbol{T}_i^k \right\|_F^2 = \left\langle \lambda \sum_{i \neq k} \boldsymbol{T}_i^k, \lambda \sum_{i \neq k} \boldsymbol{T}_i^k \right\rangle_F \tag{15}$$

$$= \lambda^2 \left( \sum_{i \neq k} \sum_{j \neq k} \langle \boldsymbol{T}_i^k, \boldsymbol{T}_j^k \rangle_F \right) \tag{16}$$

$$= \lambda^2 \left( \sum_{i \neq k} \langle \boldsymbol{T}_i^k, \boldsymbol{T}_i^k \rangle_F + \sum_{i,j \neq k} \langle \boldsymbol{T}_i, \boldsymbol{T}_j^k \rangle_F \right) \tag{17}$$

$$= \lambda^2 \left( \sum_{i \neq k} \|\boldsymbol{T}_i^k\|_F^2 + 2 \sum_{\substack{1 \leq i < j \leq n \\ i,j \neq k}} \langle \boldsymbol{T}_i^k, \boldsymbol{T}_j^k \rangle_F \right) \tag{18}$$

$$= \lambda^2 \left( \sum_{i \neq k} \|\boldsymbol{T}_i^k\|_F^2 + 2 \sum_{\substack{1 \leq i < j \leq n \\ i,j \neq k}} \|\boldsymbol{T}_i^k\|_F \|\boldsymbol{T}_j^k\|_F \cos(\boldsymbol{T}_i^k, \boldsymbol{T}_j^k) \right) \tag{19}$$

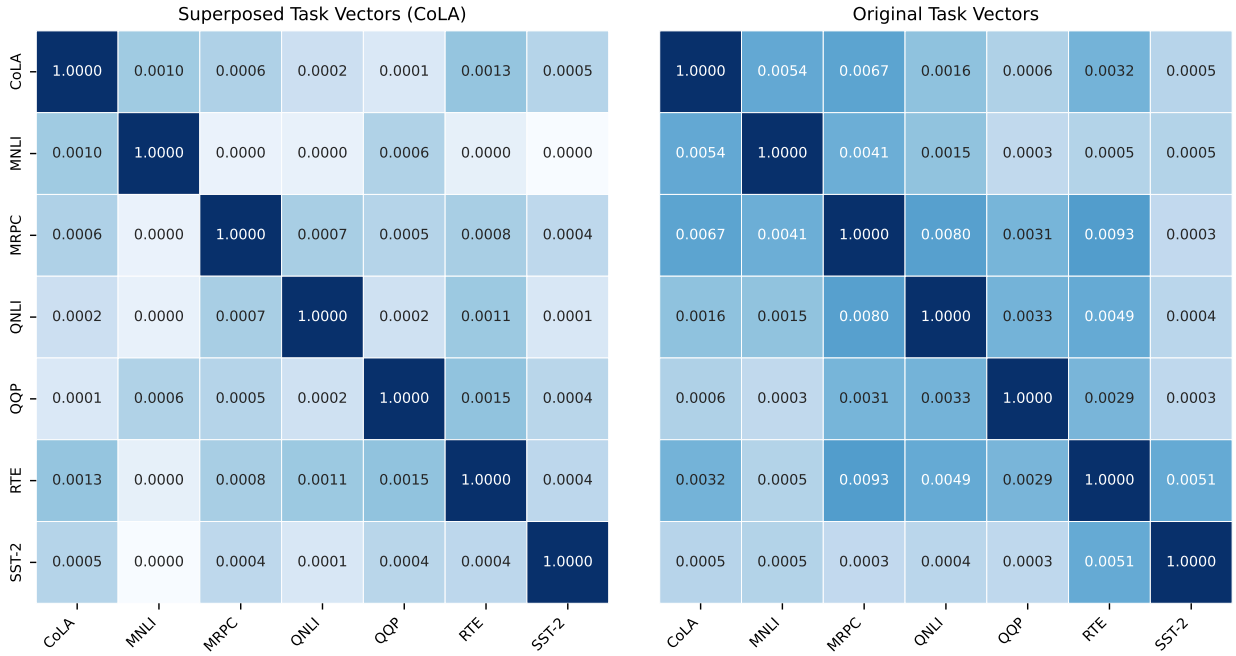



**Figure 6:** Pairwise cosine similarity comparison on GPT-2 models between superposed and original task vectors shifted 1-layer downwards with wraparound.

## B Experiment Setup

**Datasets and Models.** We evaluate our methods on three sets of datasets and models provided by *FusionBench*(Tang et al., 2024a). For text classification and generation, we have in total eight tasks from the GLUE benchmark Wang (2018): CoLA, MNLI, MRPC, QNLI, QQP, RTE, and SST2. For image classification, following (Tang et al., 2024c), we use eight tasks from CLIP's test set: SUN397 Xiao et al. (2010), Cars Krause et al. (2013), RESISC45 Cheng et al. (2017), EuroSAT Helber et al. (2019), SVHN Netzer et al. (2011), GTSRB Stallkamp et al. (2012), MNIST Deng (2012), and DTD Cimpoi et al. (2014). The model being used are: CLIP-ViT-32s (Radford et al., 2021) models with full finetuning on the vision model, Flan-T5-base full finetuning and LoRA finetunes (Longpre et al., 2023), and GPT-2 full finetunes (Radford et al., 2019).

**Baselines and Metrics** We have a extensive set of baselines to compare with, thanks to the well-established model fusion benchmark *FusionBench* (Tang et al., 2024a). For more information about each individual baselines, please refer to *FusionBench*. We also include *Parameter Superposition (PSP)* (Cheung et al., 2019) with binary diagonal matrix as an additional baseline. For evaluation metrics, our primary evaluation metric is the *average accuracy* (Avg. Acc) across all tasks. This is computed by averaging the accuracies of individual tasks on their respective test sets. For the STSB task, we report Spearman's correlation coefficient.

**Default Setup** In our default setup for model superposition, we use the same seed to generate binary diagonal matrix for all target layers within each model. We superpose all layers by default with random shuffling. To estimate variance, we run each experiment three times using different random seeds: 42, 43, 44.

## C Random Matrices in High Dimneional Space

In the following proof, we will provide some motivation about why applying random matrices to similar vectors can lead to them to being more orthogonal in high dimensional space.

Let $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^d$ be two arbitrary vectors, and let $\mathbf{O} \in \mathbb{R}^{d\times d}$ be a random orthogonal matrix. Consider the transformed vectors $\mathbf{v}_1' = \mathbf{O}\mathbf{v}_1$ and $\mathbf{v}_2' = \mathbf{O}\mathbf{v}_2$. We aim to show that the cosine similarity between $\mathbf{v}_1'$ and $\mathbf{v}_2'$ decreases as $d$ increases.

The cosine similarity between the transformed vectors is given by:

$$\cos(\mathbf{v}_1', \mathbf{v}_2') = \frac{\langle \mathbf{v}_1', \mathbf{v}_2' \rangle}{\|\mathbf{v}_1'\| \|\mathbf{v}_2'\|}.$$

Since $\mathbf{O}$ is orthogonal, the norms of the vectors are preserved, i.e., $\|\mathbf{v}_1'\| = \|\mathbf{v}_1\|$ and $\|\mathbf{v}_2'\| = \|\mathbf{v}_2\|$. Thus, we focus on the inner product $\langle \mathbf{v}_1', \mathbf{v}_2' \rangle$.

The matrix $\mathbf{O}$ is drawn from the Haar distribution over the orthogonal group, which means that the entries of $\mathbf{v}_1'$ and $\mathbf{v}_2'$ become uncorrelated as $d$ increases. By properties of random orthogonal matrices, the expected value of the inner product is:

$$\mathbb{E}[\langle \mathbf{v}_1', \mathbf{v}_2' \rangle] = 0.$$

This is because the components of $\mathbf{v}_1'$ and $\mathbf{v}_2'$ behave like independent random variables with zero mean in high dimensions. In high dimensions, the variance of the inner product $\langle \mathbf{v}_1', \mathbf{v}_2' \rangle$ is proportional to $1/d$, meaning that as $d \to \infty$, the inner product tends to zero with high probability:

$$\langle \mathbf{v}_1', \mathbf{v}_2' \rangle \approx 0 \quad \text{as} \quad d \to \infty.$$

Thus, the cosine similarity also tends to zero:

$$\mathbb{P}(\cos(\mathbf{v}_1', \mathbf{v}_2') \approx 0) \to 1 \quad \text{as} \quad d \to \infty.$$

Therefore, applying a random orthogonal matrix $\mathbf{O}$ to two high-dimensional vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ leads to an increase in orthogonality, as the cosine similarity between the transformed vectors decreases to zero with high probability as the dimensionality increases.