A BROADER LOOK AT MODEL MERGING: RETHINKING IMPLICIT REGULARIZATION INDUCED BY TASK ARITHMETIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Model merging aims to build a multi-task model cheaply by combining the weights of individual task-specific models. To perform well across multiple tasks, most existing merging methods use an additional dataset to find the coefficients for the best linear combination of task-specific weight updates. However, we identify an implicit regularization in this standard practice: searching over coefficients restricts the candidate models to a subspace spanned by task-specific weight updates. In this work, we investigate whether this regularization is actually useful. Surprisingly, empirical results show that optimizing merged-model weights without this regularization significantly boosts the performance of common merging methods across multiple architectures, domains, and even in an extremely data-limited scenario where only one instance is available per class. Moreover, directly optimizing the pretrained model weights even outperforms some existing merging methods. Analysis shows that better multi-task weights exist outside the subspace and can be found using multiple methods. We study different strategies for using the additional dataset, discussing their practical use and implications for model merging. Overall, this work calls for revisiting the existing model-merging pipeline, motivating a broader exploration of the weight space and a reconsideration of the implicit regularization induced by task arithmetic.

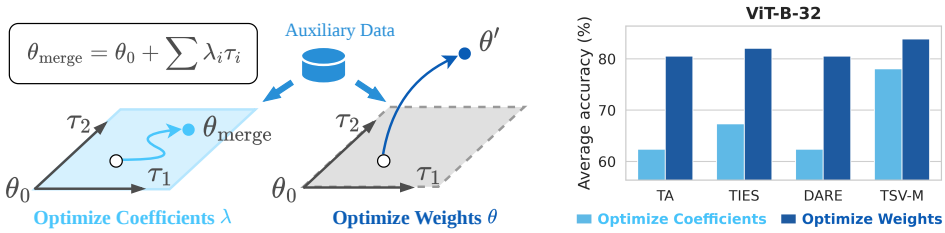


Figure 1: **A new perspective on the model-merging pipeline.** The existing model-merging pipeline builds the merged model θ_{merge} by adding a linear combination of task-specific weight updates τ_i to the pretrained model θ_0. Standard practice uses an additional dataset, which we term *auxiliary data*, to optimize the coefficients λ for multi-task performance. This coefficient optimization regularizes model updates to the subspace spanned by τ_i. **We challenge this pipeline by using the same auxiliary data to optimize the merged-model weights**, allowing optimization beyond this subspace. Experimental results show that exploring the broader weight space significantly improves the performance of merging methods (TA, TIES, DARE, and TSV-M).

1 INTRODUCTION

Model merging aims to build a unified model by combining the weights of individual task-specific models without retraining on full training data. It enables a wide range of applications, including multi-task model construction (Ilharco et al., 2023; Yadav et al., 2023), cross-lingual capabil-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ity transfer (Huang et al., 2024; Bandarkar et al., 2025), and multi-objective preference alignment (Rame et al., 2023; Jang et al., 2024; Liang et al., 2026).

Task Arithmetic (Ilharco et al., 2023) is a widely adopted framework for model merging, which builds a multi-task model by linearly combining task vectors. A task vector is defined as $\tau_i = \theta_i - \theta_0$, where θ_i is a model fine-tuned on task i from the pre-trained model θ_0. Such linear combinations have been shown to produce effective multi-task models (Ilharco et al., 2023; Ortiz-Jimenez et al., 2023). Recent studies follow this formulation and focus on refining the task vectors to reduce interference among tasks (Yadav et al., 2023; Yu et al., 2024; Gargiulo et al., 2025). However, the merged model can be sensitive to the coefficients of the linear combination (Yang et al., 2024b; Tam et al., 2026), so most merging methods rely on auxiliary data to select coefficients (Chaves et al., 2026), either through grid search on hold-out data (Ilharco et al., 2023; Yadav et al., 2023; Yu et al., 2024) or through test-time adaptation on unlabeled test data (Yang et al., 2024b; Touayouch et al., 2026). Despite these advances, a performance gap remains between merged models and individual task-specific models (Yang et al., 2026).

To mitigate this gap, we revisit the current model merging pipeline and ask whether the existing framework fully uses the auxiliary data. Optimizing the coefficients of the linear combination is equivalent to restricting weight optimization to the subspace spanned by the task vectors. We refer to this subspace as the *task-vector subspace*. From this perspective, we identify coefficient optimization under task arithmetic as a constrained optimization that implicitly regularizes candidate models to the task-vector subspace. While the empirical success of task arithmetic shows that effective multi-task weights exist within the task-vector subspace, its implicit regularization has not been examined. Thus, the research question is: *For model merging, is the implicit regularization induced by task arithmetic actually useful, or does it exclude better multi-task weights?*

To find out the answer, we investigate direct optimization in the weight space beyond the task-vector subspace. We apply the most common optimization method, gradient descent, in standard model merging benchmarks across computer vision and natural language processing. Surprisingly, we find that optimization in the weight space consistently and significantly improves the performance of various model-merging methods beyond optimizing the coefficients, as shown in Figure 1. We further downsample the auxiliary dataset and find that the conclusion still holds when data are scarce, with only one sample per class. The results confirm that the optimization flexibility does not lead to overfitting in model-merging benchmarks. Moreover, even gradient descent direct from the pretrained model can outperform some existing merging methods. However, merged methods still provide stronger initializations for gradient descent. These findings imply that the task arithmetic's implicit regularization may lead to the underuse of auxiliary data and limit merged models' performance.

We further analyze geometric properties of gradient descent on the merged models. We first find that gradient descent consistently leaves the task-vector subspace, suggesting that better multi-task weights exist outside this restricted region. Motivated by this finding and Gan & Isola (2026), we proposed *S-RandOpt* that samples weights outside the subspace. By allowing just one additional degree of freedom, *S-RandOpt* outperforms coefficient optimization, further supporting the idea that the implicit regularization of task arithmetic overly restricts model complexity. It also implies that finding better multi-task weights does not rely on gradient descent. We also investigate the memory and computational costs of different strategies for using auxiliary data. Finally, we study different strategies for using the auxiliary data under various conditions, revealing the bias-variance tradeoff as a key principle for determining how auxiliary data should be used. Our analysis motivates future merging methods to provide strong initializations that enable more effective use of auxiliary data and, more broadly, calls for rethinking how to build a strong multi-task model via model merging.

2 Overview of model merging methods

The goal of model merging is to build a multi-task model by leveraging models that are separately fine-tuned on different tasks. In this work, we focus on merging models that share the same architecture and are fine-tuned from the same pretrained model on different tasks. While multi-task learning (Ruder, 2017) and multi-task fine-tuning (Chung et al., 2024) also aim to build multi-task models, model merging focuses on the specific scenario where task-specific models are available. This work centers around the model merging scenario, but applies the multi-task learning algorithm.

Table 1: **Summary of prior works in terms of compute and data usage. Backprop.** refers to backpropagation. **Data Source** refers to the source of auxiliary data. Existing works leverages backpropagation and auxiliary data to build merged models.

	Backprop.	Data Source	Auxiliary Data	
			Inputs	Labels
TA (Ilharco et al., 2023)	×	Valid	✓	✓
Fisher averaging (Matena & Raffel, 2022)	✓	Train	✓	×
TIES (Yadav et al., 2023)	×	Valid	✓	✓
Uncertainty-based (Daheim et al., 2024)	✓	Train	✓	✓
DARE (Yu et al., 2024)	×	Valid	✓	✓
TSV-M (Gargiulo et al., 2025)	×	×	×	×
AdaMerging (Yang et al., 2024b)	✓	Test	✓	×
DivMerge (Touayouch et al., 2026)	✓	Test	✓	×

Most model merging methods are based on *task vectors* (Ilharco et al., 2023), defined as $\tau_i = \theta_{i,ft} - \theta_0$, where $\theta_{i,ft}$ is the model weight fine-tuned on i-th task and θ_0 is the pretrained model they share. Given T tasks, the merged model is then formed as

$$\theta_{\text{merged}} = \theta_0 + \sum_{i=1}^{T} \lambda_i \, \tilde{\tau}_i. \tag{1}$$

Prior works build better merged models by transforming task vectors into variants $\tilde{\tau}_i$, which we call *advanced task vectors*. One branch of existing methods constructs advanced task vectors by leveraging information from the original task vectors, including sparsification (Yadav et al., 2023; Yu et al., 2024) and low-rank decomposition (Gargiulo et al., 2025; Marczak et al., 2025). Another branch of methods requires data, including calculating gradients to estimate curvature information (Matena & Raffel, 2022; Daheim et al., 2024), or estimating activations to reduce the conflict between tasks (Yang et al., 2024a). We further discuss more existing merging methods in Appendix B.

After building the (advanced) task vectors, the next step is to determine the coefficients $\{\lambda_i\}$ of each task vector. As shown in the existing works (Yang et al., 2024b; Tam et al., 2026), merging methods' performance is highly sensitive to the coefficients. The simplest baseline to obtain good coefficients is grid search (Ilharco et al., 2023), which uses a hold-out data set to pick the best set of coefficients. However, when merging many tasks, grid search in the T-dimensional space requires extensive inference, making it impractical. Therefore, most existing works instead perform diagonal search, using a single shared coefficient for all task vectors. Another line of work develops methods to optimize the coefficients using this hold-out dataset, including gradient descent on coefficients (Yang et al., 2024b; Touayouch et al., 2026) and Bayesian optimization (Lee et al., 2025). Notably, optimizing the coefficients is equivalent to optimizing the model within the subspace spanned by (advanced) task vectors. In this work, we ask if this is the best way to leverage this hold-out dataset for building multi-task models.

2.1 Compute and data usage of existing model merging methods

As the goal of model merging is to build a multi-task model without retraining on the whole training data, the computational cost and data usage are practical considerations. We collectively refer to additional data used for different purposes as *auxiliary data*, including the hold-out (valid) dataset and unlabeled test set used for coefficient optimization, as well as the data used to build advanced task vectors. Adapted from Gargiulo et al. (2025), Table 1 summarizes the existing model merging methods. Prior work has separately used backpropagation on model weights and auxiliary data to optimize merging coefficients. We therefore investigate using auxiliary data to directly optimize model weights with gradient descent. In Appendix B, we provide more details of the existing method's compute and data usage.

3 DOES OPTIMIZING WITHIN TASK-VECTOR SUBSPACE SUFFICE?

In this section, we compare optimization within the task-vector subspace and the broader weight space. Under standard model-merging benchmarks in vision and language, we evaluate the performance of merged models from common model-merging methods under two optimization choices.

3.1 EXPERIMENTAL SETUP

Data. Following the original setup (Ilharco et al., 2023), we split 10% of the training set as the auxiliary data, and leave the rest as the training data for fine-tuning the task-specific model. We evaluate 9 vision tasks: MNIST (LeCun, 1998), GTSRB (Stallkamp et al., 2011), FER2013 (Goodfellow et al., 2013), Stanford Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), Food101 (Bossard et al., 2014), SUN397 (Xiao et al., 2016), RESISC45 (Cheng et al., 2017), and EuroSAT (Helber et al., 2019). For language, we evaluate 4 tasks: DDXPlus (Tchango et al., 2022), Banking77 (Loukas et al., 2023), IFEval (Zhou et al., 2023), and Usefulness Judge (Chen et al., 2026).

Models. We evaluate 3 CLIP-ViT variants (Radford et al., 2021) for vision, Qwen3-0.6B, 1.7B and 4B (Yang et al., 2025) and Llama-3.2-1B (Grattafiori et al., 2024) for language. Training details of task-specific models are provided in Appendix A.1.

Model merging methods. We test 4 common model merging methods: vanilla Task Arithmetic, which we refer to as TA (Ilharco et al., 2023), TIES (Yadav et al., 2023), DARE (Yu et al., 2024) and TSV-M (Gargiulo et al., 2025). Section 2 provides a detailed introduction to these methods.

3.1.1 TWO WAYS FOR OPTIMIZING MERGED MODELS

Coefficient search and subspace best. Existing methods search for good coefficients $\{\lambda_i\}$ in the final step. In this work, we denote *Naive* as setting all coefficients λ_i to $\frac{1}{T}$ or 1, and *Naive baseline* as the best between the two. *Coefficient search* refers specifically to diagonal coefficient search in this work. To estimate the empirical best performance achievable within the task-vector subspace using auxiliary data, we report the best result among three coefficient optimization methods (denoted as *subspace best*): gradient descent on coefficients, fine-grained grid search, and Bayesian optimization. See Appendix Section C.1 for details. We simply refer to the task-vector subspace as the *subspace* in this section, where optimizing the coefficients is equivalent to optimizing the model weights within this subspace. Therefore, we call gradient descent on the coefficients *Subspace GD*.

Optimizing model weights. We minimize the multi-task loss with gradient descent (denoted as *GD*), initialized from the *Naive baseline*. Specifically, we fine-tune the model using auxiliary data from each task. For the main results, we use AdamW (Loshchilov & Hutter, 2019), and discuss hyperparameters in Appendix A. We also evaluate SGD in Appendix A.5. We perform full-weight fine-tuning for vision models and LoRA fine-tuning (Hu et al., 2022) for LLMs.

3.2 RESULTS AND DISCUSSIONS

All methods benefit from optimizing coefficients. Figure 2 shows that optimizing the coefficients of all merged methods improves their performance. Meanwhile, optimizing the coefficient beyond the diagonal search can further boost performance, which is consistent with existing work (Yang et al., 2024b; Touayouch et al., 2026). Notably, coefficient optimization improves every method, including TSV-M, which is relatively insensitive to coefficient choices (Gargiulo et al., 2025).

Optimizing weights is better than optimizing coefficients. Figure 2 shows that gradient descent (GD) on weights consistently outperforms the subspace's best performance and improves all merged models. While TSV-M does not require auxiliary data for coefficient tuning (Gargiulo et al., 2025), GD still improves its performance when such data are available. We further discuss the correlation between each merging method's naive performance and its performance after GD in Section 5.2.

GD from the pretrained model is surprisingly good. For vision tasks (Figure 2 first row), GD from the pretrained model (GD from pretrained) is even better than TA, TIES, and DARE with optimized coefficients. These results support our claim that the use and importance of auxiliary data have been underestimated. Nonetheless, GD from base is still worse than GD from the merged model, which demonstrates the effectiveness of merging methods.

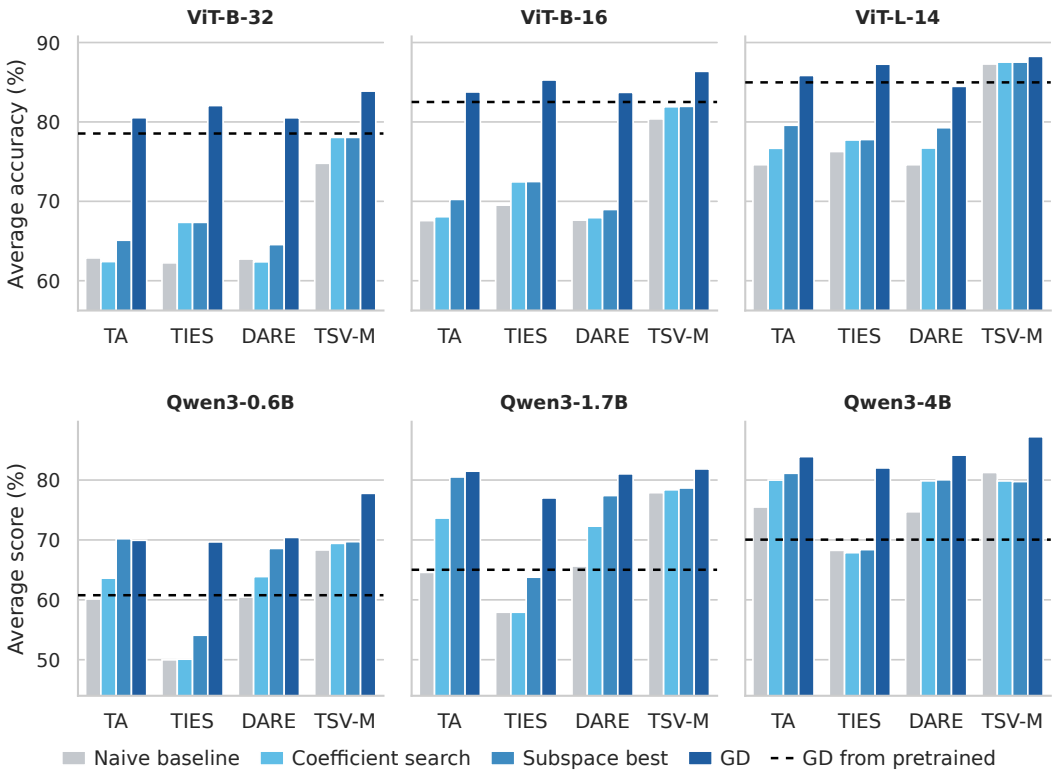


Figure 2: **Average test accuracy under different uses of auxiliary data.** Across vision tasks (top) and language tasks (bottom) with different models, Gradient Descent (GD) from the merged models consistently outperforms both coefficient search and the empirical best performance of its search space (Subspace best) for every merging method (TA, TIES, DARE, and TSV-M).

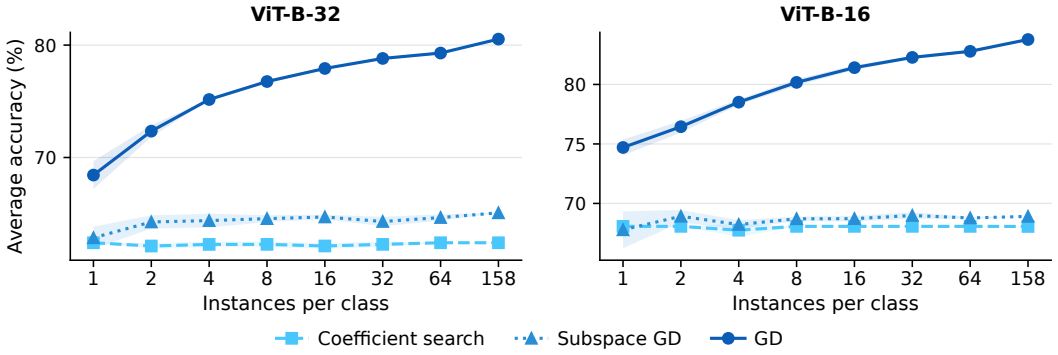


Figure 3: **Average test accuracy under different sizes and uses of auxiliary data for TA.** We downsample the auxiliary data and compare different uses of auxiliary data for TA. We run three random seeds for downsampling, and report error bars as the shaded regions. Gradient Descent (GD) consistently outperforms the other two approaches that optimize weights within the subspace.

Conclusions hold with limited data. We downsample the auxiliary dataset and report vision results for TA in Figure 3; TIES results are in Appendix C.3. Weight-space gradient descent still outperforms optimization within the task-vector subspace. This implies that in standard model merging benchmarks, optimizing flexibility does not lead to worse performance even when the data are scarce. Another observation is that weight-space gradient descent better leverages auxiliary data: its performance continues to improve as more data become available, whereas coefficient optimization quickly saturates. We further challenge the limits of gradient descent in Section 5.3.

4 LEAVING THE TASK-VECTOR SUBSPACE IMPROVES PERFORMANCE

Section 3 shows that directly optimizing the model weights with gradient descent consistently outperforms coefficient optimization within the task-vector subspace. In this section, we investigate the source of this improvement. In Section 4.1, we first examine the geometric relationship between the solutions found by gradient descent and the task-vector subspace. In Section 4.2, we propose another method showing that this improvement does not depend on gradient descent.

4.1 ANALYSIS OF GRADIENT DESCENT FOR MODEL MERGING

To understand whether the performance gain requires leaving the task-vector subspace, we first visualize gradient descent in a 2-task setting. We run gradient descent from several merged models θ_{merge} on this plane, and denote the resulting update by $\Delta = \theta_{\text{GD}} - \theta_{\text{merge}}$. For each run, we project the trajectory onto three axes centered at θ_0: two axes spanning the subspace, τ_1 and $\tau_2^{\perp}$ (the component of τ_2 orthogonal to τ_1), and a third axis $\Delta_{\perp}$, the component of Δ orthogonal to the subspace. As shown in Figure 4, gradient descent consistently leaves the subspace regardless of the initialization.

We next quantify this behavior in the 9-task setting used in Section 3. For each merging method, we run gradient descent from its merged model and decompose the update into $\Delta_{\perp}$ and the in-subspace component $\Delta_{\parallel}$. Table 2 shows that $\Delta_{\perp}$ dominates the update. Moreover, the in-subspace component alone does not improve performance: adding only $\Delta_{\parallel}$ to the merged model barely changes the accuracy, whereas the full update improves it by up to 20 points. Together with Figure 4, these results show that the performance gain of gradient descent requires leaving the task-vector subspace.

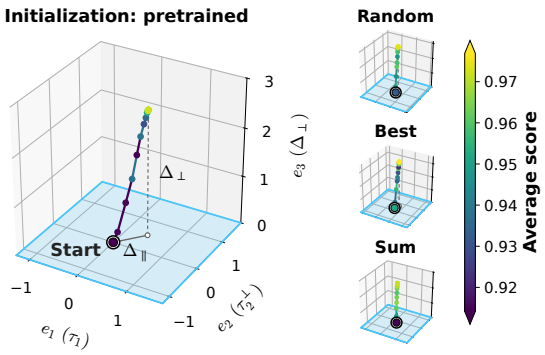


Figure 4: **Gradient descent trajectories in the 2-task setting (ViT-B-32, EuroSAT and GTSRB).** Each panel shows one run initialized on the task-vector subspace (blue plane). The axes e_1, e_2, and e_3 are the directions of τ_1, $\tau_2^{\perp}$, and each run's update's orthogonal component to the subspace $\Delta_{\perp}$. In all runs, gradient descent leaves the subspace as the score (accuracy) increases.

Table 2: **Decomposition of the gradient descent update (ViT-B-32, 9 tasks).** We decompose the fine-tuning update $\Delta = \theta_{\text{GD}} - \theta_{\text{merge}}$ into its projection onto the task-vector subspace, $\Delta_{\parallel}$, and the orthogonal component, $\Delta_{\perp} = \Delta - \Delta_{\perp}$. Adding only $\Delta_{\parallel}$ to the merged model yields no accuracy gain (Acc. Gain), whereas the full update improves accuracy by 9 to 20 points. The empirical results indicate that the performance improvement requires leaving the task-vector subspace.

Method	$\|\Delta_{\perp}\|/\|\Delta_{\parallel}\|$	Acc. Gain (%)	
		$+\Delta_{\parallel}$	$+\Delta_{\parallel} + \Delta_{\perp}$
TA	11.14	-0.04	17.66
TIES	21.60	0.07	19.81
DARE	14.50	-0.01	17.78
TSV-M	78922.16	0.00	9.09

4.2 S-RANDOPT: A SIMPLE METHOD TO BOOST MODEL MERGING

Gradient descent trajectory analysis shows that better multi-task weights are outside the task-vector subspace. Next, we ask: *Is it easy to find these weights?* Gan & Isola (2026) shows that task-specific model weights are dense around the pretrained model weights, and random sampling can find effective model weights. Specifically, they propose *RandOpt* that randomly samples weights and selects the top-performing weights using training data. Inspired by these findings, we ask whether sampling outside the task-vector subspace can find multi-task weights that outperform those within the subspace. We propose *Subspace-RandOpt (S-RandOpt)*, which uses the auxiliary data to compute the gradient from the task-vector subspace, extends the task-vector subspace by this direction, and then randomly samples within the expanded region. Finally, it uses auxiliary data for evaluation to select the best sampled weight. Appendix C.5 discusses the implementation details.

Table 3: **S-RandOpt results across vision and language models.** For two merging methods (TA and DARE), we report the average performance of top-5 sampled weights from S-RandOpt. It frequently outperforms the best performance on the task-vector subspace.

Method	Strategy	Vision		Language	
		ViT-B-32	ViT-B-16	Qwen3-0.6B	Qwen3-1.7B
TA	Subspace best	65.07	68.90	70.20	80.51
	S-RandOpt	68.76 ± 0.32	70.73 ± 0.43	70.53 ± 0.55	80.22 ± 1.10
DARE	Subspace best	64.55	68.97	68.58	77.41
	S-RandOpt	68.66 ± 0.32	70.70 ± 0.34	69.60 ± 0.58	79.53 ± 0.52

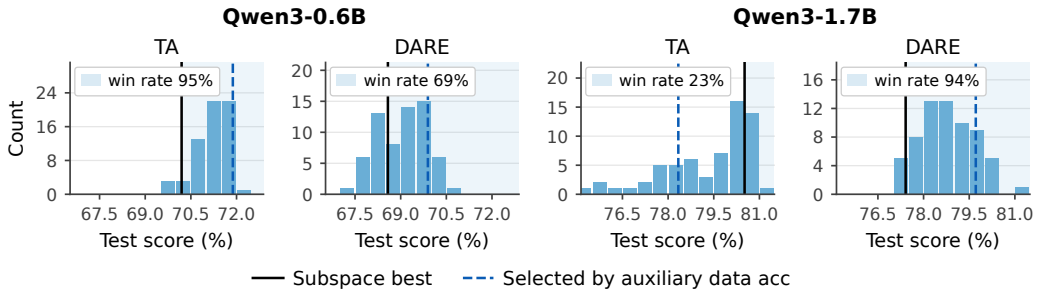


Figure 5: **S-RandOpt sampled weights' test performance.** We sample 64 random weights in language models using S-RandOpt, and compare their test performance with the best point on the task-vector subspace (subspace best). A high proportion of sampled weights from outside the subspace outperforms the best weight on the subspace (Subspace best).

As reported in Table 3, S-RandOpt frequently outperforms the best weights in the task-vector subspace. Meanwhile, Figure 5 shows that better multi-task weights are dense outside the task-vector subspace. S-RandOpt not only improves the merged models, but also demonstrates that finding better multi-task weights does not depend on gradient descent.

By adding just one additional degree of freedom beyond the task-vector subspace, even random sampling can outperform the best performance in the subspace. These results further confirm that coefficient optimization of task arithmetic might imposes unnecessarily strong regularization in model merging benchmarks. As a proof-of-concept, S-RandOpt also motivates future work on developing model-merging algorithms that make better use of auxiliary data by leaving the task-vector subspace.

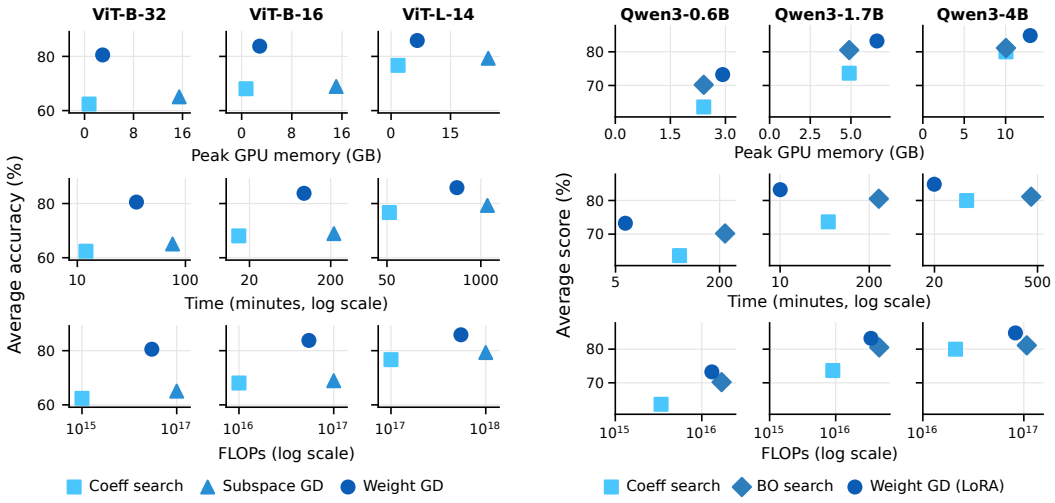
5 Analysis for Practical Use

In this section, we further analyze different strategies for using the auxiliary data, and discuss the practical implications. Specifically, we examine the computational costs of different strategies in Section 5.1, the role of model merging methods in Section 5.2, the trade-off of the regularization in Section 5.3, and test-time adaptation of the merged model in Section 5.4.

5.1 Computational Usage

As the goal of model merging is to build a multi-task model efficiently, "training" seems to be an action with high cost. To fairly investigate the cost and provide practical recommendations, we analyze different strategies of using auxiliary data from three perspectives: memory, time, and compute. The results are shown in Figure 6.

Memory. Gradient descent induces larger peak GPU memory usage as it requires backpropagation, while coefficient search and Bayesian optimization only require inference. However, extra memory usage is not huge compared to optimizing coefficients using gradient descent (Subspace GD), where existing work (Yang et al., 2024b; Touayouch et al., 2026) leverages this technique.



(a) Vision (b) Language

Figure 6: **Computational usage of different ways to use the auxiliary data.** Gradient descent on weights (Weight GD) uses slightly more memory than coefficient search. While doing gradient descent requires more compute effort (time/FLOPs), it trades compute for better performance.

Time and Compute. Gradient descent trades compute for better performance. For vision models, both time and FLOPs follow the same pattern: Subspace GD > Weight GD > Coefficient search. For language models, gradient descent is more time-efficient because coefficient search and Bayesian optimization require repeated inference rollouts with autoregressive decoding.

In summary, gradient descent requires slightly more GPU memory in the standard implementation [1], and trades compute for better performance. We tested SGD in Appendix A.5 to reduce the GPU memory usage. We recommend that practitioners select the strategy and the compute effect to achieve the Pareto frontier in each scenario.

5.2 Merging methods provide strong initialization

While Section 3 suggests that gradient descent is the better way to use auxiliary data, this finding does not dismiss the importance of model merging methods. Figure 7 shows the results of ViT-L-14, and Appendix Section D.1 provides more results of other vision and language models. The empirical results show that merging methods with higher initial accuracy also tend to achieve better performance after Gradient Descent (GD). Meanwhile, merging methods provide necessary initialization points for gradient descent, since random initialization yields weak performance. Moreover, this clearly distinguishes this work's contribution from simple multi-task learning or fine-tuning, as starting from merged models performs significantly better than starting from the pretrained model or random initializations. This result motivates future work on stronger model merging methods to provide stronger initialization for gradient descent and other strategies for using auxiliary data.

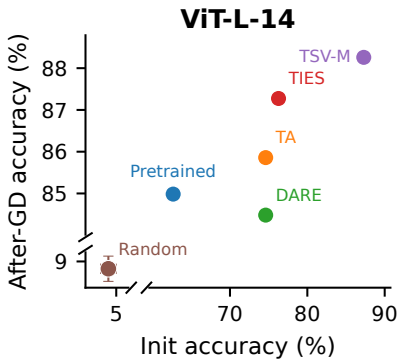


Figure 7: **Positive correlation between Init accuracy of merged models and their after-GD accuracy.** Existing model merging methods provide strong initializations for gradient descent, and stronger methods have better after-GD performance.

[1] One can always move some computation to CPU to reduce GPU memory usage.

5.3 Extreme cases and underlying concepts

To understand the big picture, we analyze when gradient descent for model merging fails. Figure 8 shows that in the 2-task setting, where the auxiliary data size is smaller than in the 9-task setting in Section 3, GD can perform worse than coefficient search when data are downsampled. As more data are provided, GD eventually outperforms coefficient search. Adding regularization for GD makes low-data scenarios better. However, it slightly decreases performance when data are sufficient. More analysis is in Appendix D.2.

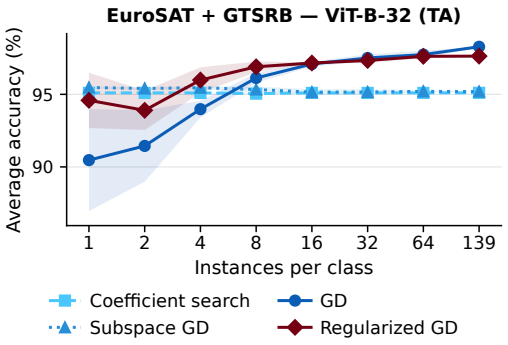


Figure 8: **Auxiliary data scaling for TA in 2-task setting**. The number of classes (53) is smaller than in the 9-task setting (854). Gradient descent's performance is worse than coefficient search when data is extremely limited, but still outperforms it when more data are provided. Adding regularization for GD mitigates the issue. The results indicate that generalization is a key factor when using auxiliary data.

Together with results of coefficient search versus weight-space fine-tuning (Section 3), S-RandOpt (Section 4.2), and full fine-tuning versus LoRA fine-tuning (Appendix C.2), these results reflect a machine learning principle: **bias-variance tradeoff**. With less data available, stronger regularization (higher bias) prevents overfitting; with more data available, variance decreases, so greater model flexibility has less risk of overfitting. Coefficient optimization allows limited degrees of freedom and thus imposes stronger regularization, whereas weight-space fine-tuning allows greater flexibility. On standard model merging benchmarks, we have shown that gradient descent does not severely overfit. We recommend practitioners consider the generalization trade-off when deciding how to use auxiliary data.

5.4 Test-Time adaptation

One way to address the generalization issue is test-time adaptation (Wang et al., 2021; Liang et al., 2025). Some previous model merging works (Yang et al., 2024b; Touayouch et al., 2026) have leveraged unlabeled test data to adapt the merged models on test data. However, they only optimize the coefficients. For TA (Ilharco et al., 2023), Table 4 shows that using gradient descent to adapt the unlabeled test data outperforms coefficient optimization. Detailed settings and more results are in Appendix D.3, further supporting the bias-variance tradeoff as a key principle for using auxiliary data.

Table 4: **Test-time adaptation methods' performance.** All methods are applied on top of TA. Compared with existing methods that use an unlabeled test set to optimize the coefficients of task vectors, using this auxiliary data to perform Gradient Descent (GD) on weights leads to better accuracy across all models.

Method	ViT-B-32	ViT-B-16	ViT-L-14
AdaMerging	62.21	66.80	76.63
DivMerge	65.43	69.62	78.49
GD	**82.43**	**85.51**	**88.15**

6 Conclusion

This work identifies that optimizing weights in the task-vector subspace is a potentially overlooked bottleneck in model merging. Through extensive experiments, we call for a broader look at model merging: reconsider whether the implicit regularization of task arithmetic is useful, and the need to explore beyond the task-vector subspace. Across standard model merging benchmarks, we find that this regularization often limits the merged model's performance. By relaxing the regularization, simple gradient descent or even sampling can improve the merged model's performance. We reveal that generalization is a key factor to consider when determining how strongly the regularization should be applied. That said, we did not identify a one-size-fits-all method for model merging, as the appropriate method depends on the models, auxiliary data, and scenarios. Overall, our findings motivate future work on developing stronger merging methods and model-merging pipelines that jointly consider auxiliary-data usage and optimization flexibility.

AI USE STATEMENT

In this work, we used generative AI tools for generating experiment code (designing or providing feedback on experiments, implementing methods). We have not used generative AI tools to develop the main idea, the conceptual framework, the literature survey, support qualitative and thematic data analysis, design or provide feedback on research methodology, and interpret results. Helping develop theoretical models, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, generating synthetic data sets, proposing or refining hypotheses, assisting with translation, and cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools to make the sentences more fluent and for grammar checking. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by 2 authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Experiment details are clearly described in the main paper and Appendix A to reproduce the empirical results. We have also provided the source code in the Supplementary Material of the submission form. The datasets used are all publicly available.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Lucas Bandarkar, Benjamin Muller, Pritish Yuvraj, Rui Hou, Nayan Singhal, Hongjiang Lv, and Bing Liu. Layer swapping for zero-shot cross-lingual transfer in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, 2014.

Levy Chaves, Eduardo Valle, and Sandra Avila. Weight weaving: Parameter pooling for data-free model merging. In *Proceedings of UniReps: the Third Edition of the Workshop on Unifying Representations in Neural Models*, 2026.

Yen-Shan Chen, Zhi Rui Tam, Cheng-Kuang Wu, and Yun-Nung Chen. Expected harm: Rethinking safety evaluation of (mis) aligned llms. *arXiv preprint arXiv:2602.01600*, 2026.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 2017.

Runxi Cheng, Feng Xiong, Yongxian Wei, Wanyun Zhu, and Chun Yuan. Whoever started the interference should end it: Guiding data-free model merging via task vectors. In *Forty-second International Conference on Machine Learning*, 2025.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of machine learning research*, 2024.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. In *Forty-third International Conference on Machine Learning*, 2026.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Ian J Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, et al. Challenges in representation learning: A report on three machine learning contests. In *International conference on neural information processing*, 2013.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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*, 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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Shih-Cheng Huang, Pin-Zu Li, Yu-Chi Hsu, Kuang-Ming Chen, Yu Tung Lin, Shih-Kai Hsiao, Richard Tsai, and Hung-yi Lee. Chat vector: A simple approach to equip llms with instruction following and model alignment in new languages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10943–10959, 2024.

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

Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. Personalized soups: Personalized large language model alignment via post-hoc parameter merging. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=EMrnoPRvxe.

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*, 2013.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Chanhyuk Lee, Jiho Choi, Chanryeol Lee, Donggyun Kim, and Seunghoon Hong. Adarank: Adaptive rank pruning for enhanced model merging. In *The Fourteenth International Conference on Learning Representations*, 2026.

Sanwoo Lee, Jiahao Liu, Qifan Wang, Jingang Wang, Xunliang Cai, and Yunfang Wu. Dynamic fisher-weighted model merging via bayesian optimization. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025.

Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 2025.

Ren-Wei Liang, Chin Ting Hsu, Chan-Hung Yu, Saransh Agrawal, Shih-Cheng Huang, Chieh-Yen Lin, Shang-Tse Chen, Kuan-Hao Huang, and Shao-Hua Sun. Adaptive helpfulness–harmlessness alignment with preference vectors. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Lefteris Loukas, Ilias Stogiannidis, Odysseas Diamantopoulos, Prodromos Malakasiotis, and Stavros Vassos. Making llms worth every penny: Resource-limited text classification in banking. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Daniel Marczak, Simone Magistri, Sebastian Cygert, Bartłomiej Twardowski, Andrew D. Bagdanov, and Joost van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In *Forty-second International Conference on Machine Learning*, 2025.

Michael S Matena and Colin Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 2022.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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*, 2021.

Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36:71095–71134, 2023.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, 2011.

Wenju Sun, Qingyong Li, Wen Wang, Yang Liu, Yangliao Geng, and Boyang Li. Towards minimizing feature drift in model merging: Layer-wise task vector fusion for adaptive knowledge integration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Derek Tam, Yash Kant, Brian Lester, Igor Gilitschenski, and Colin Raffel. Realistic evaluation of model merging for compositional generalization. *Transactions on Machine Learning Research*, 2026.

Anke Tang, Li Shen, Yong Luo, Nan Yin, Lefei Zhang, and Dacheng Tao. Merging multi-task models via weight-ensembling mixture of experts. In *Forty-first International Conference on Machine Learning*, 2024.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. DDXPlus: A new dataset for automatic medical diagnosis. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Brahim Touayouch, Loïc Fosse, Géraldine Damnati, and Gwénolé Lecorvé. Divmerge: A divergence-based model merging method for multi-tasking. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7157–7180, 2026.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 2016.

Jing Xu, Jiazheng Li, and Jingzhao Zhang. Scalable model merging with progressive layer-wise distillation. In *Forty-second International Conference on Machine Learning*, 2025.

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*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Enneng Yang, Li Shen, Zhenyi Wang, Guibing Guo, Xiaojun Chen, Xingwei Wang, and Dacheng Tao. Representation surgery for multi-task model merging. In *Forty-first International Conference on Machine Learning*, 2024a.

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

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications, and opportunities. *ACM Computing Surveys*, 2026.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

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

APPENDIX

Table of Contents

A Training details **14**
- A.1 Training individual fine-tuned models . . . 14
- A.2 Data details and split . . . 15
- A.3 Subspace GD learning rate sweep and initialization point . . . 16
- A.4 Weight GD learning rate sweep and initialization point . . . 16
- A.5 Different optimizers . . . 17
- A.6 Hyperparameters of strategies . . . 18

B More related works **20**
- B.1 Variants of task-wise merging . . . 20
- B.2 Detailed discussion regarding compute and data usage . . . 20

C Main results details **21**
- C.1 Finding Subspace Best Performance . . . 21
- C.2 Full-weight vs. LoRA fine-tuning and the effect of initialization points . . . 22
- C.3 More data scaling results . . . 23
- C.4 Main results in Tables . . . 23
- C.5 S-RandOpt implementation details . . . 24

D More analyses **25**
- D.1 Complete analysis of initializations for gradient descent . . . 25
- D.2 2-task setting . . . 26
- D.3 Test-time adaptation full results . . . 27
- D.4 LLM performance on control tasks . . . 28

A TRAINING DETAILS

A.1 TRAINING INDIVIDUAL FINE-TUNED MODELS

A.1.1 VISION MODELS

We fine-tune CLIP (Radford et al., 2021) image encoders (ViT-B-32, ViT-B-16, ViT-L-14) separately on each of nine image-classification tasks: DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), FER2013 (Goodfellow et al., 2013), Food101 (Bossard et al., 2014), GTSRB (Stallkamp et al., 2011), MNIST (LeCun, 1998), RESISC45 (Cheng et al., 2017), Stanford Cars (Krause et al., 2013), and SUN397 (Xiao et al., 2016). Following Ilharco et al. (2023), the classification head of each task is the CLIP zero-shot head: we encode every class name with the task's prompt templates using the CLIP text encoder, and the (logit-scaled) normalized text embeddings serve as the weights of a linear layer with zero bias. Both the text encoder and this head stay frozen, so only the image encoder is updated, and every task vector lives in the same parameter space. We minimize the cross-entropy loss with AdamW (learning rate 10^{-5}, no weight decay, batch size 32) for 10,000 steps. The learning rate warms up linearly over the first 10% of steps and then decays linearly. Images are preprocessed with the standard CLIP processor and no data augmentation.

A.1.2 LANGUAGE MODELS

We fully fine-tune four base models: Qwen3-0.6B-Base, Qwen3-1.7B-Base,Qwen3-4B-Base (Yang et al., 2025), and Llama-3.2-1B-Instruct (Grattafiori et al., 2024), on each of four tasks: Banking77 (Loukas et al., 2023), DDXPlus (Tchango et al., 2022), IFEval (Zhou et al., 2023), and Usefulness Judge (Chen et al., 2026). Training uses Axolotl[2] with supervised fine-tuning on chat-formatted

[2] https://github.com/axolotl-ai-cloud/axolotl

data. The loss covers only assistant turns, including the end-of-turn token. We use AdamW ($\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, no weight decay), a peak learning rate of 4×10^{-5} with 10% linear warmup and cosine decay, an effective batch size of 36, 6 epochs, bf16 precision. The maximum sequence length is 4,096 (8,192 for DDXPlus). We select the checkpoints with best evaluation accuracies on a held-out set.

A.2 Data details and split

Each task has three disjoint splits. The *train* split is used only to fine-tune the individual models. The *validation* split is the only data merging strategies can access. The *test* split is used only for the final evaluations. All random splits use a fixed seed (42).

Vision. We randomly hold out 10% of each dataset's official training split as the validation set, fine-tune on the remaining 90%, and test on the complete official test split (Food101 has no test split, so we use its official validation split). In the data-scaling experiments, we subsample $k \in \{1, 2, 4, 8, 16, 32, 64\}$ examples per class from this fixed validation set, using three seeds. The number of classes and samples of each split is listed in Table 5.

Task	#Classes	#Train	#Valid	#Test
DTD	47	3,384	376	1,880
EuroSAT	10	19,440	2,160	2,700
FER2013	7	25,838	2,871	7,178
Food101	101	68,175	7,575	25,250
GTSRB	43	23,976	2,664	12,630
MNIST	10	54,000	6,000	10,000
RESISC45	45	17,010	1,890	6,300
Stanford Cars	196	7,330	814	8,041
SUN397	397	17,865	1,985	19,850

Table 5: Vision data splits.

Language. Table 6 lists the data source for each task, and Table 7 lists the number of samples of each split. Except for IFEval, validation and test are a random split of the same held-out pool. Every task uses a validation:test ratio of 1:9, because weight GD and subspace GD train on the validation set and a larger one would favor them over coefficient search.

Task	Dataset	Used for
Banking77	`theblackcat102/bank77_m`	Train
	`appier-ai-research/bank-77` (`in_domain`)	Valid, Test
DDXPlus	`appier-ai-research/StreamBench` (`validate`)	Train
	`appier-ai-research/StreamBench` (`test`)	Valid, Test
IFEval	`argilla/ifeval-like-data` (`filtered`)	Train, Valid
	`google/IFEval`	Test
Usefulness Judge	`miulab/usefulness-judge` (`train`)	Train
	`miulab/usefulness-judge` (`test`)	Valid, Test

Table 6: Data sources for the language tasks (Hugging Face dataset IDs). IFEval has no reference answers, so its fine-tuning data and validation set both come from `argilla/ifeval-like-data`. We keep only validation responses that pass strict prompt-level accuracy, and test on the complete original IFEval benchmark.

Task	#Train	#Valid	#Test
Banking77	15,004	100	900
DDXPlus	1,372	176	1,588
IFEval	14,448	60	541
Usefulness Judge	30,732	25	225

Table 7: Language data splits.

A.3 Subspace GD learning rate sweep and initialization point

To find the task vector subspace's best performance, we tested different learning rates for subspace GD. Table 8 shows that accuracy is stable across learning rates, and we chose 10^{-2} to balance accuracy and speed. We report the subspace GD from coefficient best (Coeff. Best) to since it generally has higher performance.

Table 8: **Subspace GD with different learning rates on Task Arithmetic and TIES for ViT-B-32.** The performance is stable for learning rate from 10^{-4} to 10^{-1}.

	Task Arithmetic		TIES	
LR/Initialization point	**Avg**	**Coeff. Best**	**Merged**	**Coeff. Best**
10^{-4}	**65.22**	**65.26**	67.05	66.75
10^{-3}	65.00	65.07	66.73	66.71
10^{-2} (def.)	65.15	64.99	66.71	66.81
10^{-1}	65.00	64.75	66.83	**67.03**
1	63.08	61.78	**67.07**	66.78

A.4 Weight GD learning rate sweep and initialization point

To test the robustness of gradient descent on model weights, we tested different learning rates for GD. Across several settings in Table 9 and Table 10, setting learning rate to 10^{-5} mostly has best performance. Meanwhile, GD from better initialization (best coefficient found by coefficient search, Coeff. Best) has better performance. In the main paper (Section 3), we report the GD performance from the Naive baseline, so it does not depend on the coefficient search.

Table 9: **Effect of learning rate across architectures and merging methods for gradient descent on weights.** Gradient descent's initialization points affect the performance.

	ViT-B-32				ViT-B-16			
	Task Arithmetic		TIES		Task Arithmetic		TIES	
LR	**Avg**	**Coeff. Best**	**Avg**	**Coeff. Best**	**Avg**	**Coeff. Best**	**Avg**	**Coeff. Best**
10^{-6}	79.52	79.41	77.82	81.25	82.58	83.07	81.18	**84.94**
10^{-5} (def.)	**80.61**	**80.37**	**79.49**	**82.08**	**83.42**	**84.14**	**82.81**	84.71
10^{-4}	71.98	71.41	71.54	72.35	69.24	69.29	69.03	68.96

Table 10: **Effect of learning rate across different optimization budgets for gradient descent on weights.** Same conclusion with the Table 9.

Budget	LR	ViT-B-32 Avg	ViT-B-32 Coeff. Best	ViT-B-16 Avg	ViT-B-16 Coeff. Best
1 / class	10^{-6}	**69.12**	**68.32**	73.65	75.35
	10^{-5} (def.)	67.02	66.17	**73.83**	**75.91**
	10^{-4}	49.41	47.49	52.64	50.32
4 / class	10^{-6}	74.03	75.74	77.44	79.41
	10^{-5} (def.)	**75.03**	**76.89**	**78.24**	**80.37**
	10^{-4}	63.45	64.37	59.21	61.17
16 / class	10^{-6}	76.92	76.66	80.22	81.85
	10^{-5} (def.)	**77.98**	**77.66**	**81.35**	**82.54**
	10^{-4}	68.79	68.91	68.12	68.77

A.5 Different optimizers

In this section, we analyze whether gradient descent for model merging depends on specific optimizers. As shown in Table 11, optimizing using SGD does not decrease the performance much. Since AdamW is more efficient than SGD, we report AdamW's results in the main paper. Nonetheless, practitioners could choose SGD if the GPU memory size is limited.

Table 11: **Comparison of weight-space optimizers across different data-size merging methods for ViT-B-32.** Coefficient search and Subspace GD are provided for reference. The empirical results show that Weight GD does not depend on specific optimizers. Meanwhile, they mostly outperform coefficient search and subspace GD, which optimize within the task vector subspace.

	Task Arithmetic								TIES
Method / Data size	1 / class	2 / class	4 / class	8 / class	16 / class	32 / class	64 / class	Full	Full
Coefficient search	62.41	61.97	61.97	62.41	62.41	62.41	62.41	62.41	67.35
Subspace GD	63.74	64.27	65.06	64.88	64.94	64.72	64.71	65.08	66.73
Weight GD (AdamW)	67.23	72.05	75.17	76.93	77.97	78.85	79.43	80.53	82.07
Weight GD (SGD)	62.17	68.86	72.53	75.23	76.68	77.66	78.62	79.68	81.79
SGD − AdamW	−5.06	−3.19	−2.64	−1.70	−1.29	−1.19	−0.81	−0.85	−0.28

A.6 HYPERPARAMETERS OF STRATEGIES

Table 12 and Table 13 provide the hyperparameters for the Section 3.

Table 12: Hyperparameters used for vision exp.

Hyperparameter	ViT-B-32	ViT-B-16	ViT-L-14
Batch size	32	8	4
Coefficient search $\lambda_{\min}$	0.0	0.0	0.0
Coefficient search $\lambda_{\max}$	1.0	1.0	1.0
Coefficient search steps	11	11	11
Weight GD optimizer	AdamW	AdamW	AdamW
Weight GD learning rate	10^{-5}	10^{-5}	10^{-5}
Weight GD epochs	10	10	10
Weight GD warmup ratio	0.1	0.1	0.1
Weight GD scheduler	cosine	cosine	cosine
Weight GD patience	5	5	5
Subspace GD optimizer	AdamW	AdamW	AdamW
Subspace GD learning rate	10^{-2}	10^{-2}	10^{-2}
Subspace GD epochs	20	20	20
Subspace GD warmup ratio	0.1	0.1	0.1
Subspace GD scheduler	cosine	cosine	cosine
Subspace GD patience	5	5	5
TIES density	0.2	0.2	0.2
DARE drop rate	0.5	0.5	0.5

Table 13: Hyperparameters used for language models.

Hyperparameter	Qwen3-0.6B	Qwen3-1.7B	Qwen3-4B
Coefficient search $\lambda_{\min}$	0.1	0.1	0.1
Coefficient search $\lambda_{\max}$	1.0	1.0	1.0
Coefficient search steps	10	10	10
Weight GD optimizer	AdamW	AdamW	AdamW
Weight GD learning rate	3×10^{-5}	3×10^{-5}	3×10^{-5}
Weight GD epochs	5	5	5
Weight GD warmup ratio	0.1	0.1	0.1
Weight GD scheduler	cosine	cosine	cosine
Weight GD patience	none	none	none
Weight GD micro-batch size	1	1	1
Weight GD grad accum steps	4	4	4
Weight GD effective batch size	4	4	4
Weight GD max sequence length	2048	2048	2048
Weight GD LoRA learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
LoRA rank	16	16	16
LoRA alpha	32	32	32
LoRA dropout	0.05	0.05	0.05
Subspace GD optimizer	AdamW	AdamW	AdamW
Subspace GD learning rate	10^{-2}	10^{-2}	10^{-2}
Subspace GD epochs	5	5	5
Subspace GD warmup ratio	0.1	0.1	0.1
Subspace GD scheduler	cosine	cosine	cosine
Subspace GD micro-batch size	1	1	1
Subspace GD grad accum steps	4	4	4
Subspace GD effective batch size	4	4	4
Generation temperature	0.01	0.01	0.01
Generation top-p	0.95	0.95	0.95
Generation thinking	off	off	off
Max new tokens	256 (IFEval), 16 (other datasets)		
TIES density	0.2	0.2	0.2
DARE drop rate	0.5	0.5	0.5

B MORE RELATED WORKS

In this section, we discuss more details of the related works.

B.1 VARIANTS OF TASK-WISE MERGING

Equation (1) mainly discusses the **task-wise merging**, where the coefficients are set to each task. We now discuss other variants.

Global-wise merging produces a single advanced task vector $\tilde{\tau} = f(\{\tau_i\}_{i=1}^T)$, A single coefficient $\lambda \in \mathbb{R}$ is shared across all tasks:

$$\theta_{\text{merged}} = \theta_0 + \lambda \cdot \tilde{\tau}. \tag{2}$$

TIES (Yadav et al., 2023) and TSV-M (Gargiulo et al., 2025) belong to this class. Their task vector subspace is a one-dimensional subspace, and optimizing this single coefficient also implicitly regularizes the candidate models into this one-dimensional task-vector subspace. One can rewrite Equation (2) into Equation (1) (task-wise merging).

Layer-wise merging (Yang et al., 2024b; Sun et al., 2025; Xu et al., 2025) further merged parameters at each layer l are merged independently:

$$\theta_{\text{merge}}^{(l)} = \theta_0^{(l)} + \sum_{i=1}^{T} \lambda_{i,l} \cdot \tilde{\tau}_{i,l}, \quad \forall l = 1, \ldots, L, \tag{3}$$

where L is the number of layers. The final merged model is

$$\theta_{\text{merge}} = \theta_0 + \sum_{i=1}^{T} \sum_{l=1}^{L} \lambda_{i,l} \cdot \tilde{\tau}_{i,l}, \tag{4}$$

where $\tilde{\tau}_{i,l} \in \mathbb{R}^d$ denotes the layer-l component of task vector i, embedded in the full weight space with d parameters, and can be viewed as a type of advanced task vector. Layer-wise merging uses $L * T$ coefficients, compared with at most T coefficients for task-wise merging. Therefore, it has more trainable parameters (coefficients) than task-wise merging. However, it still allows fewer trainable parameters (coefficients) than full-weight fine-tuning. Essentially, layer-wise merging creates a larger task-vector subspace that regularizes candidate models into this subspace.

Notably, Xu et al. (2025) compared their layer-wise merging method with full-weight fine-tuning with auxiliary data. However, they did not specify the initialization of the fine-tuning with gradient descent, and our studies in Section 3, Section 5.2, Section A.4 and Section C.2 have demonstrated the importance of the initialization point for gradient descent. However, results in Section D.2 and Appendix D.3 show that full-weight fine-tuning can underperform task-wise or layer-wise merging due to overfitting when data are scarce. How strong the regularization should be depends on the auxiliary dataset.

B.2 DETAILED DISCUSSION REGARDING COMPUTE AND DATA USAGE

Besides the related work discussed in Section 5.1, we also identify other existing works (Tang et al., 2024; Lee et al., 2026) doing backpropagation through the model weights. This suggests that, when computational resources allow, practitioners may trade additional computation for better performance.

We also clarify that model merging methods classified as data-free in prior work (Cheng et al., 2025; Chaves et al., 2026; Yang et al., 2026) are typically insensitive to coefficient choices (Gargiulo et al., 2025) or have no tunable coefficients or hyperparameters (Cheng et al., 2025). Section 3 show that for methods that have coefficients, carefully tuning them leads to better performance. Even for merging methods without tunable coefficients, auxiliary data can still be used to further improve performance through gradient descent. As shown in Section 5.2, merged models provide strong initialization points for gradient descent. Thus, when auxiliary data are available, further optimization may help close the performance gap between a merged model and individual task-specific models.

C MAIN RESULTS DETAILS

C.1 FINDING SUBSPACE BEST PERFORMANCE

We tried three methods to find the best performance on the task-vector subspace by using auxiliary data: gradient descent on coefficients (Subspace GD), fine-grained diagonal search, and Bayesian optimization. For a fair comparison with gradient descent on weights, we aim to find the best performance on the subspace through **training** data. This separates the question of optimization capacity from train-test generalization and gives the task-vector subspace a favorable comparison. Nonetheless, we still report the best **test** performance among these methods as the subspace best. For Bayesian optimization (BO), we do $2 * T + 1$ warmups and sample 50 points in total, where T is the number of task vectors. To the best of our knowledge, we are the first one that to apply Bayesian Optimization to the coefficients of vision model merging.

For vision models, the results are in Table 14. BO and subspace GD have similar performance. Since Subspace GD is optimized based on cross-entropy loss, there is an accuracy-loss mismatch that makes BO have better accuracy. Nonetheless, Subspace GD always has the lowest training loss. We did not do Subspace GD on ViT-L-14 since it requires huge GPU memory that is beyond our computational resources. The fine-grained diagonal search and BO have similar performance on TIES, since it only has one coefficient. Since BO and subspace GD have similar performance, we only run subspace GD for DARE, and only run fine-grained grid search for TSV-M.

For language models, the results are in Table 15. BO has higher scores (accuracy) than subspace GD since the accuracy-loss mismatch is more severe in LLMs. We did not do Subspace GD on Qwen3-4B since it requires huge GPU memory that is beyond our computational resources.

Table 14: **Finding the best average accuracy on the task-vector subspace for vision models.** Bayesian Optimization (BO) and Subspace GD have similar performance.

Merging	Method	ViT-B-32	ViT-B-16	ViT-L-14
TA	BO	**65.10**	**70.24**	**79.58**
	Subspace GD	65.07	68.90	79.31
TIES	BO	**67.34**	**72.49**	**77.75**
	Subspace GD	66.72	71.84	–

Table 15: **Finding the best average score on the task-vector subspace for language models.** If Subspace GD is worse than coefficient search, we do grid search instead. Bayesian Optimization (BO) generally has the best performance.

Merging	Method	Qwen3-0.6B	Qwen3-1.7B	Qwen3-4B	Llama-3.2-1B
TA	BO	**70.20**	**80.51**	**81.13**	76.40
	Subspace GD / Grid Search	66.84	73.25	–	**77.19**
TIES	BO	**54.09**	63.50	**68.38**	**57.72**
	Subspace GD / Grid Search	53.55	**63.78**	–	57.67
DARE	BO	**68.58**	**77.41**	**80.04**	76.57
	Subspace GD / Grid Search	68.52	73.57	–	**77.52**
TSV-M	BO	**69.71**	78.52	**79.74**	79.34
	Subspace GD / Grid Search	69.52	**78.67**	–	**80.48**

C.2 FULL-WEIGHT VS. LORA FINE-TUNING AND THE EFFECT OF INITIALIZATION POINTS

In this section, we compare the performance of full-weight fine-tuning and LoRA fine-tuning (Hu et al., 2022) for the LLM model merging. As shown in Table 16, LoRA fine-tuning usually has better performance on merging tasks. We believe this is because the auxiliary dataset is small, and LoRA has fewer parameters than the full weights, reducing the risk of overfitting. We report the LoRA fine-tuning results in Figure 2. We also test two initialization points. Same as the naive baseline, we report 1.0 for TIES and TSV-M, $1/T$ for TA and DARE in Figure 2.

Table 16: **Average score of full-weight fine-tuning and LORA fine-tuning across different merging methods and initialization points.** We set the rank of LORA to 16, and test two initialization points in the task-vector subspace: all coefficients set to the average $1/T$ or 1.0. We did not evaluate full-weight fine-tuning on Qwen3-4B due to computational resource constraints. Overall, LORA fine-tuning has better performance on the merging tasks.

Merging	Method	Qwen3-0.6B	Qwen3-1.7B	Qwen3-4B	Llama-3.2-1B
Pretrained	full-weight fine-tuning	53.76	58.42	–	51.83
	LORA fine-tuning	60.77	65.02	70.05	53.11
TA	full-weight fine-tuning ($1/T$)	70.22	75.05	–	61.03
	full-weight fine-tuning (1.0)	70.98	76.10	–	78.83
	LORA fine-tuning ($1/T$)	69.94	76.93	83.91	80.77
	LORA fine-tuning (1.0)	73.02	81.48	83.40	82.04
TIES	full-weight fine-tuning ($1/T$)	56.68	62.92	–	60.86
	full-weight fine-tuning (1.0)	67.72	71.87	–	76.38
	LORA fine-tuning ($1/T$)	62.88	69.29	77.26	68.48
	LORA fine-tuning (1.0)	69.67	77.01	82.02	79.83
DARE	full-weight fine-tuning ($1/T$)	70.68	74.84	–	59.89
	full-weight fine-tuning (1.0)	71.18	75.31	–	77.53
	LORA fine-tuning ($1/T$)	70.42	77.40	84.16	82.42
	LORA fine-tuning (1.0)	73.19	81.03	82.99	81.26
TSV-M	full-weight fine-tuning ($1/T$)	68.48	70.27	–	57.75
	full-weight fine-tuning (1.0)	75.78	81.77	–	83.39
	LORA fine-tuning ($1/T$)	68.63	73.82	80.23	76.67
	LORA fine-tuning (1.0)	77.78	81.85	87.23	82.99

C.3 More data scaling results

Figure 9 shows the performance of different strategies for using auxiliary data under different dataset sizes for TIES. Since TIES only has one coefficient, the subspace GD and coefficient search have similar performance. However, doing gradient descent on weights (GD) consistently has better accuracy across different dataset sizes.

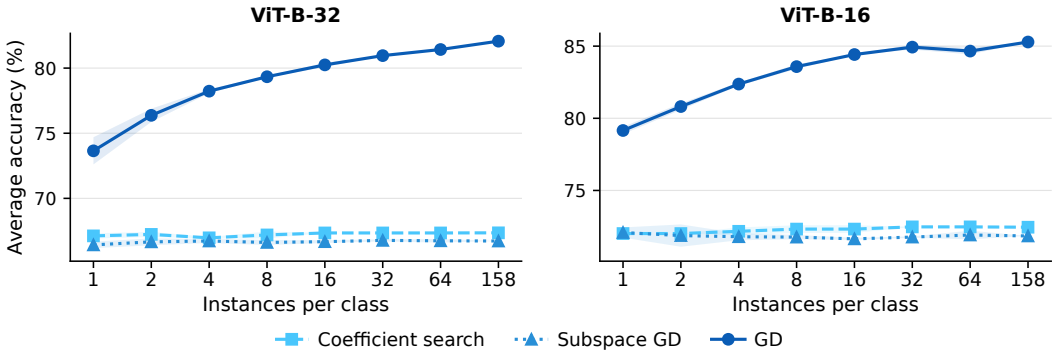


Figure 9: **Average test accuracy under different sizes and uses of auxiliary data for TIES.** We downsample the auxiliary data and compare three different uses of auxiliary data for TIES. We run three random seeds for downsampling, and report error bars as shaded regions. The findings are consistent with the TA's result in Figure 3.

C.4 Main results in Tables

Figure 2's numerical values are shown in Table 17. We also include the results on Llama-3.2-1B, and the findings are consistent with Qwen3 models: gradient descent on model weights outperforms optimizing within the task-vector subspace. Table 18 and Table 19 report the data-scaling results for TA and TIES, respectively.

Table 17: **Numerical values of Figure 2**. We also provide the Llama-3.2-1B results, and the results are consistent.

		Vision			Language			
Merging Method	**Baseline**	**ViT-B-32**	**ViT-B-16**	**ViT-L-14**	**Qwen3-0.6B**	**Qwen3-1.7B**	**Qwen3-4B**	**Llama-3.2-1B**
Pretrained	gradient descent	78.54	82.51	84.98	60.77	65.02	70.05	53.11
TA	naive baseline	62.87	67.57	74.61	60.13	64.58	75.49	74.87
	coefficient search	62.41	68.06	76.68	63.62	73.65	79.99	76.22
	subspace best	65.10	70.24	79.58	70.20	80.51	81.13	77.19
	gradient descent	80.53	83.77	85.85	69.94	81.48	83.91	80.77
TIES	naive baseline	62.25	69.52	76.27	49.98	57.93	68.26	57.84
	coefficient search	67.35	72.45	77.72	50.09	57.93	67.87	58.22
	subspace best	67.35	72.49	77.78	54.09	63.78	68.38	57.72
	gradient descent	82.06	85.28	87.27	69.67	77.01	82.02	79.83
DARE	naive baseline	62.73	67.63	74.61	60.46	65.64	74.70	75.15
	coefficient search	62.38	67.94	76.71	63.88	72.30	79.86	77.07
	subspace best	64.55	68.97	79.27	68.58	77.41	80.04	77.52
	gradient descent	80.52	83.72	84.48	70.42	81.03	84.16	82.42
TSV-M	naive baseline	74.79	80.38	87.29	68.32	77.89	81.26	70.36
	coefficient search	78.05	81.91	87.53	69.43	78.37	79.85	79.77
	subspace best	78.05	81.96	87.53	69.71	78.67	79.74	80.48
	gradient descent	83.88	86.37	88.26	77.78	81.85	87.23	82.99

Table 18: **Numerial values of Figure 3**: data scaling for TA.

Method / Data size	1	2	4	8	16	32	64	158
ViT-B-32								
Coefficient search	62.42 ± 0.00	62.12 ± 0.25	62.27 ± 0.26	62.27 ± 0.25	62.13 ± 0.25	62.27 ± 0.25	62.42 ± 0.00	62.41
Subspace GD	62.86 ± 0.99	64.26 ± 0.59	64.39 ± 0.62	64.56 ± 0.31	64.71 ± 0.20	64.31 ± 0.42	64.66 ± 0.29	65.08
GD	68.44 ± 1.24	72.34 ± 0.52	75.16 ± 0.02	76.76 ± 0.17	77.92 ± 0.05	78.81 ± 0.06	79.29 ± 0.12	80.53
ViT-B-16								
Coefficient search	68.07 ± 0.01	68.07 ± 0.01	67.75 ± 0.56	68.07 ± 0.01	68.07 ± 0.01	68.07 ± 0.01	68.07 ± 0.01	68.07
Subspace GD	67.78 ± 1.55	68.92 ± 0.53	68.23 ± 0.38	68.71 ± 0.19	68.72 ± 0.23	68.99 ± 0.28	68.78 ± 0.09	68.91
GD	74.70 ± 0.66	76.45 ± 0.49	78.51 ± 0.23	80.17 ± 0.28	81.41 ± 0.20	82.27 ± 0.13	82.78 ± 0.11	83.78

Table 19: **Numerial values of Figure 9**: data scaling for TIES.

Method / Data size	1	2	4	8	16	32	64	158
ViT-B-32								
Coefficient search	67.12 ± 0.20	67.23 ± 0.20	66.97 ± 0.06	67.19 ± 0.27	67.35 ± 0.00	67.35 ± 0.00	67.35 ± 0.00	67.35
Subspace GD	66.43 ± 0.25	66.66 ± 0.26	66.73 ± 0.15	66.62 ± 0.19	66.68 ± 0.11	66.77 ± 0.08	66.74 ± 0.14	66.73
GD	73.65 ± 1.04	76.37 ± 0.51	78.23 ± 0.16	79.34 ± 0.06	80.25 ± 0.16	80.96 ± 0.02	81.43 ± 0.02	82.07
ViT-B-16								
Coefficient search	72.01 ± 0.00	72.01 ± 0.00	72.17 ± 0.27	72.32 ± 0.27	72.32 ± 0.27	72.48 ± 0.00	72.48 ± 0.00	72.45
Subspace GD	72.08 ± 0.35	71.87 ± 0.77	71.79 ± 0.25	71.77 ± 0.15	71.65 ± 0.07	71.77 ± 0.05	71.91 ± 0.30	71.85
GD	79.15 ± 0.25	80.81 ± 0.22	82.37 ± 0.11	83.58 ± 0.16	84.42 ± 0.03	84.93 ± 0.17	84.66 ± 0.23	85.29

C.5 S-RANDOPT IMPLEMENTATION DETAILS

In Section 4.2, we calculate the gradient from the best coefficient search point for vision models, and from the subspace best for the language models. From this init point, we create an extended space around it, which we name the *Sample Space*. From this initialization point θ_{init}, we construct a bounded region, which we call the *Sample Space*. Let d denote the component of gradient descent that is orthogonal to the task-vector subspace. We define the Sample Space as

$$\mathcal{S} = \left\{ \theta_{\text{init}} + \sum_{i=1}^{T} a_i \tilde{\tau}_i + b\, d \;:\; a_i \in [-\alpha, \alpha],\; b \in [0, \beta] \right\}.$$

Thus, each task-vector coefficient is allowed to vary within $[-\alpha, \alpha]$, while the coefficient along the additional descent direction varies within $[0, \beta]$. We set $(\alpha, \beta) = (0.1, 0.01)$ for vision models and $(\alpha, \beta) = (0.05, 0.0005)$ for language models.

D MORE ANALYSES

D.1 COMPLETE ANALYSIS OF INITIALIZATIONS FOR GRADIENT DESCENT

Figure 10 shows the full results of gradient descent with different initializations. Across all models, the message is consistent: merging methods provide strong initializations for gradient descent, and they are better than fine-tuning from the pretrained model or random initializations.

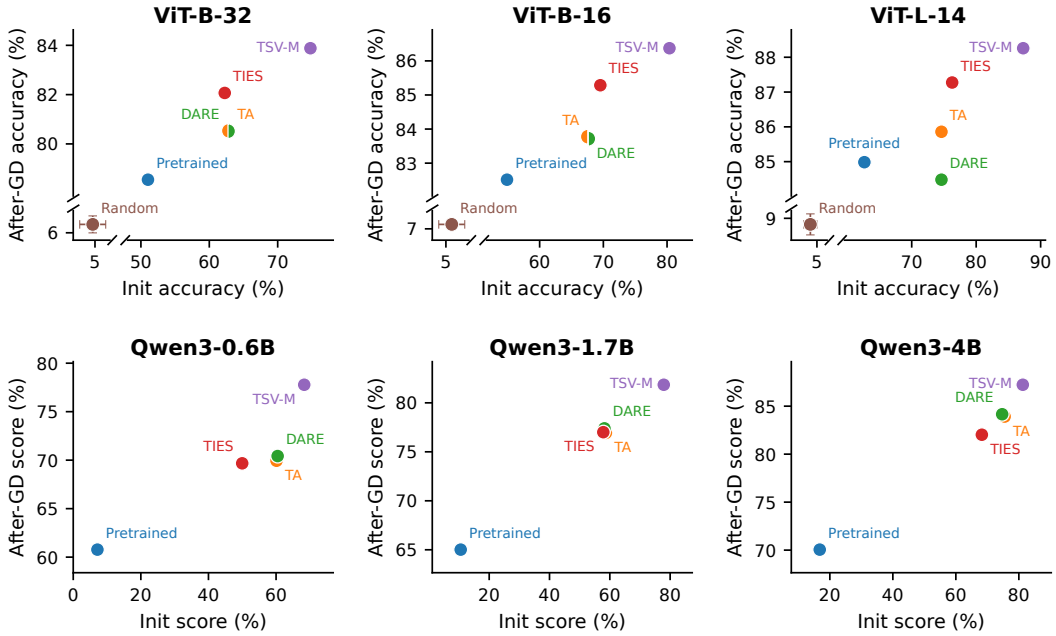


Figure 10: **Gradient Descent (GD) Initialization results across vision and language models.** For vision, we measure average test accuracy, and report the average accuracy of three random initialization points. Init accuracy is the naive coefficient's performance.

D.2 2-TASK SETTING

For the two-task scenario, we evaluate two task combinations: EuroSAT (Helber et al., 2019) and GTSRB (Stallkamp et al., 2011), as well as MNIST (LeCun, 1998) and FER2013 (Goodfellow et al., 2013). We tested both regularized and unregularized versions of gradient descent. We apply the ℓ_2 norm from the starting point, L2-SP, as the regularization added to the loss function: $\frac{\alpha}{2}\|\theta - \theta_{init}\|_2^2$. θ_{init} is the initialization point for gradient descent, and we set $\alpha = 3$.

Figure 11 shows the experimental results. GD and regularized GD learn the training set perfectly across all data sizes in two scenarios, while coefficient search and subspace GD struggle even with more data. Overall, when data are scarce, GD may lead to overfitting, but the risk is smaller when data are sufficient. In contrast, coefficient search and subspace GD do not fit the training set well, so their test performance changes little as more data become available. The appropriate regularization strength for balancing the bias-variance trade-off is a key factor in determining how auxiliary data should be used.

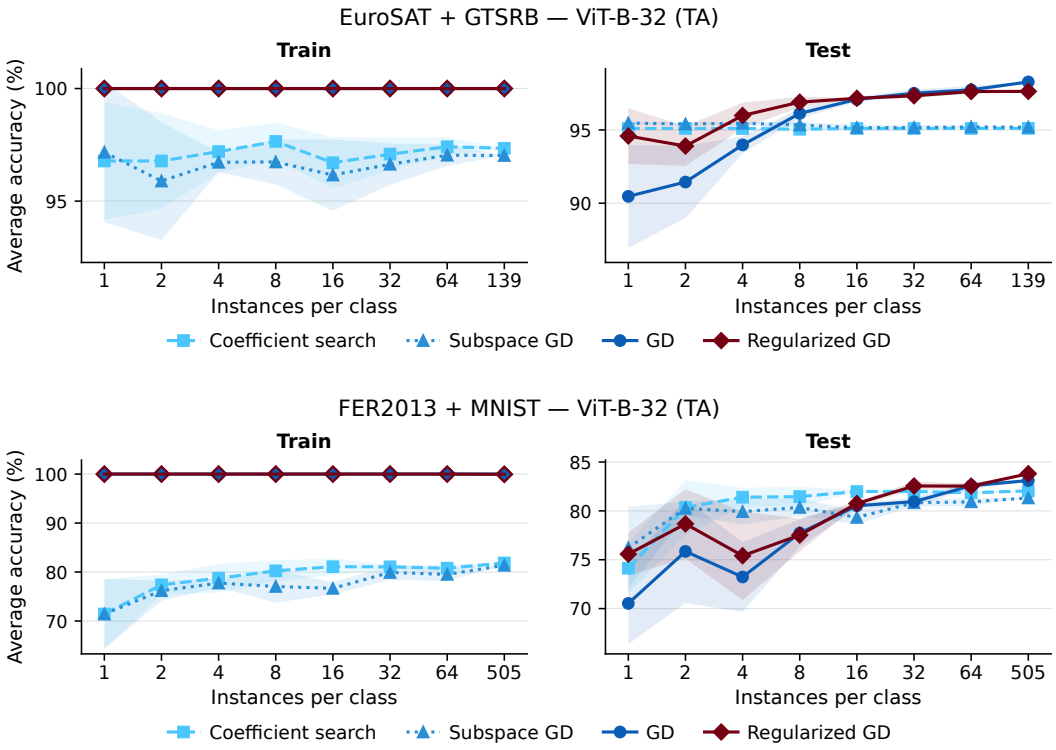


Figure 11: **Train and test set accuracy of different strategies in 2-task scenarios.** GD overfits when data are scarce, and additional regularization improves performance. On the other hand, the test accuracy of coefficient search and subspace GD is largely insensitive to data size.

D.3 TEST-TIME ADAPTATION FULL RESULTS

Experiment Setup We compare GD with two test-time adaptation methods for model merging, AdaMerging (Yang et al., 2024b) and DivMerge (Touayouch et al., 2026). Both methods learn the merging coefficients on unlabeled test data: AdaMerging minimizes the prediction entropy of the merged model, while DivMerge minimizes the Jensen–Shannon divergence between the predictions of the merged model and those of the task-specific models. We apply both task-wise and layer-wise merging, which are described in detail in Section B, and initialize the merging coefficients to $1/T$.

For GD, we use the predictions of the task-specific models on the unlabeled test data as supervision. We consider two types of labels: soft labels use the softmax outputs of the task-specific models directly, while hard labels convert them into one-hot labels. All methods use 1,000 unlabeled test samples per task.

Experiment results Table 20 reports the full experiment results. GD on weights has the best performance in smaller models. Layer-wise DivMerge has more trainable parameters than the task-wise version, yet fewer trainable parameters than full-weight fine-tuning. Therefore, it has less risk of overfitting the unlabeled test data, which turns out to achieve the best performance in ViT-L-14. We expect applying regularization on GD will improve it performance, as shown in Section 5.3. Overall, the empirical results demonstrate that the bias-variance tradeoff is the key principle in deciding how strong the regularization should be when using auxiliary data.

Table 20: **Full comparison results of test-time adaptation methods for TA.**

Method	ViT-B-32	ViT-B-16	ViT-L-14
(task-wise) AdaMerging	62.21	66.80	76.63
(task-wise) DivMerge	65.43	69.62	78.49
(layer-wise) AdaMerging	75.67	79.67	87.42
(layer-wise) DivMerge	80.56	84.48	**89.90**
GD Soft	**82.43**	**85.51**	88.15
GD Hard	82.08	85.12	88.33

D.4 LLM PERFORMANCE ON CONTROL TASKS

When merging LLMs, the merged model should not only acquire the capabilities of the expert models, but also retain the general capabilities of the pre-trained model. For example, a multilingual medical assistant merged from a multilingual expert and a medical expert should still retain capabilities such as mathematical reasoning and code generation. Since gradient descent optimizes the model weights on the target tasks only, it may cause forgetting of such capabilities (Luo et al., 2025). We therefore compare it with merging within the task-vector subspace on control tasks, i.e., tasks not targeted by any expert model. We use GSM8K (Cobbe et al., 2021) and MBPP (Austin et al., 2021) as control tasks, which evaluate mathematical reasoning and code generation, respectively.

Table 21 reports the results. Overall, gradient descent does not systematically cause more forgetting than coefficient search. On the 1.7B model, it improves the average accuracy on both control tasks (GSM8K: 64.8 $\rightarrow$ 74.5; MBPP: 41.7 $\rightarrow$ 49.8), reaching a level comparable to the pre-trained model. On the 0.6B model, it improves GSM8K (40.7 $\rightarrow$ 50.6) but degrades MBPP (30.7 $\rightarrow$ 25.7). Moreover, coefficient search itself can cause severe forgetting: on the 1.7B model, TA drops to 21.7 on MBPP, whereas gradient descent from the same merged model reaches 50.0. Staying within the task-vector subspace thus does not guarantee the retention of general capabilities.

Table 21: **Performance on target and control tasks for merged LLMs.** We compare coefficient search within the task-vector subspace (Coeff.) with gradient descent on LoRA parameters (GD). Seen denotes the average performance on the target tasks; GSM8K and MBPP are control tasks not targeted by any expert model. All results are evaluated on 300 samples per task. †The pre-trained model is evaluated with raw few-shot prompts instead of the chat template and serves only as a reference. Avg. denotes the average over the four merging methods, excluding the pre-trained model. The better result of each pair is shown in bold.

		Seen		**GSM8K**		**MBPP**	
Model	**Method**	Coeff.	GD	Coeff.	GD	Coeff.	GD
0.6B	Pre-trained†	–		48.0		32.0	
	TA	63.6	**69.9**	42.0	**55.7**	**32.3**	26.7
	DARE	63.9	**70.4**	32.3	**52.0**	**31.7**	25.7
	TIES	50.1	**69.7**	**58.3**	49.7	30.0	**32.7**
	TSV-M	69.4	**77.8**	30.3	**45.0**	**28.7**	17.7
	Avg.	61.8	**72.0**	40.7	**50.6**	**30.7**	25.7
1.7B	Pre-trained†	–		73.7		50.7	
	TA	73.6	**76.9**	61.3	**77.0**	21.7	**50.0**
	DARE	72.3	**77.4**	72.7	72.7	44.3	**51.3**
	TIES	57.9	**77.0**	55.0	**78.0**	**51.3**	48.7
	TSV-M	78.4	**81.8**	70.0	**70.3**	49.3	49.3
	Avg.	70.6	**78.3**	64.8	**74.5**	41.7	**49.8**