Split-Merge: Scalable and Memory-Efficient Merging of Expert LLMs

Sruthi Gorantla[1], **Aditya Rawal**[1], **Devamanyu Hazarika**[1], **Kaixiang Lin**[1], **Mingyi Hong**[1,2], **Mahdi Namazifar**[1]

[1]Amazon AGI, [2]University of Minnesota

Correspondence: srgnt@amazon.com

Abstract

We introduce a zero-shot merging framework for large language models (LLMs) that consolidates specialized domain experts into a single model without any further training. Our core contribution lies in leveraging relative task vectors—difference representations encoding each expert's unique traits with respect to a shared base model—to guide a principled and efficient merging process. By dissecting parameters into common dimensions (averaged across experts) and complementary dimensions (unique to each expert), we strike an optimal balance between generalization and specialization. We further devise a compression mechanism for the complementary parameters, retaining only principal components and scalar multipliers per expert, thereby minimizing overhead. A dynamic router then selects the most relevant domain at inference, ensuring that domain-specific precision is preserved. Experiments on code generation, mathematical reasoning, medical question answering, and instruction-following benchmarks confirm the versatility and effectiveness of our approach. Altogether, this framework enables truly adaptive and scalable LLMs that seamlessly integrate specialized knowledge for improved zero-shot performance.

1 Introduction

Large Language Models (LLMs) have made substantial advancements in recent years, demonstrating proficiency in specialized fields such as natural language processing, code generation, and scientific reasoning (Abdin et al., 2024). Although effective methods for fine-tuning LLMs on individual tasks are well-studied, the process of fine-tuning these models for multiple, distinct tasks remains less understood. The challenges associated with fine-tuning for diverse tasks are complex and multifaceted, from finding the right data mixture proportions to tuning sensitive hyper-parameters such as learning rates, batch sizes, and optimization strategies. Without a comprehensive and well-guided search over these parameters, the model's performance could degrade across tasks.

In fine-tuning, accuracy increases smoothly when shifting a pre-trained model's weights toward its fine-tuned counterpart (Frankle et al., 2020a; Izmailov et al., 2018; Fort et al., 2020; Wortsman et al., 2022a; Choshen et al., 2022; Wortsman et al., 2022c; Matena and Raffel, 2022), and averaging the weights of multiple fine-tuned models can boost performance on those tasks (Li et al., 2022; Choshen et al., 2022; Wortsman et al., 2022b). These observations align with our results of Task Arithmetic in Section 4.

Simply averaging might result in loss of information as highly conflicting parameters across different tasks are canceled out. Recent works such as TIES (Yadav et al., 2024b), DARE (Yu et al., 2024b), try to resolve conflicts using basic arithmetic operations. Conflict resolution may not always be possible. On the other hand, Twin-Merge Lu et al. (2024) maintains a low-rank representation of the task parameters and routes to specific task at inference time. This method does not scale well with the number of experts.

In this paper, we propose a merging method that overcomes the drawbacks of the previous methods. Briefly, our contributions are as follows:

- We propose a novel merging algorithm called `Split-Merge` (see Algorithm 1) inspired by linear algebraic techniques to merge expert LLMs, in a zero-shot fashion, without any further training on data. Our approach takes a set of expert LLMs, each of size d, and combines them into a single model of size $(1+\alpha)d$, where α can be arbitrarily small. The algorithm is designed to effectively integrate the parameters of the expert models, maximizing their individual contributions without introducing redundancy or significantly increasing the overall model size.
- Unlike previous zero-shot merging algorithms,

that resolve interferences in the model parameters using heuristics, we provide theoretical motivation for our approach of resolving (or not resolving) interferences (see Theorem 3).

- We conduct extensive experiments to evaluate the effectiveness of our proposed method. The results are encouraging, showing that our approach consistently outperforms existing baselines for model merging both in terms of accuracy and efficiency across a variety of expert tasks (Section 4).

This work highlights the potential of model merging and opens new avenues for optimizing LLMs across multiple domains. As LLMs grow in size and complexity, developing efficient methods for handling specialized tasks will become increasingly crucial. Model merging represents a key step forward, offering a scalable, effective solution for unifying expert LLMs into a single unified model.

2 Related Works

Motivation from the theory of linear mode connectivity. Model merging techniques have relied on the key property of neural networks called Linear Mode Connectivity (LMC) (Nagarajan and Kolter, 2019; Frankle et al., 2020b; Lubana et al., 2022), which says that, two neural networks trained on the same loss function, from the same initialization, do not have a loss barrier on the line joining two models. Recent work on neural network geometry (Li et al., 2018; Garipov et al., 2018; Draxler et al., 2018; Kuditipudi et al., 2019; Fort et al., 2019; Czarnecki et al., 2019; Wortsman et al., 2021; Benton et al., 2021; Li et al., 2022) shows that even though these models are non-linear, interpolating between two networks that share part of their optimization path can preserve high accuracy. When these assumptions do not hold, recent works (Wortsman et al., 2022a; Ramé et al., 2023; Entezari et al., 2022; Ainsworth et al., 2023; Stoica et al., 2023) have proposed various optimization objectives, the solutions of which bring the models closer to one another to remove the loss barrier between them. In a recent work, (Adilova et al., 2024) demonstrated that LMC can occur layer-wise, even in deep linear networks.

However, little is known about merging models trained on very different domains, such as *math*, *coding*, or *chat* tasks. The key challenge is that each model is fine-tuned for a different loss landscape depending on its own data distribution, hence, LMC may not hold in either of the loss landscapes.

Zero-shot merging. Ilharco et al. (2023) proposed task vectors as the vectors formed by subtracting the pre-trained model parameters from the fine-tuned ones, and analysed several arithmetic operations on these task vectors. Among the zero-shot model merging methods, Yadav et al. (2023); Yu et al. (2024b) resolve conflicts in the task vectors using heursitic algorithms, resulting in a loss of information. On the other hand, Lu et al. (2024) propose to dynamically route to experts models, rather than resolving the conflicts.

Non zero-shot merging. Daheim et al. (2024); Ortiz-Jimenez et al. (2024) proposed gradient matching, and Matena and Raffel (2022) proposed to use the Fisher information between the parameters to define weights of linear merging. Yu et al. (2024a) computes the direction and magnitude of each expert in a linear model merge separately. However, these methods are all dense merges, hence incurring a loss of information on highly conflicting dimensions. A recent survey by Yadav et al. (2024a) studies different characteristics of merging methods that also incorporate routing to expert models.

Mixtrue of Experts. Unlike the zero-shot merging methods, Mixture of Experts (MoE), trains all the experts simultaneously by activating one or a few experts and routing the input to the chosen set of experts (Jacobs et al., 1991; Jordan and Jacobs, 1994; Sukhbaatar et al., 2024). The huge success of MoE is due to the ability to train larger models, while creating experts (Jiang et al., 2024). Several design choices lead to many innovative algorithms (Zadouri et al., 2023; Muqeeth et al., 2023; Fedus et al., 2022b; Lepikhin et al., 2020). We refer the readers to this recent comprehensive survey about the MoE models (Fedus et al., 2022a).

Federated learning. Approaches like FedMix (Reisser et al., 2021) and FedJETs (Dun et al., 2023) in federated learning focus on training different models on private user data while sharing only a router (McMahan et al., 2017), hence, not all the expert models are available simultaneously. Given its similarity to (Matena and Raffel, 2022)–a model merging method, it is easy to see that several Federated learning methods can be applied to model merging, with appropriate adjustments to the algorithms.

3 Method

3.1 Relative Task Vectors

Let $\theta_{\text{pre}} \in \mathbb{R}^d$ denote the vectorized pre-trained model parameters and $\theta_{\text{ft}}^{(t)} \in \mathbb{R}^d$ denote the vectorized model parameters of the t-th expert LLM, where $t = [n]$[1], that is fine-tuned from the pre-trained model. Therefore, the LLMs share the same architecture, allowing us to simplify the notation as well as merge each layer of the architecture independently.

Previous zero-shot merging algorithms defined task vectors (Ilharco et al., 2023; Yadav et al., 2023; Yu et al., 2024b) and implemented arithmetic operations, conflict resolutions, and scaling to come up with a unified model. Formally

Definition 1 (**Task Vector** (Ilharco et al., 2023))**.** *When a pre-trained model with model parameters θ_{pre} is fine-tuned on a domain t to get task-specific model parameters $\theta_{ft}^{(t)}$, its task vector, represented as τ_t, is defined as $\tau_t := \theta_{ft}^{(t)} - \theta_{pre}$.*

Because the pre-trained model may not contain any task-specific knowledge, the task vector could carry a lot of information that can not be expressed meaningfully with less memory. Taking motivation from the literature on linear mode connectivity (LMC) in language models (Adilova et al., 2024), we define *relative task vectors* that capture deviations of the tasks from their average, that captures common knowledge due to LMC. Formally,

Definition 2 (**Relative Task Vectors (Ours)**)**.** *When a pre-trained model with model parameters θ_{pre} is fine-tuned on several expert domains $t = 1, 2, \ldots, n$ to get task-specific model parameters $\theta_{ft}^{(1)}, \theta_{ft}^{(2)}, \ldots, \theta_{ft}^{(n)}$, the shared knowledge is represented by a simple average $\mu = \frac{1}{n}\sum_{t\in[n]} \theta_{ft}^{(t)}$ and the expert knowledge for each domain t is represented as the relative task vector, δ_t, defined as $\delta_t := \theta_{ft}^{(t)} - \mu$.*

Note that Lu et al. (2024) also work with the $\theta_t - \mu$. However, they work with the matrix form of the parameters while we work with the vectorized form. As a result, we design a more memory-efficient and scalable algorithm as described in the next section. Figure 1 shows an example of how relative task vectors could be very different from task vectors depending on the orientation of the pre-trained model.

[1]For any positive integer a, $[a] := \{1, 2, \ldots, a\}$

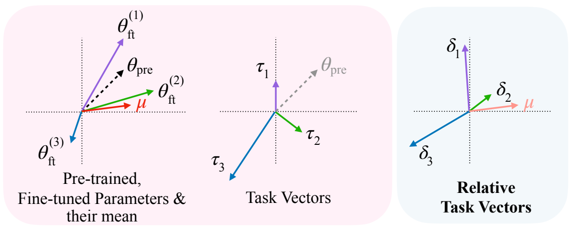

Figure 1: Example showing that the relative task vectors may be oriented very differently than the task vectors.

To make our algorithm scalable to any number of experts and memory efficient, we propose to store the top principal component of the relative task vectors in the top k dimensions where their variance is largest, and a scalar multiplier for each expert, $c_t, \forall t \in [n]$. Here, k is a hyperparameter of the algorithm that is an integer in the range $[0, d]$. First, we construct the ordered set of indices, ψ, for each $i \in [k]$:

$$\psi(i) \leftarrow \underset{j\in[d]\setminus\{\psi(1),\ldots,\psi(i-1)\}}{\arg\max} \operatorname{Var}(\delta_{1,j}, \ldots, \delta_{n,j}).$$

where $\delta_{t,i}$ is the i-th entry of the task vector δ_t, and $\operatorname{Var}(\delta_{1,i}, \delta_{2,i}, \ldots, \delta_{n,i})$ denotes their variance. We then project the relative task vectors onto the subspace formed by the dimensions in ψ, by keeping the values in dimensions in the ordered set ψ. Let the resulting relative task vector for task $t \in [n]$ be denoted by $\hat{\delta}_t$.

$$\hat{\delta}_{t,i} = \delta_{t,\psi(i)}, \quad \forall i \in [k], \forall t \in [n].$$

We now want a memory-efficient representation of these projected vectors using a single vector. To minimize the loss of information due to this compression, we propose to use their first principal component as the unified single vector, $\mathbf{v}^*$, as it minimizes the reconstruction error defined by the following objective function,

$$\mathbf{v}^* = \arg \min_{\mathbf{v}\in\mathbb{R}^d, \|\mathbf{v}\|=1} \sum_{t=1}^{T} \left\| \hat{\delta}_t - (\mathbf{v}^\top \hat{\delta}_t) \cdot \mathbf{v} \right\|^2 \quad (1)$$

We then compute the lengths of the projections of each of δ_t onto the principal component to reconstruct the task vectors in the remaining dimensions, which has minimal reconstruction error. The projections (scalars) can be computed as

$$c_t = (\mathbf{v}^*)^\top \hat{\delta}_t, \quad \forall t \in [n].$$

Since we need the principal component to be in d dimensions, we construct another vector η in $\mathbb{R}^d$,

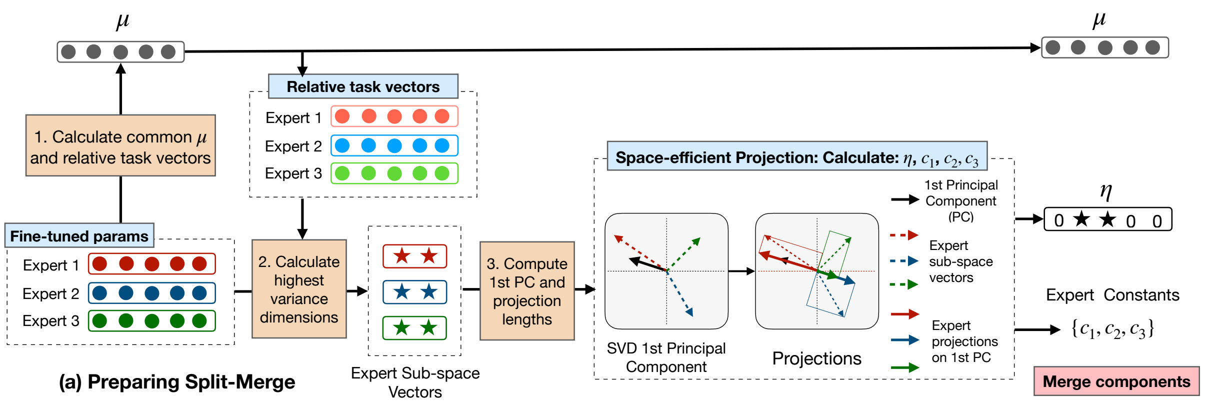
μ
1. Calculate common μ and relative task vectors
Fine-tuned params
Expert 1
Expert 2
Expert 3
Relative task vectors
2. Calculate highest variance dimensions
Expert Sub-space Vectors
3. Compute 1st PC and projection lengths
Space-efficient Projection: Calculate: η, c1, c2, c3
SVD 1st Principal Component
Projections
1st Principal Component (PC)
Expert sub-space vectors
Expert projections on 1st PC
η
0 ★ ★ 0 0
Expert Constants
{c1, c2, c3}
Merge components
(a) Preparing Split-Merge

Figure 2: Computing merge components with Split-Merge.

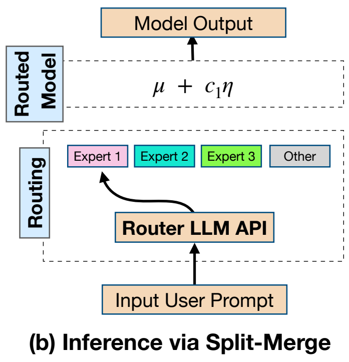

Figure 3: Inference with Split-Merge

using $\mathbf{v}^*$, to undo the projection. That is,

$$\eta(\psi(i)) = \mathbf{v}^*(i),\ \forall_{i\in[k]} \text{ and } \eta(i) = 0, \forall_{i\in[d]\setminus\psi}.$$

Therefore, vectors μ, η, and the scalars $c_1, c_2, \ldots, c_n$ form the components of our merging algorithm. Using this approach, we can reduce the memory requirement to $d + k + n$ for the merged model, d to store μ, k to store η, and n to store scalars, $c_1, c_2, \ldots, c_n$. Figure 2 shows the inner workings of our algorithm on a toy example with three expert models in $\mathbb{R}^5$ fine-tuned from the same base model for different domains. Note that none of the operations done on the task vectors to construct a merged model require collecting expensive training data. Hence, our method is **zero-shot**.

Inference. An off-the-shelf router model θ_{router} categorizes the input prompt x into one of the n domains or ‘other’. The final model used for inference is constructed as $\mu + c_t\eta$, if $t \in [n]$. If the input is classified as ‘other’, we use the base model, μ. This can be achieved by setting $c_0 = 0$ and setting the ‘other’ class to map to this value (see Figure 3). Algorithm 1 outlines a pseudo code of the algorithm.

3.2 Choice of k

The choice of k depends on the underlying loss landscapes of the expert models. Several properties of the merged model can be controlled using k.

Let $\eta(k)$ be the principal component and $\hat{\delta}_t(k)$ be the projected task vectors computed for k in Algorithm 1. By definition, any other vector, including the one that minimizes the reconstruction error of the vectors $\hat{\delta}(k')$ for any $k' > k$, has to have a reconstruction error equal to or more than that of $\mathbf{v}^*$ for k. Hence, constructing a principal component for the most varying k dimensions is more beneficial than constructing a principal component on the most varying k' dimensions, whenever $k < k'$, to capture variances in the top k dimensions better.

On the other hand, a larger value of k ensures that the common component μ is not performing averaging on highly conflicting dimensions leading to a higher increase in the overall loss. Consider the following theorem (proof in Appendix A).

Theorem 3. *Consider L-Lipschitz loss functions $\mathcal{L}_1, \ldots, \mathcal{L}_n$ for all the expert domains, for some constant L. Let $\mathcal{S} \subset [d]$ be any subset of dimensions. Let $\mu = \frac{1}{n}(\theta_{ft}^{(1)} + \theta_{ft}^{(2)} + \cdots + \theta_{ft}^{(n)})$. Let $\hat{\theta}_t$ be the model t after partial merging. That is, $\hat{\theta}_t(i) := \mu(i),\ \forall\ i \in \mathcal{S}$ and $\theta_{ft}^{(t)}(i),\ \forall\ i \notin \mathcal{S}, \forall t \in [n]$. Then, $\|(\Delta\mathcal{L}_1, \ldots, \Delta\mathcal{L}_n)\|_2 \le L \cdot \sqrt{\sum_{i\in\mathcal{S}} \mathrm{Var}(\delta_{1,i}, \ldots, \delta_{n,i})}$. where $\Delta\mathcal{L}_t := \mathcal{L}_t(\theta_{ft}^{(t)}) - \mathcal{L}_t(\hat{\theta}_t)$ for all $t \in [n]$.*

This theorem says that under some smoothness assumptions of the loss functions of the experts, the variance of the k least varying dimensions gives us an upper bound on the overall increase in the loss in terms of an ℓ_2 norm. Hence, choosing $\mathcal{S}$ to be a small set of least varying dimensions of the relative task vectors gives a good upper bound

Algorithm 1 Algorithm 1: Split-Merge

```
Input: $\theta_{\text{ft}}^{(1)} \in \mathbb{R}^d, \forall t \in [n]$, $k \in [d]$, a router
       model $\theta_{\text{router}}$, and a set of prompts $\mathcal{X}$.
Output: A set of outputs $\mathcal{Y}$.

// Constructing merge components
Compute shared component: $\mu = \frac{1}{n}\sum_{t\in[n]} \theta_{\text{ft}}^{(t)}$.
Compute relative task vectors: $\delta_t = \theta_{\text{ft}}^{(t)} - \mu, \forall t$.
Initialize: $\psi(i) = 0, \forall i \in [k]$
for $i = 1, 2, \ldots, k$ do
    Compute the top $i$-th variance dimension:
      $\psi(i) \leftarrow \underset{j\in[d]\setminus\{\psi(1),\ldots,\psi(i-1)\}}{\arg\max} \operatorname{Var}(\delta_{1,j}, \ldots, \delta_{n,j})$
end
Project onto $\mathbb{R}^k$ formed by dimensions in $\psi$:
      $\hat{\delta}_{t,i} \leftarrow \delta_{t,\psi(i)}, \forall i \in [k], \forall t \in [n]$.
Compute top PC as in Eq. (1):
  $\mathbf{v}^* \leftarrow PCA(\hat{\delta}_1, \hat{\delta}_2, \ldots, \hat{\delta}_n)$.
Compute coefficients: $c_t \leftarrow (\mathbf{v}^*)^\top \hat{\delta}_t, \ \forall t \in [n]$.
Set $c_0 = 0$ for base model, $\mu$.
Initialize $\eta$ as a zero vector in $\mathbb{R}^d$.
Project back to $\mathbb{R}^d$: $\eta(\psi(i)) \leftarrow \mathbf{v}^*(i), \ \forall i \in [k]$

// Inference
Initialize: $\mathcal{Y} = \emptyset$.
for an input prompt $x \in \mathcal{X}$ do
    Classify domain within $[n] \cup \{0\}$:
      $t = \theta_{\text{router}}(x)$.
    Perform inference: $y = (\mu + c_t\eta)(x)$.
    Append: $\mathcal{Y} = \mathcal{Y} \cup \{y\}$.
end
Return $\mathcal{Y}$.
```

on the overall increase in the loss functions, implying that larger k is better. Therefore, there is a sweet spot for the parameter k that achieves the best performance, while being memory-efficient.

4 Experiments

We perform experiments using two different pre-trained language models, and various expert domains, to analyze the scalability and memory-efficiency of our algorithms across different architectures and different number of experts.

4.1 Setup

Varying different architectures and domains. We perform experiments using two different pre-trained language models – `Qwen2.5-1.5B` (Team, 2024) and `Mistral-7B-v0.1` (Jiang et al., 2023) to observe results across *different architectures* and *sizes*. The Qwen pre-trained model is fine-tuned on four domains: (a) **Math**: Tasks related to symbolic math, arithmetic, and theorem proving; (b) **Coding**: Programming-related tasks, including code generation and editing; (c) **General Instruction Following**: Tasks related to following user instructions, and (d) **MedQA**: Tasks related to multiple choice questions from the medical domain. We use the `Qwen2.5-1.5B-Instruct`[2] as the instruction following expert, but train all other experts using supervised fine-tuning on the pre-trained model. The setup of Mistral model is deferred to Appendix B.2.

Varying Expert Domains. We use the `nickrosh/Evol-Instruct-Code-80k-v1` (Luo et al., 2023), `meta-math/MetaMathQA` (Yu et al., 2024c), and `bigbio/med_qa` (Jin et al., 2021) datasets to train the Coding, Math, and Medical experts, all available on Huggingface. We **do not** perform any modifications on these datasets. We evaluate all the merged models on relevant public benchmarks: HumanEval-Python (Chen et al., 2021), GSM8K (Cobbe et al., 2021), MedQA (Jin et al., 2021), and IFEval (Zhou et al., 2023).

Varying Routers. We experiment with several router models for Split-Merge as well as other routing-based baselines. We experiment with two LLMs as routers: (a) Mistral-7B-Instruct-v0.2 (Chaplot et al., 2023) (`M-7b`) and (b) Mixtral-8x7B-Instruct-v0.1 (AI, 2023) (`M-8x7b`), where we expect the latter to be a stronger router than the former. For prompting these LLM-Routers for routing, we adopt a simple multiple-choice prompt where we ask the LLM-router to classify the expert category of a user prompt. The prompt we use is detailed in Appendix B.4. Note that we did not train any router ourselves. (c) **Random Routing**: Additionally, we also introduce a *random* router that randomizes the routing decisions across the categories to stress-test our algorithm. (d) **Oracle Routing**: Finally, oracle routing means we select the appropriate expert model for the given user prompts, e.g., math expert for math prompt. We use the same routing setup for Twin-Merge.

[2]https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct

Method	**GSM8K**	**IFEval**	**MedQA**	**H-Eval**	Overall
	(acc)	(acc)	(acc)	(acc)	Eq. (2)
(#samples)	(1319)	(541)	(1273)	(164)	(3297)
Pre-trained	0.6209	0.3106	0.2781	0.0	0.6753
Math Expert	0.6331	0.3945	0.4171	0.1098	0.8597
IF Expert	0.3093	**0.5012**	0.4627	0.2317	0.7394
Med Expert	0.4898	0.3573	**0.5075**	0.2744	0.8455
Code Expert	0.5519	0.3118	0.443	0.4146	0.8376
Routing to **LoRA** (**rank = 16**) Experts as baseline:					
↬ Oracle	0.6543	$\underline{0.4412}$	0.4258	0.3963	0.9294
Zero-shot Dense Merging baselines:					
TA	0.655	0.3897	0.4886	$\underline{0.4634}$	0.9688
TIES	0.6687	0.3945	0.4792	0.4329	0.9682
DARE-TA	0.6679	0.3897	0.4918	0.4573	0.9787
DARE-TIES	0.6459	0.3897	0.3629	0.3537	0.8543
Model Stock	0.6399	0.3297	0.2828	0.0183	0.7296
Zero-shot Routing-based Merging baseline: **Twin-Merge** (**rank = 16**)					
↬ Oracle	0.6073	0.4317	0.4878	0.3902	0.9430
↬ `M-7b`	0.6232	0.4020	0.4790	0.3780	0.9352
↬ `M-8x7b`	0.6588	0.4140	0.4800	0.3902	0.9638
↬ Random	0.6361	0.3980	0.4530	0.2866	0.9113
Our algorithm: **Split-Merge** (**density = 0.95**)					
↬ Oracle	**0.6755**	0.3777	0.4902	**0.4817**	$\underline{0.9812}$
↬ `M-7b`	$\underline{0.6694}$	0.3729	0.4965	**0.4817**	0.9806
↬ `M-8x7b`	0.6687	0.3765	0.4965	**0.4817**	**0.9814**
↬ Random	0.6664	0.3765	$\underline{0.4996}$	0.4695	0.9808

Table 1: The performance of the Qwen **expert models** and **merged models** on the benchmarks. We **bold** the number of the highest performing model and underline the close second. Note: H-Eval refers to HumanEval. In the first column, ↬ `<model>` represents the usage of a `<model>` for routing the input prompt to the appropriate expert component in the merged model. See Equation (2) for the metric in last column.

4.2 Results of Split-Merge and Baselines

Table 1 reports results of various merge methods. Lu et al. (2024) defined an average normalized scoring function (Eq. 4 in the paper) to compare various merged models using a unified score. However, their score takes a simple average of the normalized score per domain. To account for varying test set sizes, we use a slightly different version of their score where we weigh the domain-wise normalized score with the relative sizes of the datasets,

$$\text{Wt. Norm. Score}(\theta) := \sum_{t=1}^{n} \frac{N_t}{N} \cdot \frac{\underset{x\sim\mathcal{D}_t}{\text{Score}}[f(x;\theta)]}{\underset{x\sim\mathcal{D}_t}{\text{Score}}[f(x;\theta_{\text{ft}}^{(t)})]}, \tag{2}$$

where $N_t = |\mathcal{D}_t|$ and $N = \sum_{t'=1}^{n} |\mathcal{D}_{t'}|$. This score, always ≥ 0, typically falls below 1; a value $= 1$ indicates that the merged model matches oracle-routed experts, and values > 1 suggest surpassing them (though this is uncommon due to model compression). When all the datasets

Method	**Qwen2.5-1.5B**	**Mistral-7B-v0.1**	**Avg.**
	4 tasks	3 tasks	
Pre-trained	0.6753	0.3620	0.5186
Fine-tuned	1	1	1
TA	0.9688	0.8951	0.9319
TIES	0.9682	0.9101	0.9392
DARE-TA	$\underline{0.9787}$	0.8259	0.9023
DARE-TIES	0.8543	0.7967	0.8255
Twin ↬ `M-8x7b`	0.9638	$\underline{0.9209}$	$\underline{0.9424}$
Split ↬ `M-8x7b`	**0.9814**	**0.9264**	**0.9539**

Table 2: Average of the Wt. Norm. Scores across different architectures with different number of experts.

are of equal size, Normalized Score in (Lu et al., 2024) and Wt. Norm. Score in Equation (2) are equal. We use **Overall Score** interchangeably with Wt. Norm. Score.

Expert Models. We begin with a Pre-trained base model, then produce specialized expert models for each domain: Math Expert, Instruct Expert, MedQA Expert, and Coding Expert. As expected, each expert excels primarily on its own domain (see Table 1)–for instance, the Math Expert achieves higher accuracy on GSM8K, while the MedQA Expert does best on MedQA.

Routing to LoRA Experts. One natural baseline is to consider the parameter-efficient fine-tuning of expert models using techniques like LoRA (Hu et al., 2022), and using the router model to route the input prompt to the corresponding experts during inference. This gives us a memory-efficient baseline as the experts are all low-rank and do not need any further processing. We observe that the LoRA experts perform poorly on almost all of the benchmarks, and get a poor overall score compared to other methods.

Zero-shot Dense Merging Baselines. Task Arithmetic (TA) (Ilharco et al., 2023), TIES (Yadav et al., 2023), DARE-TA, DARE-TIES (Yu et al., 2024b), and Model Stock (Jang et al., 2024) merge the domain-specific experts without further training on combination of strategies. While these baselines have better overall scores than any single expert's performance (especially for tasks that rely on multiple capabilities), they under-perform compared to the router-based zero-shot methods (Twin-Merge and Split-Merge) as they are restricted to resolve conflicts forcing the merged model to lose information on conflicting dimensions.

Routing-Based Zero-Shot Merging (Twin-Merge (Lu et al., 2024) with rank = 16). Next, we report the results of the Twin-Merge, which can be paired with various "routers" to decide which expert's parameters to leverage. Twin-Merge achieves decent results (e.g., 0.9638 overall with the `M-8×7b` router), confirming that routing-based expert selection can provide a unified model.

Our Algorithm: Split-Merge ($\alpha = 0.95$). For the sake of experiments, we use α to be the **density** parameter of our algorithm where $k = \alpha * d$ in Algorithm 1. The lower block of results in Table 1 shows our proposed Split-Merge method, again using several router strategies (oracle, M-7b, M-8×7b, random). Across nearly all task-specific metrics and the Overall Score, Split-Merge exceeds the previous baselines and Twin-Merge. A few observations stand out:

- **Consistently High Performance:** All Split-Merge variants exceed the strongest zero-shot baselines and Twin-Merge results with the Weighted Normalized Score higher than DARE-TA and Twin-Merge $\looparrowright$ M–8×7b.
- **Robustness to Routers:** Perhaps surprisingly, a random router yields an overall score marginally below the best from the M–8×7b router suggesting Split-Merge's parameter-splitting mechanism provides a strong foundation.
- **Per-Task vs. Aggregate:** On a task-by-task basis, Split-Merge sometimes ties or slightly underperforms on IFEval. However, it makes up ground on other tasks (notably coding), giving it a higher aggregate score.
- **Best Average Results across Architectures** Table 2 shows that when we average the weighted normalized scores over the Qwen and the Mistral models, Split-Merge beats all the baselines, even though some of the baselines seem strong on one of these architectures (for example, DARE-TA on Qwen). Therefore, our results show that our method is robust across model architectures, model sizes, and the number of experts.

4.3 Memory Requirements of Split-Merge

While both Twin-Merge (Lu et al., 2024) and Split-Merge maintain an expert representation, the key difference is in the compression technique. They use SVD-based reduction, compressing each expert independently. As a result, memory requirements grow much faster for Twin-Merge scales with the number of experts than Split-Merge.

Toy example. Consider a model architecture with just one parameter matrix of dimensions $d_{\text{in}} \times d_{\text{out}}$, hence having a total of $n \cdot d_{\text{in}} \cdot d_{\text{out}}$ parameters for n experts. The SVD operation with rank r in Twin-Merge reduces the final parameter count to $d_{\text{in}} \cdot d_{\text{out}} + nr(d_{\text{in}} + d_{\text{out}} + 1),$ where $r \in [1, \min\{d_{\text{in}}, d_{\text{out}}\}]$. In contrast, Split-merge will have a final parameter count of $(2 - \alpha) \cdot d_{\text{in}} \cdot d_{\text{out}} + n,$ where $\alpha \in [0, 1]$.

Fair comparison between Twin- and Split-Merge. Given a fixed number of expert domains n, for every rank value r for Twin-Merge, there is a corresponding density value α for Split-Merge such that they both have roughly equal memory. To analyze the cases when Split-Merge is more memory-efficient than Twin-Merge, we plot the *efficiency ratio*, defined in Equation (3), by varying Twin's rank, Split's density, and number of experts.

$$\text{Efficiency ratio}(n, \alpha, r) := \frac{\text{Memory(Split; }\alpha, n)}{\text{Memory(Twin; }r, n)}. \tag{3}$$

Figure 4 plots efficiency ratio for `Qwen2.5-1.5B`. As the number of expert domains increases, say 64 experts, we observe that even for very small values of rank (say 2) Twin-Merge has larger memory than Split-Merge with many values of density parameter ($\alpha \leq 0.97$). Therefore, Split-Merge scales better with the number of experts. See Figure 7 for efficiency ratio of `Mistral-7B-v0.1`. We also plot in Figure 5 the comparison of memory requirements and weighted normalized score of Split-Merge and Twin-Merge with different hyperparameter values for 4 Qwen experts, when used with an Oracle router. Clearly Split-merge achieves higher performance with fewer parameters.

Zero-shot dense merging baselines. Task Arithmetic, TIES, DARE, and Model Stock would only need $d_{\text{in}} \cdot d_{\text{out}}$ many parameters, hence being the most memory-efficient merges; however, their performance is lower than our method.

LoRA Routing baseline. The memory requirements of a LoRA model with rank r for all the n experts will be similar to those of the Twin-merge with rank r of n experts. This is because both maintain independent rank r decompositions of n experts. Therefore, the efficiency ratio of Split with Twin-Merge will be similar to the efficiency ratio fo Split with LoRA expert routing. In our experiments, we also compare Split-Merge with density parameter $\alpha = 0.95$ with rank 16 LoRA experts

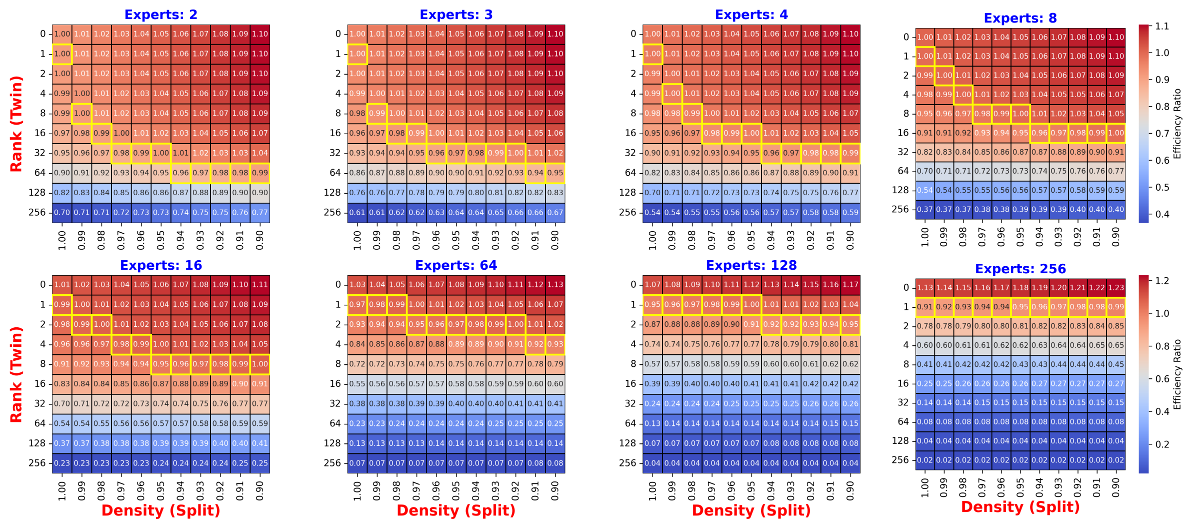
Figure 4: Each cell in this subfigure represents *efficiency ratio*–the ratio of memory requirements of Split-merge to that of Twin-merge. For example, a ratio of 1 indicates equal memory usage and > 1 means Split-Merge uses more. Within each subfigure, we plot this ratio while varying Twin's **rank** and Split-Merge's **density**, and across subfigures we vary the **number of experts**. The **yellow boxes** highlight $(rank, density)$ settings where ratio ≈ 1. In experiments, we compare Twin and Split merges where the ratio is ≤ 1, ensuring Twin has at least as much memory as Split-Merge for a fair comparison.

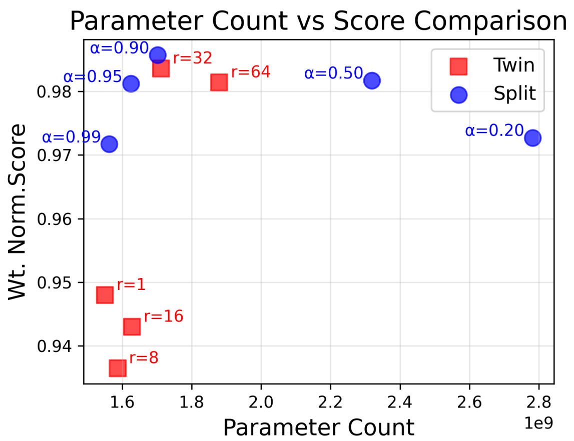

Figure 5: Memory vs. Score comparison of Split-Merge (with varying density parameter, α) and Twin-Merge (with varying rank parameter, r). The top left corner indicates highest score with lowest memory, mostly occupied by the Split-Merge variants.

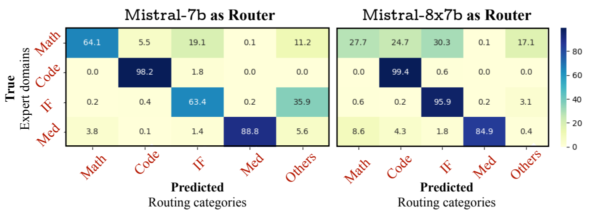

Figure 6: Routing prediction accuracy for 4 domains (Math, Code, IF, and Med) when the prompt is categorized into one of these domains or 'others', using `Mistral-7b` and `Mistral-8x7b` as routers.

and this setting gives us efficiency ratio ≤ 1, indicated by the yellow box in Figure 4 for 4 experts.

4.4 Varying Router Model

Routing Analysis. Table 6 shows the performance of each expert model constructed by the algorithm (before routing), called `Split-Merge w/ [Expert]`, all of which have high overall performance compared to the pre-merge experts. Therefore, our algorithm is very robust to the errors in routing. Figures 6 and 8 summarize the routing predictions, where we observe that the LLM-routers are highly effective in identifying the relevant expert categories given user prompts. For example, `Mistral-7b` router achieves 64.1% routing accuracy in Math and 98.2% routing accuracy in Code. Between the two LLM-routers, we observe that `M-8x7b` generally has better accuracy in routing compared to `M-7b`, except for the math domain. Interestingly, routing errors does not imply a drop in performance (See last block of results in Table 1). In fact, in Table 5 we demonstrate how some prompts from one domain can be logically classified as another domain. Appendix B.4 demonstrates how we test various prompt complexities for different number of experts. Surprisingly, our algorithm has good performance even if we randomly route the prompts (last row of Table 1).

4.5 Generalization

We report MT-Bench results of some of the models to understand **generalization** to unseen domains in Table 7. The results show that Twin/Split Expert components mostly retain the generalization abilities of the fine-tuned models.

5 Conclusion

We introduced a novel zero-shot merging algorithm that efficiently integrates multiple domain-specific expert models into a unified model. By leveraging relative task vectors to decompose the model parameters into shared and domain-specific components, we are able to maintain expert-level performance across distinct domains. Our approach incorporates principal component analysis for dimensionality reduction and a router model to dynamically select the most relevant domain for each input. Extensive experiments demonstrate that our method outperforms existing zero-shot merging techniques, providing a highly effective and scalable solution without the need for additional retraining or inference steps.

Limitations

While our proposed merging framework demonstrates strong zero-shot performance across diverse domains, this study is constrained by the availability of computational resources. In particular, the number of experts, model sizes, and tasks considered in our experiments were limited to what was feasible under our compute budget. As a result, we were unable to explore several promising directions, such as scaling the framework to a larger number of experts, incorporating more diverse or multilingual domains, or evaluating the performance of `Split-Merge` on significantly larger architectures. Exploring more sophisticated routing mechanisms, potentially based on multi-task learning or reinforcement learning, could improve domain selection accuracy in more complex tasks. Merging models with differing architectures or sizes remains an open challenge.

Ethical Considerations

This paper introduces research intended to advance the field of machine learning by proposing new methods and insights. As with many developments in this area, there may be a range of societal implications depending on how the work is applied in practice. However, after careful consideration, we do not identify any specific risks or concerns that need to be highlighted at this stage. We encourage future users and researchers to consider the broader impacts of deploying this work in real-world settings.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, and 1 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Linara Adilova, Maksym Andriushchenko, Michael Kamp, Asja Fischer, and Martin Jaggi. 2024. Layer-wise linear mode connectivity. In *The Twelfth International Conference on Learning Representations*.

Mistral AI. 2023. Mixtral of experts. Accessed: 2024-10-09.

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2023. Git re-basin: Merging models modulo permutation symmetries. *Preprint*, arXiv:2209.04836.

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A framework for the evaluation of code generation models. `https://github.com/bigcode-project/bigcode-evaluation-harness`.

Gregory Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Gordon Wilson. 2021. Loss surface simplexes for mode connecting volumes and fast ensembling. In *International Conference on Machine Learning (ICML)*. `https://arxiv.org/abs/2102.13042`.

Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, and 1 others. 2023. Albert q. jiang, alexandre sablayrolles, arthur mensch, chris bamford, devendra singh chaplot, diego de las casas, florian bressand, gianna lengyel, guillaume lample, lucile saulnier, lélio renard lavaud, marie-anne lachaux, pierre stock, teven le scao, thibaut lavril, thomas wang, timothée lacroix, william el sayed. *arXiv preprint arXiv:2310.06825*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. 2022. Fusing finetuned models for better pretraining. https://arxiv.org/abs/2204.03044.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Wojciech Marian Czarnecki, Simon Osindero, Razvan Pascanu, and Max Jaderberg. 2019. A deep neural network's loss surface contains every low-dimensional pattern. https://arxiv.org/abs/1912.07559.

Nico Daheim, Thomas Möllenhoff, Edoardo Ponti, Iryna Gurevych, and Mohammad Emtiyaz Khan. 2024. Model merging by uncertainty-based gradient matching. In *The Twelfth International Conference on Learning Representations*.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. 2018. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/1803.00885.

Chen Dun, Mirian Hipolito Garcia, Guoqing Zheng, Ahmed Hassan Awadallah, Robert Sim, Anastasios Kyrillidis, and Dimitrios Dimitriadis. 2023. Fedjets: Efficient just-in-time personalization with federated mixture of experts. *Preprint*, arXiv:2306.08586.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. 2022. The role of permutation invariance in linear mode connectivity of neural networks. In *ICLR*.

William Fedus, Jeff Dean, and Barret Zoph. 2022a. A review of sparse expert models in deep learning. *ArXiv*, abs/2209.01667.

William Fedus, Barret Zoph, and Noam Shazeer. 2022b. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. 2020. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/2010.15110.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. 2019. Deep ensembles: A loss landscape perspective. https://arxiv.org/abs/1912.02757.

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

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2020b. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/1802.10026.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. *CoRR*, abs/1803.05407.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. 2024. Model stock: All we need is just a few fine-tuned models. *Preprint*, arXiv:2403.19522.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421.

Michael I Jordan and Robert A Jacobs. 1994. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Rong Ge, and Sanjeev Arora. 2019. Explaining landscape connectivity of low-cost solutions for multilayer nets. *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/1906.06247.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/1712.09913.

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

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

Ekdeep Singh Lubana, Eric J Bigelow, Robert P Dick, David Krueger, and Hidenori Tanaka. 2022. Mechanistic mode connectivity. https://arxiv.org/abs/2211.08422.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *Preprint*, arXiv:2306.08568.

Michael S Matena and Colin A Raffel. 2022. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Mohammed Muqeeth, Haokun Liu, and Colin Raffel. 2023. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745*.

Vaishnavh Nagarajan and J. Zico Kolter. 2019. Uniform convergence may be unable to explain generalization in deep learning. In *Neural Information Processing Systems*.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. 2024. *Task arithmetic in the tangent space: improved editing of pre-trained models*. Curran Associates Inc., Red Hook, NY, USA.

Alexandre Ramé, Guillaume Couairon, Mustafa Shukor, Corentin Dancette, Jean-Baptiste Gaya, Laure Soulier, and Matthieu Cord. 2023. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. In *NeurIPS*.

Matthias Reisser, Christos Louizos, Efstratios Gavves, and Max Welling. 2021. Federated mixture of experts. *Preprint*, arXiv:2107.06724.

George Stoica, Daniel Bolya, Jakob Bue Bjorner, Taylor N. Hearn, and Judy Hoffman. 2023. Zipit! merging models from different tasks without training. *ArXiv*, abs/2305.03053.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Roziere, Jacob Kahn, Shang-Wen Li, Wen tau Yih, Jason E Weston, and Xian Li. 2024. Branch-train-mix: Mixing expert LLMs into a mixture-of-experts LLM. In *First Conference on Language Modeling*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. 2021. Learning neural network subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022a. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and 1 others. 2022b. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR.

Mitchell Wortsman, Gabriel Ilharco, Mike Li, Jong Wook Kim, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. 2022c. Robust fine-tuning of zero-shot models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. https://arxiv.org/abs/2109.01903.

Prateek Yadav, Colin Raffel, Mohammed Muqeeth, Lucas Caccia, Haokun Liu, Tianlong Chen, Mohit Bansal, Leshem Choshen, and Alessandro Sordoni. 2024a. A survey on model moerging: Recycling and routing among specialized experts for collaborative learning. *arXiv preprint arXiv:2408.07057*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024a. Extend model merging from fine-tuned to pre-trained large language models via weight disentanglement. *Preprint*, arXiv:2408.03092.

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

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024c. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. 2023. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. *arXiv preprint arXiv:2309.05444*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

A Missing Proofs

A.1 Proof of Theorem 3

Proof of Theorem 3. Given, $\theta_{\text{ft}}^{(1)}, \theta_{\text{ft}}^{(2)} \in \mathbb{R}^d$ be the model parameters of the fine-tuned expert model, and $\theta_{\text{pre}} \in \mathbb{R}^d$ be the common base model. Let $\mathcal{S} \subset [n]$ be any subset of size k that are chosen to apply partial merging. We want to show that applying a partial merge on the dimensions chosen in $\mathcal{S}$ gives us a good upper bound on the increase in the loss functions of both models. Let $\mu = \frac{1}{n}(\theta_{\text{ft}}^{(1)} + \theta_{\text{ft}}^{(2)} + \cdots + \theta_{\text{ft}}^{(n)})$. Let $\hat{\theta}_t$ be the model t after partial merging. That is,

$$\hat{\theta}_t(i) := \begin{cases} \mu(i) & ; \ \forall \ i \in \mathcal{S} \\ \theta_{\text{ft}}^{(t)}(i) & ; \ \forall \ i \notin \mathcal{S}, \end{cases} \qquad \forall t \in [n].$$

From the assumption that the loss functions used to fine-tune both the experts are L-Lipschitz, for some constant L, we get that,

$$\Delta \mathcal{L}_t(\theta_{\text{ft}}^{(t)}, \hat{\theta}_t) := \mathcal{L}_t(\hat{\theta}_t) - \mathcal{L}_t(\theta_{\text{ft}}^{(t)}) \le L \|\theta_{\text{ft}}^{(t)} - \hat{\theta}_t\|_2. \tag{4}$$

where $\mathcal{L}_t$ is the loss function of the experts t.

Now, note that for any $t \in [n]$,

$$\begin{aligned} \|\theta_{\text{ft}}^{(t)} - \hat{\theta}_t\|_2^2 &= \sum_{i \in \mathcal{S}} \left(\theta_{\text{ft}}^{(t)}(i) - \mu(i)\right)^2 + \sum_{i \notin \mathcal{S}} \left(\theta_{\text{ft}}^{(t)}(i) - \theta_{\text{ft}}^{(t)}(i)\right)^2 & (5) \\ &= \sum_{i \in \mathcal{S}} \text{Var}(\theta_{\text{ft}}^{(1)}(i), \theta_{\text{ft}}^{(2)}(i), \ldots, \theta_{\text{ft}}^{(n)}(i)) & (6) \\ &= \sum_{i \in \mathcal{S}} \text{Var}(\theta_{\text{ft}}^{(1)}(i) - \mu(i), \theta_{\text{ft}}^{(2)}(i) - \mu(i), \ldots, \theta_{\text{ft}}^{(n)}(i) - \mu(i)) & (7) \\ &= \sum_{i \in \mathcal{S}} \text{Var}(\delta_{1,i}, \delta_{2,i}, \ldots, \delta_{n,i}) & (8) \end{aligned}$$

Therefore,

$$\| (\Delta\mathcal{L}_1, \Delta\mathcal{L}_2, \ldots, \Delta\mathcal{L}_n) \|_2 \le L \cdot \sqrt{\sum_{i \in \mathcal{S}} \text{Var}(\delta_{1,i}, \delta_{2,i}, \ldots, \delta_{n,i})}.$$

Hence, choosing $\mathcal{S}$ to be a small set of least varying dimensions of the relative task vectors gives a good upper bound on the overall increase in the loss functions. $\square$

B Additional Experimental Details

B.1 Hyperparameters

We fine-tuned all the experts with learning rate $2e - 5$ and global batch size 16 for one epoch. This created experts as shown in Table 1. For our algorithm and the baselines, we do a grid search on the density parameter and choose the model with the best performance across the benchmarks. For Algorithm 1 we vary the parameter $\alpha \in \{0.1, 0.2, \ldots, 0.9, 1.0\}$. For any given α in this range, we can set $k = \lceil (1 - \alpha) \cdot d \rceil$ in Algorithm 1, where d is the number of parameters in the model. Hence, in the implementation of the algorithm we use α instead of k. We observe that $\alpha = 0.1$, that is doing a linear merge on 90% of the least varying dimensions and leaving the rest 10% as conflicting dimensions works the best for our algorithm. However, each expert may need a different value of k where `SPLIT w/ Expert` performs the best. This is because some expert domains may share more common knowledge than others, i.e, doing the partial linear merge with a large value of k might preserve the performance, but the same value of k may lead to losing high variance parameters in other experts. Figure 5 shows how the average performance of our method varies with α vs how the performance of Twin merge varies with its hyperparameter, rank.

For TIES and DARE, we do a grid search on the density parameter λ. Specifically we search in $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ and report the results from the best checkpoint.

For all the algorithms, we use uniform weights on all the source models while performing a weighted average. We use the `lm-evaluation-harness`[3] (Gao et al., 2024) and `bigcode-evaluation-harness`[4] (Ben Allal et al., 2022) github public repositories to evaluate our merged models.

B.2 Results on Mistral

We use three expert models fine-tuned for the Mistral architecture. We use `Mistral-7B-Instruct-v0.1` (Jiang et al., 2023) as the instruction following expert, but train a Math and a Code expert using supervised fine-tuning on the pre-trained model, `Mistrla-7B-v0.1` with learning rate $2e-5$ and batch size 16 for one epoch.

Model	**Human-Eval**	**GSM8K**	**TruthfulQA**	**Overall**
	(acc)	(acc)	(BLEU acc)	
Pre trained	0.3048	0.3836	0	0.362
Coding Expert	0.378	0.2434	0.377	0.5394
Math Expert	0.1158	0.7225	0.3721	0.8667
Instruct Expert	0.3109	0.3412	0.4871	0.6847
TA	0.3658	0.6179	0.4602	0.8951
TIES	0.3109	0.6285	0.4835	0.9101
TA w/ DARE	0.3476	0.5193	0.4774	0.8259
TIES w/ DARE	0.3171	0.4549	0.5153	0.7967
Twin-Merge (rank = 32)				
↬ Oracle	0.335	0.629	0.507	0.9322
↬ `M-7b`	0.329	0.608	0.506	0.9137
↬ `M-8x7b`	0.359	0.61	0.506	0.9209
↬ Random	0.341	0.558	0.465	0.8463
Split-Merge (density = 0.97)				
↬ Oracle	0.354	0.668	0.503	0.9637
↬ `M-7b`	0.347	0.602	0.508	0.9137
↬ `M-8x7b`	0.335	0.619	0.51	0.9264
↬ Random	0.341	0.524	0.477	0.8281

Table 3: The performance of the **expert models** and **merged models** on the benchmarks. We **bold** the number of the highest performing model and <u>underline</u> the close second.

Split	**Human-Eval**	**GSM8K**	**TruthfulQA**	**Overall**
	(acc)	(acc)	(BLEU acc)	
w/ Coding Expert	0.3536	0.6611	0.4357	0.9092
w/ Math Expert	0.3414	0.6679	0.4321	0.9096
w/ Instruct Expert	0.3231	0.4867	0.5031	0.8141

Table 4: The performance of the **expert models** before routing.

[3]https://github.com/EleutherAI/lm-evaluation-harness/blob/main/LICENSE.md

[4]https://github.com/bigcode-project/bigcode-evaluation-harness/blob/main/LICENSE

B.3 Efficiency Ratio for Mistral Models

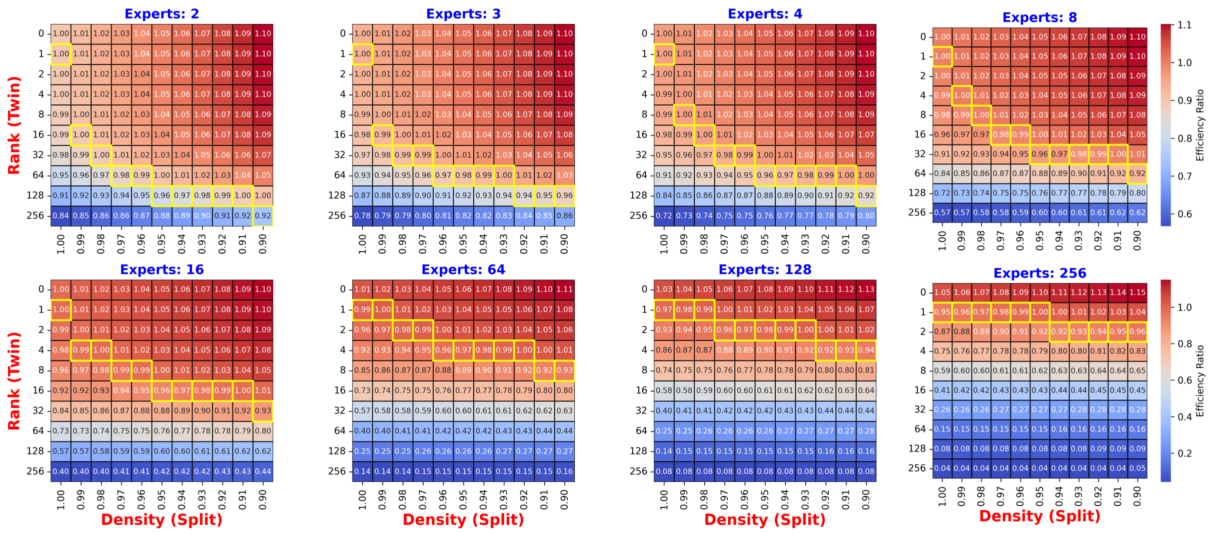

Figure 7: Efficiency ratios for the Mistral models.

B.4 Routing

Prompt for LLM-Router. In our routing prompt, we provide multiple choices of topics that a user could ask about. Additionally, as the space of topics is expansive, and our provided choices might not be exhaustive, we provide a category dubbed `OTHER` to route all user prompts that do not fall in the specified categories. Finally, we also provide in-context examples to demonstrate the format of prediction to the LLM-router. With this setup, we apply each sample in our test sets to an LLM-router and ask the router to classify amongst the provided categories.

In our Mistral experiments, we route prompts classified as `CODING` to our *coding* expert, as `MATH/REASONING QUESTION` to our *math* expert, and the remaining categories to our *instruct* expert. Finally, the prompts classified as `OTHER` is routed to the 'Split w/ Base' model.

In our Qwen experiments, we test a simpler prompt category, with lesser number of total categories for the router to classify amongst. In this setting, ambiguous classifications are expected to result in the OTHER category.

Routing Prompt Example for Mistral Experts

You are given a prompt string asked by a user. Your job is to classify the topic of this prompt amongst the following categories:

- A) CODING: a prompt that is related to coding or programming languages.
- B) MATH/REASONING QUESTION: this is a math/reasoning prompt where the user asks for a solution to a math problem or a problem that requires reasoning skills, e.g., calculating something or proving something, etc.
- C) INSTRUCTION FOLLOWING: this is a prompt where the user is asking the model to generate something by explicitly following a set of instructions, e.g., generating in some format, or following a set of instructions, etc.
- D) GENERAL QUESTION: this is a prompt where the user is asking a question to the model, e.g., asking for a definition of a word, asking for a solution to a problem, etc.
- E) CREATIVE WRITING: this is a prompt where the user is asking for creative content, such as stories, poems, or scripts.
- F) ANALYSIS: this is a prompt where the user is asking for an analysis or interpretation of text, data, or a situation.
- G) TRANSLATION: this is a prompt where the user is asking for translation between languages.
- H) ROLEPLAY: this is a prompt where the user is asking the model to assume a specific role or persona.
- I) SUMMARIZATION: this is a prompt where the user is asking for a summary of text or information.
- J) OTHER: this is a prompt that does not fit into any of the above categories

Choose only amongst the above options and give a reasoning/evidence of your choice.
If there are multiple questions in the prompt, answer for the last question.
Choose more specific choices over more general ones, e.g. if a question is about ’math’, choose ’math’ over ’question’.

PROMPT: Write a Python function to calculate the factorial of a number.
CLASSIFICATION: (A)

PROMPT: Calculate the probability of rolling a sum of 7 with two six-sided dice.
CLASSIFICATION: (B)

PROMPT: Compose a haiku about the changing seasons.
CLASSIFICATION: (E)

PROMPT: prompt
CLASSIFICATION:

Routing Prompt Example for Qwen Experts

You are given a prompt string asked by a user. Your job is to classify the topic of this prompt amongst the following categories:

- A) CODING: a prompt that is related to coding or programming languages.
- B) MATH/REASONING QUESTION: this is a math/reasoning prompt where the user asks for a solution to a math problem or a problem that requires reasoning skills, e.g., calculating something or proving something, etc.
- C) MEDICAL: this is a prompt where the user is asking for a medical question or answer.
- D) INSTRUCTION FOLLOWING: this is a prompt where the user is asking the model to generate something by explicitly following a set of instructions, e.g., generating in some format, or following a set of instructions, etc.
- E) GENERAL QUESTION: this is a prompt where the user is asking a question to the model, e.g., asking for a definition of a word, asking for a solution to a problem, etc.
- F) OTHER: this is a prompt that does not fit into any of the above categories

Choose only amongst the above options and give a reasoning/evidence of your choice.
If there are multiple questions in the prompt, answer for the last question.
Choose more specific choices over more general ones, e.g. if a question is about ‘math’, choose ‘math’ over ‘question’.
If the prompt is a code prompt or requires coding to answer, choose ‘code’.

PROMPT: Write a Python function to calculate the factorial of a number.
CLASSIFICATION: (A)

PROMPT: Calculate the probability of rolling a sum of 7 with two six-sided dice.
CLASSIFICATION: (B)

PROMPT: Compose a haiku about the changing seasons.

CLASSIFICATION: (D)

PROMPT: Compose a haiku about the changing seasons such that each line has 5 syllables.
CLASSIFICATION: (D)

PROMPT: A surgeon is performing a heart transplant. Which of the following is NOT a critical step in the procedure?
A) Connecting the donor heart’s blood vessels
B) Administering immunosuppressant drugs
C) Painting the operating room walls blue
D) Monitoring vital signs during surgery
CLASSIFICATION: (C)

PROMPT: prompt
CLASSIFICATION:

Routing Errors

In Table 5, we observe several examples of prompts that are classified under different categories, such as *code* rather than maths.

In the first example, classification to *code* stems from the problem’s structure, which involves performing

Prompt	Original Category	Predicted Category
Ivan had $10 and spent 1/5 of it on cupcakes. He then spent some money on a milkshake and had only $3 left. How much is the milkshake?	math	code
John builds a model rocket that can travel 500 ft in the air. He builds a second rocket that can travel twice as high. What is the combined height of the two rockets?	math	code

Table 5: Examples of prompts classified as *code* instead of pure math.

a series of arithmetic operations that could easily be interpreted by a computer program or algorithm, similar to how code processes numerical inputs step-by-step. Similarly, the second prompt is also classified as *code*. Like the previous example, this problem lends itself to a straightforward computational approach, breaking down the steps needed to calculate the combined height of the rockets, a task that could be readily implemented in code.

These examples illustrate how some prompts lend themselves more to computational, step-by-step reasoning and get classified as prompts that should be routed to a coding expert.

B.5 Qwen Routing

In Table 6, we also show the results of each expert model constructed by the algorithm (before routing), called `Split-Merge w/ [Expert]`. For example, if we always use the coefficient corresponding to the Coding expert and say that the Coding domain is index by 1 among the 4 domains (Math, IF, Coding, MedQA), we get the model `Split w/ Coding`, that is, $\mu + c_1\eta$.

Split	GSM8K	IFEval	MedQA	H-Eval	Overall
w/ Code	0.6657	0.3789	0.4949	0.4817	0.9790
w/ Math	0.6755	0.3777	0.4611	0.4634	0.9569
w/ IF	0.6641	0.3777	0.4902	0.4878	0.9748
w/ Med	0.6717	0.3765	0.4902	0.4634	0.9763

Table 6: Results of the Qwen **experts** created by Split-Merge before routing. Notably, after routing, the Weighted Normalized Score increases with any of the routers we have experimented with (see Split-Merge rows in Table 1).

B.6 Mistral Routing

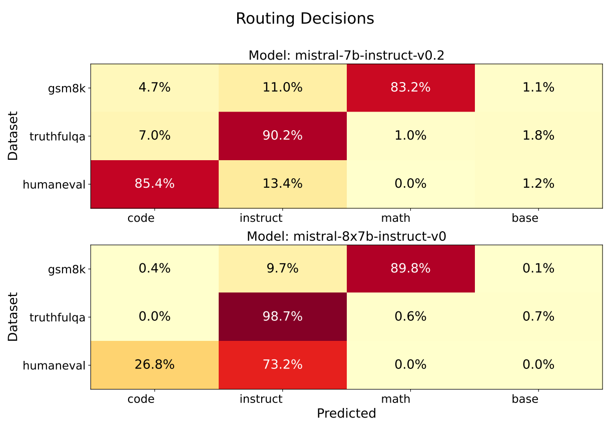

Figure 8: Routing prediction over three test sets: GSM8k, TruthfulQA and Coding HumanEval with `M-7b` and `M-8x7b`.

B.7 Generalization to Unseen Domains

Table 7 reports MT-Bench results of some of the models to understand generalization to unseen domains. The results show that Twin/Split Expert components mostly retain the generalization abilities of the fine-tuned models.

Method	**Code Exp**	**Math Exp**	**IF Exp**	**Med Exp**
Fine-Tuned	5.8266	6.1258	7.0609	6.2344
Twin ($r = 16$)	6.0219	6.1625	6.8438	6.7406
Split ($\alpha = 0.95$)	6.6250	6.6969	6.4531	6.5594

Table 7: **MT-Bench** numbers of the (a) Fine-Tuned expert models and the (b) experts created by Split-Merge, and (c) experts created by Twin-Merge. Note that the Qwen Pre-trained model only has MT-Bench score of 1.6840.