SIMMERGE: LEARNING TO SELECT MERGE OPERATORS FROM SIMILARITY SIGNALS

Oliver Bolton
Cohere Labs

Aakanksha
Cohere

Arash Ahmadian
Google

Sara Hooker
Adaption Labs

Marzieh Fadaee
Cohere Labs
marzieh@cohere.com

Beyza Ermis
Cohere Labs
beyza@cohere.com

ABSTRACT

Model merging combines multiple models into a single model with aggregated capabilities, making it a powerful tool for large language model (LLM) development. However, merge performance heavily depends on the choice of merge operator and merge order, often requiring expensive merge-and-evaluate searches to maximize. In this work, we introduce SIMMERGE, a predictive merge-selection method that identifies high-performing merges using inexpensive similarity signals between models. Given a small set of unlabeled probes, SIMMERGE extracts functional and structural features to predict the performance of candidate two-way merges, enabling merge operator and order selection without iterative evaluation. We show that SIMMERGE consistently outperforms the best fixed merge operator across 7B-parameter LLMs and generalizes to multi-way merges and 111B-parameter LLMs without retraining. We further introduce a bandit variant that supports adding new tasks and operators online. Our results suggest that learning how to merge enables scalable model composition when checkpoint catalogs are large and evaluation budgets are limited.

1 INTRODUCTION

Model merging combines multiple fine-tuned checkpoints into a single model, offering an efficient alternative to joint training or inference-time ensembling for post-training model composition. A large body of previous work has focused on designing improved merge operators (Wortsman et al., 2022; Ilharco et al., 2023; Matena & Raffel, 2022; Yadav et al., 2023; Shoemake, 1985; Huang et al., 2024; Stoica et al., 2024). Despite these advances, merge performance remains highly sensitive to merge configuration choices such as the choice of operator and the order in which models are merged. In practice, selecting a good merge configuration often requires costly merge-and-evaluate searches over these options, which quickly become impractical as the number of available checkpoints grows.

We address this selection challenge with **SIMMERGE**, a predictive merge-selection method that uses inexpensive, pre-merge similarity signals to choose merge operators and merge plans without iterative evaluation. Given a small set of unlabeled probes, SIMMERGE extracts both functional similarity (e.g., divergences between model outputs) and structural similarity (e.g., distances in weight space), and learns to predict downstream merge performance. From this SIMMERGE then selects the operators and merge orders that are predicted to give the best performance, selecting from three popular merging operators: LINEAR interpolation (Wortsman et al., 2022), SLERP (Shoemake, 1985), and TIES merging (Yadav et al., 2023).

Although trained only on pairwise merges of 7B models, we find that the same similarity features efficiently score candidate multiway merge plans and transfer to larger 111B-parameter models without retraining. We also demonstrate how these similarity signals can support an online contextual bandit variant of SIMMERGEthat adapts under partial feedback and supports adding new tasks, models, and operators on-the-fly for evolving checkpoint catalogs.

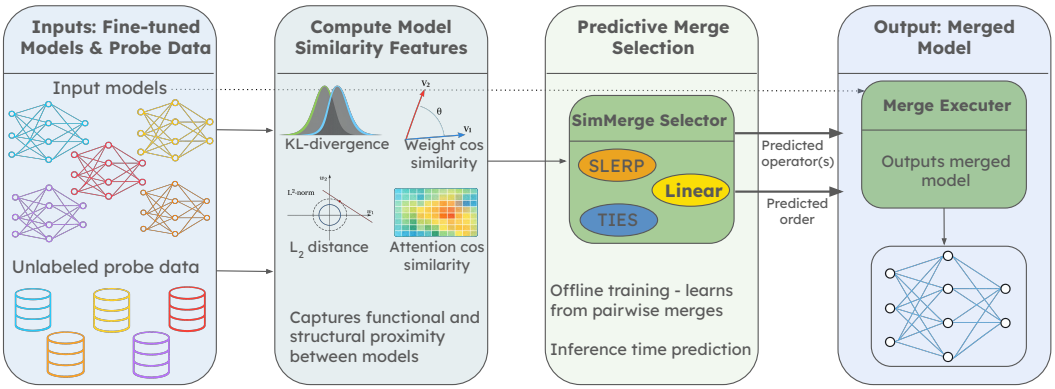


Figure 1: SIMMERGE: Given a set of domain-specialized checkpoints and small unlabeled probe set for each domain, we compute similarity signals, predict the merge operator for each binary merge step and the merge order, and then execute the selected plan once to obtain a single merged model.

Empirically, SIMMERGE consistently outperforms fixed merge operators across code, math, multilingual, and retrieval-augmented generation (RAG) tasks, improving merged model quality while avoiding expensive search. These results suggest that learning to select merges from cheap similarity signals is an effective and practical approach to model composition.

2 METHOD

We consider merging post-trained checkpoints that share a common pretrained base. A pairwise merge combines two models using a specified merge operator, producing a single merged checkpoint. Common operators interpolate parameters or representations in weight space and differ in how they handle interference between models. Multiway merging of k models is performed by iteratively applying pairwise merges, resulting in an ordered merge plan of operators and models. As many merge operators are not associative, different merge orders can yield different final models, making both operator choice and merge order important.

Similarity Signals. SIMMERGE relies on pre-merge similarity signals that capture both functional and structural relationships between models. Functional similarity is measured using model outputs and intermediate representations on small unlabeled probe sets, including divergences between predictive distributions and similarities between intermediate representations across layers. Structural similarity is measured directly in weight space using metrics such as cosine similarity and parameter distance. All metrics are summarized with simple statistics (e.g., means and quantiles) to form a fixed-dimensional feature vector for each ordered model pair. These signals are inexpensive to compute and cached per checkpoint, enabling efficient comparison without downstream evaluation.

Predictive Merge Selection. Given similarity features for an ordered pair of models, SIMMERGE predicts the downstream utility of each candidate merge operator. A lightweight predictor is trained offline on previously evaluated pairwise merges, learning to map similarity features to expected merge performance. At deployment, SIMMERGE selects the operator with the highest predicted utility, replacing merge-and-evaluate search with a single forward pass through the selector. Because common merge operators are not associative, the order in which models are merged can significantly affect performance. SIMMERGE extends pairwise selection to multiway merges by representing a k-way merge as an ordered sequence of pairwise merges. Candidate merge plans are scored using the same pairwise utility predictor, by recursively predicting the utility of each pairwise merge, the overall merge plan performance can be estimated.

Online Extension. In settings where new checkpoints or tasks are introduced over time, we cast operator selection as a contextual bandit problem. Similarity features form the context, merge operators are actions, and downstream performance provides the reward. A lightweight neural-linear bandit updates online from observed outcomes, allowing SIMMERGE to adapt without retraining the similarity feature pipeline.

3 RESULTS

We evaluate SIMMERGE on pairwise and multiway merges across code, math, multilingual, and RAG tasks. We focus on four questions: (i) can similarity features predict the best merge operator for a given model pair, (ii) does this generalize to multiway merges, (iii) does a selector trained on small models transfer to larger ones, and (iv) do the same features support adaptation when merge utilities change online.

Pairwise Operator Selection. We first study pairwise (2-way) merges, which provide the cleanest setting to test whether merge operators can be selected predictively. For each expert–auxiliary model pair, we compute pre-merge similarity features and use SIMMERGE to select among LINEAR, SLERP, and TIES, comparing against each fixed operator.

Figure 2 reports *GapClosed*, the fraction of the expert–auxiliary performance gap recovered by the merged model. Fixed operators exhibit strong task dependence: no single operator performs well across all domains, and some operators regress below the auxiliary baseline on certain tasks. In contrast, SIMMERGE consistently closes the largest fraction of the expert gap across all four domains. These results indicate that inexpensive similarity signals are sufficient to select effective merge operators on a per-instance basis, avoiding the brittleness of global merge recipes.

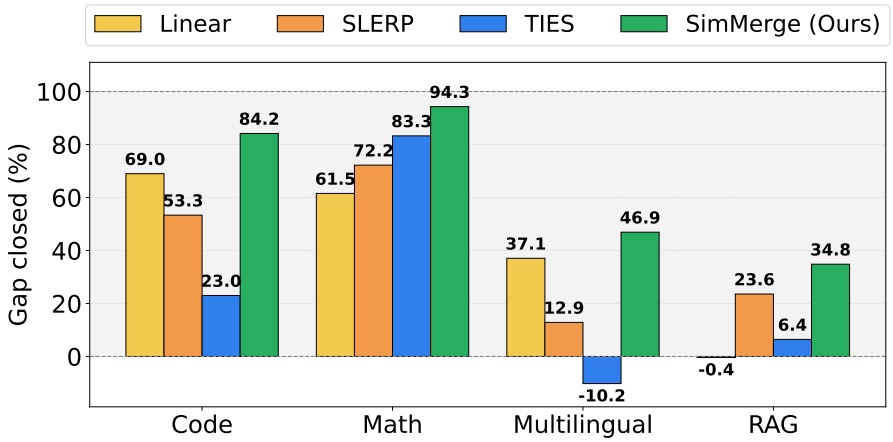


Figure 2: Percentage of the expert–auxiliary performance gap closed by each merge method across Code, Math, Multilingual, and RAG tasks. SIMMERGE consistently recovers a larger fraction of expert performance than fixed merge operators across all domains.

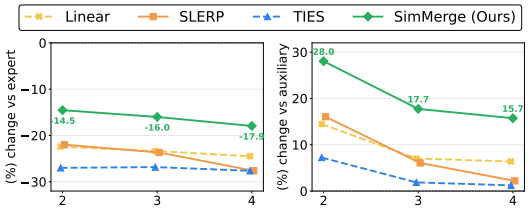


Figure 3: Overall relative performance of 2-, 3-, and 4-way merges, reported as percentage change vs. the task expert (left) and vs. auxiliary models (right), averaged over tasks. SIMMERGE consistently improves over auxiliaries and limits degradation relative to experts for all merge numbers.

Multiway and 111B Merges. To test the robustness of SIMMERGEunder different conditions, we reuse the selector trained exclusively on 7B pairwise merges, to 3-way merges of 7B and 111B-parameter models without retraining. Figure-3 summarizes the effect of increasing the number of merged models from $k = 2$ to 3 and 4. As k increases, all methods degrade relative to the expert and gain less over auxiliaries, but SIMMERGE consistently achieves the strongest expert–auxiliary trade-off across different k, indicating that similarity-driven selection learned from pairwise merges transfers to multiway merge plans. Figure 4 reports macro-averaged performance relative to expert and

auxiliary baselines. Despite the shift in parameter scale, SIMMERGE again yields the best expert–auxiliary trade-off, reducing degradation relative to the expert while achieving the largest gains over auxiliary models compared to all fixed operators. This demonstrates that the similarity features and learned selection strategy transfer across model scales, supporting the use of SIMMERGE in practical settings where large multi model merge plans are expensive to merge and evaluate exhaustively.

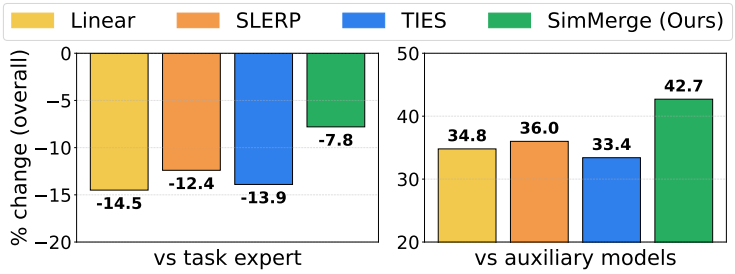


Figure 4: 3-way merging at 111B: macro-averaged performance change vs. the task expert (left) and auxiliary models (right). SIMMERGE yields the smallest expert degradation and largest gains over auxiliaries, despite training only on 7B checkpoints.

Online Adaptation Finally, we consider a setting where merge utilities evolve over time as new checkpoints or tasks are introduced. Rather than retraining the selector, we treat operator selection as a contextual decision problem and update selection online from observed merge outcomes, using the same similarity features as context.

Figure 5 shows cumulative regret and final merge performance for 3-way merges under different online policies. A contextual Thompson-sampling variant rapidly reduces regret compared to uniform random selection and closely approaches an oracle that always selects the best operator in hindsight. In terms of downstream quality, the learned policy achieves substantially better expert–auxiliary trade-offs than random selection. These results indicate that the similarity signals used by SIMMERGE remain informative under distribution shift, and can support lightweight online adaptation when merge utilities change, without altering the underlying feature pipeline.

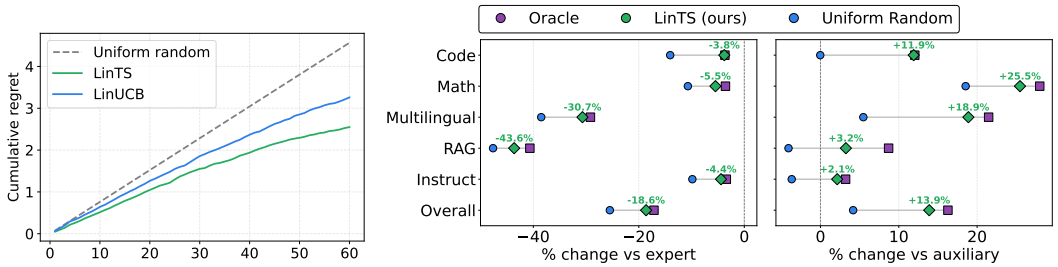


Figure 5: **Left:** Cumulative regret relative to the oracle over 60 rounds for 3-way merges. Uniform random accumulates regret roughly linearly; both contextual bandits reduce regret quickly, with LinTS consistently lower than LinUCB. **Right:** Final percentage change in performance for 3-way merges under uniform random, LinTS, and oracle policies, relative to the task expert (left axis) and auxiliary baseline (right axis), aggregated across domains and macro-average.

4 CONCLUSION

Model merging is a practical form of post-training model composition, but selecting merge operators and orders is inherently uncertain and costly to evaluate. We introduced SIMMERGE, which resolves this uncertainty by predicting effective merge decisions from inexpensive pre-merge similarity signals, avoiding merge-and-evaluate search. We show that SIMMERGE is effective both within domain and when generalizing across diverse domains and model scales, while its online variant further adapts under partial feedback as model pools evolve. These results suggest that learning uncertainty-aware merge selection is a flexible and scalable approach to post-training model updates.

REFERENCES

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 127–135. PMLR, 2013.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Team Cohere, Arash Ahmadian, Marwan Ahmed, Jay Alammar, Milad Alizadeh, Yazeed Alnumay, Sophia Althammer, Arkady Arkhangorodsky, Viraat Aryabumi, Dennis Aumiller, et al. Command-A: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*, 2025.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2 edition, 2006.

Imre Csiszár. Information-type measures of difference of probability distributions and indirect observation. *Studia Sci. Math. Hungarica*, 2:299–318, 1967.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Chenyu Huang, Peng Ye, Tao Chen, Tong He, Xiangyu Yue, and Wanli Ouyang. Emr-merging: Tuning-free high-performance model merging. *Advances in Neural Information Processing Systems*, 37:122741–122769, 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.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

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

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*, 2022.

Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 245–254. ACM, 1985. doi: 10.1145/325165.325242. URL https://dl.acm.org/doi/10.1145/325165.325242.

George Stoica, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=LEYUkvdUhq.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning (ICML)*, volume 162, pp. 23965–23998. PMLR, 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Zhongshen Zeng, Pengguang Chen, Shu Liu, Haiyun Jiang, and Jiaya Jia. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation. *arXiv preprint arXiv:2312.17080*, 2023.

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.

5 APPENDIX

A METHODOLOGY DETAILS

We consider a catalog of post-trained checkpoints $\mathcal{M} = \{m_1, \ldots, m_n\}$ derived from a shared pretrained base, and a set of tasks $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$. Evaluating a model $m \in \mathcal{M}$ on a task $t \in \mathcal{T}$ yields a scalar *utility* $U(m, t)$ computed from the task's evaluation metric on held-out data. Our goal is to construct a single composite model $\tilde{m}$ by merging checkpoints from $\mathcal{M}$ so that $U(\tilde{m}, t)$ is high for a target task t, or high on average across a set of tasks.

A.1 MERGE OPERATORS AND PLANS

Binary merge operators. Pairwise merges are performed using binary merge operators $o \in \mathcal{O}$. Given two checkpoints (m_a, m_b) and a mixing coefficient $\alpha \in [0, 1]$, the merged checkpoint is denoted by $o(m_a, m_b; \alpha)$. In all experiments we use equal-weight merges with $\alpha = 0.5$ and the operator set $\mathcal{O} = \{\text{LINEAR}, \text{SLERP}, \text{TIES}\}$. Formal operator definitions are given in Appendix B.

Merge plans. A *merge plan* specifies an ordered sequence of checkpoints and the merge operator used at each binary step. Given an ordered plan $\pi = (m_{i_1}, m_{i_2}, \ldots, m_{i_k})$, the merged model is obtained by sequentially applying binary merge operators:

$$\begin{cases} M_1 = m_{i_1}, \\ M_j = o_j(M_{j-1}, m_{i_j}; \alpha), \quad j = 2, \ldots, k, \end{cases}$$

where $o_j \in \mathcal{O}$ and the final merged model is $\tilde{m} = M_k$.

Many merge operators are not associative, including SLERP and TIES, so both the order in π and the per-step operator choices $\{o_j\}$ can affect the resulting model and its utility $U(\tilde{m}, t)$. The number of candidate plans grows rapidly with the number of checkpoints: for a fixed subset of k models there are $k!$ possible orders and $|\mathcal{O}|^{k-1}$ operator sequences, making merge-and-evaluate search infeasible at scale. SIMMERGE addresses this bottleneck by selecting operators and merge plans from pre-merge similarity signals.

A.2 SIMILARITY FEATURES

For each task $t \in \mathcal{T}$ we draw a small unlabeled probe set $\mathcal{P}_t$ from the input distribution of t. For every *ordered* pair of checkpoints (m_a, m_b) and task t, SIMMERGE constructs a feature vector $x(m_a, m_b, t) \in \mathbb{R}^m$ using $\mathcal{P}_t$ and the checkpoint weights. These metrics are inexpensive to compute and capture both functional behavior and structural alignment. (Probe sizes are reported in Appendix D.5.)

We first compute functional similarity signals on $\mathcal{P}_t$, including the KL divergence between predictive distributions, $D_{\mathrm{KL}}(p_a \,\|\, p_b)$, and cosine similarity between intermediate activations $h_a^{(\ell)}$ and $h_b^{(\ell)}$ at each layer ℓ, averaged over inputs and layers. We then compute weight-based measures that compare checkpoints directly in parameter space, including cosine similarity between flattened parameter vectors, Euclidean distance between parameters, and parameter norms. We additionally include cosine similarity of attention patterns as a separate feature channel.

Many metrics yield either a scalar or a short sequence over layers or modules. To obtain a fixed-dimensional representation, we summarize sequence-valued metrics using mean, median, and selected quantiles, and concatenate all summaries into $x(m_a, m_b, t)$. By default, we append a task encoding $c(t) \in \mathbb{R}^{d_c}$ and use

$$\tilde{x}(m_a, m_b, t) = x(m_a, m_b, t) \oplus c(t) \in \mathbb{R}^{m+d_c}$$

as input to all learned components. We also evaluate a task-agnostic variant that omits $c(t)$ and report the comparison in Appendix J.1. The improvement from task encoding is modest but consistent, so we use it by default. Full details of the similarity metrics and aggregation procedures are provided in Appendix C.

Computation cost. Probe-based metrics require only forward passes over small probe sets. We cache per-checkpoint probe outputs so pairwise comparisons reduce to inexpensive post-processing.

Weight-based metrics require a single pass over parameters, and the overall cost is far lower than running merge-and-evaluate searches.

A.3 PREDICTIVE MERGE SELECTION

Using the precomputed similarity features, SIMMERGE predicts merge operators and scores candidate merge plans without executing merges during selection. Throughout this appendix we fix the mixing coefficient to $\alpha = 0.5$ and omit it from the notation, writing $o(m_a, m_b)$ for $o(m_a, m_b; 0.5)$.

Pairwise utility prediction. SIMMERGE is built around a learned utility predictor that estimates the performance of a merge configuration for a given task. In the pairwise setting, for an ordered pair of checkpoints (m_a, m_b) and task $t \in \mathcal{T}$, we define a utility predictor

$$f_{\text{plan}} : \mathbb{R}^{m+d_c} \to \mathbb{R}^{|\mathcal{O}|},$$

which predicts the utility of merging m_a and m_b under each merge operator $o \in \mathcal{O}$. The predictor takes as input pre-merge similarity features $\tilde{x}(m_a, m_b, t)$ and is trained on offline two-way merges, where each operator is applied and the downstream utility $U(o(m_a, m_b), t)$ is observed. Training minimizes a regression loss toward the observed utilities for all operators:

$$\widehat{U}(o(m_a, m_b), t) = f_{\text{plan}}(\tilde{x}(m_a, m_b, t))_o.$$

At inference time, we select the operator with the highest predicted utility,

$$\hat{o} = \arg\max_{o \in \mathcal{O}} \widehat{U}(o(m_a, m_b), t),$$

replacing exhaustive merge-and-evaluate search.

Multi-way merge plan scoring. Multi-way merging generalizes this formulation by selecting an entire merge plan that maximizes predicted utility. A k-way merge plan $\pi = (m_{i_1}, \ldots, m_{i_k})$ is represented as an ordered sequence of binary merge steps. SIMMERGE estimates the utility of a plan by recursively predicting the utility of each intermediate merge, using the same plan scorer to select operators at each step.

Because intermediate merged models are not explicitly constructed during scoring, similarity features involving an intermediate model $M_{1:k-1}$ are approximated using features propagated from the original pairwise similarity table (see Appendix H). Different merge orders induce different sequences of predicted utilities, allowing SIMMERGE to capture the effects of both operator choice and merge order.

At test time, SIMMERGE enumerates or samples a small set of candidate merge plans, estimates their predicted utilities, and selects the plan with the highest predicted utility:

$$\hat{\pi} = \arg\max_{\pi} \widehat{U}(\pi, t).$$

The selected plan is then executed as a sequence of binary merges with the step-wise operators $\{\hat{o}_j\}$ chosen by the pairwise predictor.

A.4 BANDIT VIEW AND EVALUATION PROTOCOL

The offline selector performs well on the distribution of model pairs and tasks seen during training, but it must be retrained when new checkpoints, tasks, or operators are introduced. To support such scenarios without retraining the similarity feature computation, we cast operator selection as a contextual bandit and add a neural-linear bandit layer on top of the precomputed similarity features. As new model pairs and tasks appear, we compute their contexts from the same fixed feature pipeline and update the bandit online using the newly observed rewards.

Contextual bandit formulation. Each merge step, either a standalone two-way merge or a step within a multiway plan, defines a decision round. We observe a context vector $s \in \mathbb{R}^{m+d_c}$ derived from pre-merge similarity features. For pairwise merges, $s = \tilde{x}(m_a, m_b, t)$. For multiway merges, s is constructed analogously and includes propagated similarity features for intermediate merges. We then choose an action a from the operator set $\mathcal{O}$, apply the corresponding merge operator, and

observe a scalar reward $r(a)$ given by downstream evaluation. Rewards for unchosen operators are not observed. The objective is to learn a policy $\pi(s)$ that maximizes cumulative reward, or equivalently minimizing regret relative to an oracle that always selects the best operator for each context.

We adopt a neural-linear design. An MLP feature map g_ϕ transforms the context s into a representation $z(s) = g_\phi(s)$. We warm-start g_ϕ using the logged pairwise data described below, then keep it fixed during online adaptation. On top of $z(s)$ we fit a linear contextual bandit: for each operator $a \in \mathcal{O}$ we assume a linear reward model

$$\mathbb{E}[r(a) \mid z(s)] \approx w_a^\top z(s),$$

with an unknown parameter vector w_a and a Gaussian posterior $\mathcal{N}(\hat{w}_a, \Sigma_a)$ maintained via Bayesian linear regression. We consider both LinUCB and linear Thompson sampling (LinTS) (Abbasi-Yadkori et al., 2011; Agrawal & Goyal, 2013). Empirically, LinTS produces lower regret and higher downstream performance in our setting, and we therefore use it as our main bandit variant. Since we have a small number of operators and a low-dimensional representation, posterior updates can be done with rank-one updates in $O(d^2)$ time per round, where d is the dimension of $z(s)$.

Warm-start and online adaptation. The bandit is initialized using the fully-observed pairwise merge dataset from the offline setting. For each historical 2-way merge we observe the context s and the utilities $\$U(\mathsf{a}(m_a, m_b), t)$ for all operators $a \in \mathcal{O}$, which corresponds to full-information feedback. This warm-start phase anchors the reward model before any online interaction and reduces the amount of exploration required when new tasks or models are introduced.

After warm-start, we introduce a distribution shift by adding a checkpoint trained on a different task that did not appear in the logged data. For each new merge involving this checkpoint, we compute the context s from pre-merge similarity signals, query the bandit to select an operator a, execute the merge, and observe the resulting reward $r(a)$. Only the posterior corresponding to the chosen arm a is then updated. This is the partial-feedback regime where counterfactual rewards for unchosen operators are not observed.

B MERGE OPERATORS

This appendix specifies the merge operators used in the main text: linear interpolation (LINEAR), spherical linear interpolation (SLERP), and a TIES-style sign-consistent merge (TIES). All operators act on corresponding parameter tensors of two models m_a and m_b with flattened parameters $\theta_1 = \theta(m_a), \theta_2 = \theta(m_b) \in \mathbb{R}^d$. Unless stated otherwise, we use a fixed mixing coefficient $\alpha = 0.5$ in the experiments.

B.1 LINEAR INTERPOLATION

Linear interpolation (LINEAR) combines parameters by a convex combination,

$$M_{\text{Lin}}(\theta_1, \theta_2; \alpha) = (1 - \alpha)\theta_1 + \alpha\theta_2.$$

In practice we apply this operation layerwise on each parameter tensor such as attention and MLP weights and biases. We do not apply any additional rescaling beyond the convex weights.

B.2 SPHERICAL LINEAR INTERPOLATION (SLERP)

Spherical linear interpolation (SLERP) (Shoemake, 1985) interpolates on the unit sphere in parameter space, preserving the norms of the inputs. For each layer we normalize

$$\hat{\theta}_i = \frac{\theta_i}{\|\theta_i\|_2}, \quad i \in \{1, 2\},$$

compute the angle $\varphi = \arccos\langle \hat{\theta}_1, \hat{\theta}_2 \rangle$ and form the spherical interpolation

$$\tilde{\theta}_{\text{unit}} = \frac{\sin((1-\alpha)\varphi)}{\sin\varphi}\,\hat{\theta}_1 + \frac{\sin(\alpha\varphi)}{\sin\varphi}\,\hat{\theta}_2.$$

We then rescale $\tilde{\theta}_{\text{unit}}$ to match the average input norm,

$$\tilde{\theta} = \frac{\|\theta_1\|_2 + \|\theta_2\|_2}{2} \cdot \frac{\tilde{\theta}_{\text{unit}}}{\|\tilde{\theta}_{\text{unit}}\|_2}.$$

The normalization and rescaling are applied per layer, using the layerwise parameter tensors. This follows the standard SLERP construction and keeps parameter magnitudes comparable across merges.

B.3 TIES-Style Sign-Consistent Merge

The TIES-merging (TRIM, ELECT SIGN and MERGE) (Yadav et al., 2023) is a sign-consistent rule that suppresses conflicting updates while interpolating non-conflicting entries. We abstract it as

$$M_{\text{TIES}}(\theta_1, \theta_2; \alpha) = T_\tau(\theta_1, \theta_2; \alpha),$$

where $\tau \geq 0$ is a threshold hyperparameter.

Let $\theta_1[j], \theta_2[j]$ denote the j-th coordinate of the two parameter vectors. The operator T_τ is defined coordinate-wise:

$$T_\tau(\theta_1, \theta_2; \alpha)[j] = \begin{cases} \alpha\,\theta_1[j] + (1-\alpha)\,\theta_2[j], & \text{if } \theta_1[j]\theta_2[j] > 0 \text{ and } \max(|\theta_1[j]|, |\theta_2[j]|) \geq \tau, \\ \theta_1[j], & \text{if } \theta_1[j]\theta_2[j] \leq 0 \text{ and } |\theta_1[j]| \geq |\theta_2[j]| \text{ and } |\theta_1[j]| \geq \tau, \\ \theta_2[j], & \text{if } \theta_1[j]\theta_2[j] \leq 0 \text{ and } |\theta_2[j]| > |\theta_1[j]| \text{ and } |\theta_2[j]| \geq \tau, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, coordinates with aligned sign and sufficient magnitude are interpolated linearly, while coordinates with sign conflicts are resolved by selecting the larger-magnitude entry, and small-magnitude coordinates are pruned. In our implementation, T_τ is applied layerwise to each parameter tensor, and the same threshold τ is used across layers. The value of τ is treated as a hyperparameter and tuned on the pairwise validation split.

C Similarity Metrics and Feature Construction

This appendix specifies the similarity metrics and the construction of the feature vectors $x(m_a, m_b, t)$ and $\tilde{x}(m_a, m_b, t)$ used by SimMerge.

C.1 Probe Data and Notation

For each task $t \in \mathcal{T}$ we draw an unlabeled probe set $\mathcal{P}_t = \{x_1, \ldots, x_{N_t}\}$ from the input distribution of t. No labels are used in any similarity metric.

For a model m and a prompt x, let $z_m(x)$ denote the logits produced under teacher forcing. At each decoding position j (token index), we write

$$p_m(\cdot \mid x, j) = \text{softmax}(z_m(x)_j)$$

for the next-token predictive distribution over the vocabulary. For transformer activations, let $h_m^{(\ell)}(x) \in \mathbb{R}^{T \times d_\ell}$ denote the post-residual hidden states at layer ℓ, where T is the sequence length. When needed, we flatten $h_m^{(\ell)}(x)$ across sequence positions into a single vector in $\mathbb{R}^{Td_\ell}$. Equivalently, we concatenate token representations.

For attention patterns, let $A_m^{(\ell,h)}(x) \in \mathbb{R}^{T \times T}$ denote the attention weight matrix (softmax over keys) at layer ℓ and head h for prompt x.

For each model m we denote its flattened parameter vector by $\theta(m) \in \mathbb{R}^d$. For brevity, we write $\theta_a = \theta(m_a)$ and $\theta_b = \theta(m_b)$.

C.2 Data-Based Metrics

Data-based metrics compare model behavior on the probe set $\mathcal{P}_t$.

KL divergence between predictive distributions. For an ordered pair (m_a, m_b), prompt $x \in \mathcal{P}_t$, and position j, the pointwise KL divergence is

$$D_{\mathrm{KL}}\big(p_a(\cdot \mid x, j) \,\|\, p_b(\cdot \mid x, j)\big) = \sum_i p_a(i \mid x, j) \, \log \frac{p_a(i \mid x, j)}{p_b(i \mid x, j)}.$$

We average over positions and prompts to obtain

$$\mathrm{KL}_{\text{mean}}(m_a, m_b, t) = \frac{1}{N_t} \sum_{x \in \mathcal{P}_t} \frac{1}{|J(x)|} \sum_{j \in J(x)} D_{\mathrm{KL}}\big(p_a(\cdot \mid x, j) \,\|\, p_b(\cdot \mid x, j)\big),$$

where $J(x)$ is the set of teacher-forced positions used for evaluation. We additionally record robust summary statistics over prompts (median and empirical quantiles, such as 25th/75th/90th percentiles). KL is computed in log-space with standard numerical stabilization.

Activation cosine similarity. For each layer ℓ and prompt $x \in \mathcal{P}_t$, let $\mathrm{vec}(h_m^{(\ell)}(x)) \in \mathbb{R}^{Td_\ell}$ denote the flattened hidden states. We define

$$\cos_h^{(\ell)}(m_a, m_b, x) = \frac{\Big\langle \mathrm{vec}(h_a^{(\ell)}(x)),\, \mathrm{vec}(h_b^{(\ell)}(x)) \Big\rangle}{\Big\| \mathrm{vec}(h_a^{(\ell)}(x)) \Big\|_2 \, \Big\| \mathrm{vec}(h_b^{(\ell)}(x)) \Big\|_2}, \qquad \cos_h^{(\ell)}(m_a, m_b, t) = \frac{1}{N_t} \sum_{x \in \mathcal{P}_t} \cos_h^{(\ell)}(m_a, m_b, x).$$

We keep either the full per-layer sequence $\{\cos_h^{(\ell)}\}_{\ell=1}^{L}$ for later summarization or aggregate across layers immediately.

Attention-pattern cosine similarity. For each layer ℓ, head h, and prompt $x \in \mathcal{P}_t$, we flatten attention matrices and compute

$$\cos_A^{(\ell,h)}(m_a, m_b, x) = \frac{\Big\langle \mathrm{vec}(A_a^{(\ell,h)}(x)),\, \mathrm{vec}(A_b^{(\ell,h)}(x)) \Big\rangle}{\Big\| \mathrm{vec}(A_a^{(\ell,h)}(x)) \Big\|_2 \, \Big\| \mathrm{vec}(A_b^{(\ell,h)}(x)) \Big\|_2}.$$

We summarize these values across prompts, heads, and layers using the same robust statistics as above (mean/median/quantiles), yielding a compact set of attention-similarity features.

C.3 Weight-Based Metrics

Weight-based metrics compare parameters directly and do not depend on $\mathcal{P}_t$.

Weight cosine similarity. For flattened parameter vectors $\theta_a, \theta_b \in \mathbb{R}^d$,

$$\cos_W(m_a, m_b) = \frac{\langle \theta_a, \theta_b \rangle}{\|\theta_a\|_2 \, \|\theta_b\|_2}.$$

We optionally compute layerwise or module-restricted variants by restricting θ_a, θ_b to parameters of a given transformer block or to attention/MLP submodules.

Weight ℓ_2 distance.

$$d_W(m_a, m_b) = \|\theta_a - \theta_b\|_2,$$

again optionally computed per-layer or per-module by restriction to parameter subsets.

Weight norms. We record $\|\theta_a\|_2$ and $\|\theta_b\|_2$ (and optionally their layerwise/modulewise norms) to capture global scale differences that can interact with merge behavior.

C.4 Feature Vector Construction

Each metric yields either a scalar or a short sequence indexed by layers and, for attention, optionally heads. To obtain a fixed-dimensional representation, sequence-valued metrics are summarized using

robust statistics such as the mean, median, and selected quantiles, and all summaries are concatenated into a single feature vector

$$x(m_a, m_b, t) \in \mathbb{R}^m.$$

By default, we append an explicit task encoding $c(t) \in \mathbb{R}^{d_c}$ and use

$$\tilde{x}(m_a, m_b, t) = x(m_a, m_b, t) \oplus c(t) \in \mathbb{R}^{m+d_c}$$

as the input to all learned components. We also evaluate a task-agnostic variant that omits $c(t)$; the comparison is reported in Appendix J.1. The improvement from the task encoding is modest but consistent, so we keep it enabled in the main experiments.

D Experiment Setup Details

We evaluate SimMerge on four domains: code generation, mathematical reasoning, multilingual understanding and RAG. All experts share a common pretrained base (Command-A 7B or 111B (Cohere et al., 2025)), ensuring differences arise from fine-tuning and merging rather than capacity.

For a target task t and a set of k models, we compare three fixed merge operators – Linear, Slerp, and TIES against our learned selector, SimMerge. Fixed operators apply the same rule at every merge step, using a predetermined order when $k > 2$. In contrast, SimMerge chooses an operator (and, for $k > 2$, a merge order) based on pre-merge similarity features, allowing performance gains without exhaustive merge-and-evaluate search.

D.1 Experts and Auxiliary Models

For each domain t, we fine-tune a shared base model on task-specific data and designate the best resulting checkpoint as the *task expert* m_t^{exp}. When evaluating task t, any model not fine-tuned on t is treated as an *auxiliary* model; thus, experts from other domains act as auxiliaries for t. In all merge configurations, we combine one task expert with one or more auxiliary models. We report the standalone performance of task experts and auxiliary models as reference upper/lower bound baselines for task t, noting the best-performing expert or auxiliary may vary across metrics or configurations.

Across offline experiments at 7B, we use 85 domain-specialized checkpoints: 23 code, 24 math, 24 multilingual, and 15 RAG. For online bandit experiments, we add 15 instruction-tuned 7B checkpoints, yielding 100 distinct 7B models in total. At the 111B, we evaluate on 18 additional task-specific checkpoints (5 code, 5 math, 4 multilingual, and 4 RAG).

D.2 Merge Configurations

Pairwise merges. We study pairwise merges between a task expert and a single auxiliary model, which both generate supervised training data and provide a controlled setting to test whether similarity features can predict the best merge operator. The selector is trained on expert-auxiliary pairs and evaluated on a disjoint held-out set (Appendix D.6); classifier results are in Appendix J.1.

Multi-way merges. We then evaluate 3-way and 4-way merges that combine one task expert with auxiliary models from different domains. We compare fixed operators (Linear, Slerp, TIES) against SimMerge. For $k > 2$, where merge order affects performance, we compare random merge orders with the order selected by SimMerge.

Scaling and online evaluation. To test transfer across model scales, we repeat multiway merging experiments at the 111B scale using selectors trained at 7B, without retraining. We additionally evaluate a bandit-based selector on a separate set of merge configurations that includes instruct checkpoints and is disjoint from the offline training data.

D.3 Evaluations

We evaluate expert and merged models on held-out evaluation sets covering our four core domains, plus an instruction-following suite used in the bandit experiments.

For each task t, we evaluate on a small collection of standard benchmarks: . Math is evaluated on MATH (Hendrycks et al., 2021) and GSM8K (Zeng et al., 2023); code generation on HUMANEVAL (Chen, 2021) and MBPP+ (Liu et al., 2023); multilingual understanding on MGSM (Shi et al., 2022) and an internal multilingual QA suite; retrieval-augmented generation (RAG) on TAUBENCH (Yao et al., 2025) and BFCL (Patil et al., 2025); and instruction-following on IFEVAL (Zhou et al., 2023) (details in Appendix I). Metrics follow standard task conventions; we repeat each evaluation three times with different seeds and report the mean. For each model and task t, we report a single task score given by the unweighted mean across t's benchmarks.

D.4 METRICS

We evaluate merge quality using both absolute task performance and normalized percentage change relative to two natural baselines: (i) the task expert and (ii) the auxiliary model(s) involved in a merge. We denote these normalized changes by Δ_{expert} and Δ_{aux}, respectively. Reporting normalized change relative to these baselines enables fair comparison across tasks with different difficulty and score scales.

To summarize how effectively a merge recovers task-specific capability, we additionally report *Gap-Closed*, which measures the fraction of the performance gap between the auxiliary baseline and the task expert that is recovered by the merged model (0% corresponds to auxiliary performance and 100% to expert performance). Values above 100% indicate exceeding the expert.

All metrics are macro-averaged across tasks. Formal definitions and normalization details are provided in Appendix E.

D.5 SELECTOR ARCHITECTURE AND TRAINING DETAILS

Offline classifier. We train a lightweight MLP that takes the similarity features $\tilde{x}(m_a, m_b, t)$ as input and outputs the predicted utility for the merge on task t. We use a two-layer network with ReLU activations and Adam optimization. We tune basic hyperparameters (hidden width, learning rate, batch size, dropout) on a held-out validation subset of the pairwise merge dataset and use early stopping on validation accuracy.

Bandit-based selector. We adopt the neural-linear contextual bandit described in Section A.4. We instantiate the bandit feature map as a separate MLP encoder g_ϕ, and take $z(\tilde{x})$ to be its final hidden-layer representation. We warm-start this encoder and the per-operator Bayesian linear models using the training pairwise merge data described in Section A.4, and then keep the encoder fixed during online adaptation. On top of $z(\tilde{x})$, we maintain a Bayesian linear model for each operator and use linear Thompson sampling (Abbasi-Yadkori et al., 2011; Agrawal & Goyal, 2013) as our primary bandit policy, tuning its exploration and regularization hyperparameters on a validation split of the logged data. This policy is then used to select operators when new tasks or models, for instance, a new set of instruct checkpoints are introduced, without retraining the encoder.

D.6 MERGE CONFIGURATIONS

Pairwise merges. We construct a dataset of pairwise merges between task experts and auxiliary models. These experiments provide supervised training data for the offline selector and allow us to test whether similarity features are sufficient to predict the best-performing merge operator.

The pairwise training set contains 240 distinct expert–auxiliary merges. Among these, the best-performing operator is Linear in 96 cases, SLERP in 88 cases, and TIES in 56 cases, indicating that no single operator dominates. For offline evaluation, we construct a held-out test set of 60 additional expert–auxiliary pairs. For each test pair, we evaluate all three operators and record the best-performing method, which is used as the label for assessing selector accuracy. Detailed classifier results are reported in Appendix J.1.

All evaluation splits are disjoint at the pair level: no expert–auxiliary pair used for selector training appears in the held-out test set or in any multiway merge configuration.

Multiway merges. We next consider 3-way and 4-way merges. Each configuration contains at most one expert per task, combining one task expert with auxiliary models from different domains. We apply the same set of merge operators (Linear, SLERP, TIES) as well as SIMMERGE.

For $k > 2$, merge order affects the final model. We therefore evaluate both random-order baselines and the merge order proposed by the multiway selector. In total, we construct 145 distinct 3-way merge configurations and 130 distinct 4-way configurations. Each configuration is evaluated with all fixed operators and with SIMMERGE, yielding a large corpus of merge outcomes used to assess multiway planning quality.

Scaling to a larger model. To evaluate transfer across scales, we repeat the 3-way merge experiments using a larger 111B-parameter base model. We reuse the same similarity features and selector architectures trained on the 7B models, without retraining. At the 111B scale, we evaluate 100 distinct 3-way merge configurations.

Online evaluation merges. For the linear bandit experiments, we construct a separate evaluation set of merge configurations that includes additional *instruct* checkpoints. This bandit evaluation set contains 60 merge instances across 2-way, 3-way, and 4-way merges and is disjoint from the offline pairwise training data at the ordered-pair level.

E EVALUATION METRIC DEFINITIONS

We evaluate merge quality using both absolute task performance and normalized percentage change relative to natural baselines. Normalization is necessary to enable fair comparison across tasks with different metrics and score ranges.

E.1 BASELINES

For a task t, we define the *expert baseline* score as

$$s_{\text{expert}}(t) := \text{score}(\text{expert}_t, t),$$

where expert_t is the model fine-tuned specifically for task t.

We define the *auxiliary baseline* score $s_{\text{aux}}(t)$ differently depending on the merge configuration. For pairwise merges,

$$s_{\text{aux}}(t) := \text{score}(\text{aux}_t, t),$$

where aux_t is the single auxiliary model paired with the expert. For multiway merges, we define

$$s_{\text{aux}}(t) := \frac{1}{|\mathcal{A}_t|} \sum_{m' \in \mathcal{A}_t} \text{score}(m', t),$$

where $\mathcal{A}_t$ is the set of auxiliary models included in the merge configuration.

E.2 NORMALIZED PERCENTAGE CHANGE

For a merged model m evaluated on task t, we compute normalized percentage change relative to each baseline:

$$\Delta_{\text{expert}}(m, t) = 100 \cdot \frac{\text{score}(m, t) - s_{\text{expert}}(t)}{s_{\text{expert}}(t)},$$
$$\Delta_{\text{aux}}(m, t) = 100 \cdot \frac{\text{score}(m, t) - s_{\text{aux}}(t)}{s_{\text{aux}}(t)}.$$

These metrics quantify relative gains over the expert and auxiliary baselines, respectively, and account for differences in scale across tasks.

E.3 GAP CLOSED

In addition, we report the *gap closed* metric, which measures how much of the performance gap between the auxiliary baseline and the task expert is recovered by the merged model:

$$\mathrm{GapClosed}(m,t) = 100 \times \frac{\mathrm{score}(m,t) - s_{\text{aux}}(t)}{s_{\text{expert}}(t) - s_{\text{aux}}(t)}.$$

Under this normalization, 0% corresponds to auxiliary performance and 100% corresponds to expert performance. Values above 100% indicate that the merged model exceeds the expert, while negative values indicate performance below the auxiliary baseline.

E.4 AGGREGATION

All normalized metrics are aggregated across tasks by macro-averaging over $t \in \mathcal{T}$. Absolute performance is also reported by macro-averaging the raw task scores to anchor normalized percentage change to the original evaluation scales.

F MERGE CONFIGURATION DETAILS

Pairwise merges. We construct a dataset of pairwise merges between task experts and auxiliary models. These experiments provide supervised training data for the offline selector and allow us to test whether similarity features are sufficient to predict the best-performing merge operator.

The pairwise training set contains 240 distinct expert–auxiliary merges. Among these, the best-performing operator is Linear in 96 cases, SLERP in 88 cases, and TIES in 56 cases, indicating that no single operator dominates. For offline evaluation, we construct a held-out test set of 60 additional expert–auxiliary pairs. For each test pair, we evaluate all three operators and record the best-performing method, which is used as the label for assessing selector accuracy. Detailed classifier results are reported in Appendix J.1.

All evaluation splits are disjoint at the pair level: no expert–auxiliary pair used for selector training appears in the held-out test set or in any multiway merge configuration.

Multiway merges. We next consider 3-way and 4-way merges. Each configuration contains at most one expert per task, combining one task expert with auxiliary models from different domains. We apply the same set of merge operators (Linear, SLERP, TIES) as well as SIMMERGE.

For $k > 2$, merge order affects the final model. We therefore evaluate both random-order baselines and the merge order proposed by the multiway selector. In total, we construct 145 distinct 3-way merge configurations and 130 distinct 4-way configurations. Each configuration is evaluated with all fixed operators and with SIMMERGE, yielding a large corpus of merge outcomes used to assess multiway planning quality.

Scaling to a larger model. To evaluate transfer across scales, we repeat the 3-way merge experiments using a larger 111B-parameter base model. We reuse the same similarity features and selector architectures trained on the 7B models, without retraining. At the 111B scale, we evaluate 100 distinct 3-way merge configurations.

Online evaluation merges. For the linear bandit experiments, we construct a separate evaluation set of merge configurations that includes additional *instruct* checkpoints. This bandit evaluation set contains 60 merge instances across 2-way, 3-way, and 4-way merges and is disjoint from the offline pairwise training data at the ordered-pair level.

G WHY PAIRWISE TRAINING CAN TRANSFER TO MULTI-WAY PLANNING

This section provides intuition for why a scorer trained using pairwise-derived signals can be effective for ranking multi-way merge plans. For a k-way plan $\pi = (m_{i_1} \to \cdots \to m_{i_k})$ on task t, SIMMERGE represents the plan by concatenating step-wise feature blocks,

$$X(\pi, t) = \big[x(m_{i_1}, m_{i_2}, t),\, \ldots,\, x(m_{i_{k-1}}, m_{i_k}, t)\big] \oplus c(t).$$

This construction is motivated by the observation that non-associativity makes the *local* interaction between consecutive merge steps consequential: changing the order changes which pairs interact early versus late, and these interactions are reflected in the corresponding pairwise similarity regimes.

A sufficient condition for this representation to be useful is that the utility of executing a plan, $U(\pi, t)$, depends smoothly on (or can be well-approximated by) a low-order function of these step-wise interactions. For example, if

$$U(\pi, t) \approx \sum_{s=1}^{k-1} \psi\big(x(m_{i_s}, m_{i_{s+1}}, t),\, t\big)$$

for some unknown function ψ, then a learned plan scorer can estimate $U(\pi, t)$ from $X(\pi, t)$ by aggregating step-wise contributions. More generally, if $U(\pi, t) = F(x_1, \ldots, x_{k-1}, c(t))$ is a sufficiently smooth function of the step blocks x_s, then a first-order expansion around typical interaction regimes yields an approximately additive dependence on the concatenated features. This motivates learning f_{plan} on plan representations built from the same pairwise feature blocks used for operator selection.

H Propagation of Similarity Metrics to Multi-Way Plans

This appendix details how we construct approximate similarity features for intermediate steps when scoring multi-way merge plans, without explicitly constructing and evaluating intermediate merged parameters for each candidate plan.

A multi-way plan involves intermediate merged checkpoints. To score candidate plans efficiently, we use a *proxy* representation for an intermediate step formed by merging a and b with coefficient α. In the main experiments we fix $\alpha = \frac{1}{2}$ and treat the intermediate as an equal-weight combination for the purpose of constructing features. For data-based quantities such as KL, where the intermediate model's predictive distribution is not available without executing the merge, we use mixture-inspired proxy estimates derived from standard inequalities; these values serve as inexpensive features rather than exact measurements of the true intermediate model.

Orientation is explicit because some metrics are asymmetric (notably KL). We write

$$(a{+}b,\, c): \quad G_L = (1{-}\alpha)P_a{+}\alpha P_b,\; G_R = P_c, \qquad \text{and} \qquad (c,\, a{+}b): \quad G_L = P_c,\; G_R = (1{-}\alpha)Q_a{+}\alpha Q_b,$$

where $P_\cdot$ and $Q_\cdot$ denote predictive distributions on the probe set or distributional proxies derived from logits.

H.1 KL Divergence Proxies

By the log-sum inequality and the joint convexity of KL (more generally, f-divergences) in each argument (Csiszár, 1967; Cover & Thomas, 2006),

$$\mathrm{KL}\big((1-\alpha)P_a + \alpha P_b \,\|\, P_c\big) \le (1-\alpha)\,\mathrm{KL}(P_a \,\|\, P_c) + \alpha\,\mathrm{KL}(P_b \,\|\, P_c), \tag{1}$$

$$\mathrm{KL}\big(P_c \,\|\, (1-\alpha)Q_a + \alpha Q_b\big) \le (1-\alpha)\,\mathrm{KL}(P_c \,\|\, Q_a) + \alpha\,\mathrm{KL}(P_c \,\|\, Q_b). \tag{2}$$

More generally, when both arguments are mixtures,

$$G_L := \sum_i w_i P_i, \qquad G_R := \sum_j v_j Q_j,$$

with $w_i, v_j \ge 0$ and $\sum_i w_i = \sum_j v_j = 1$, we have

$$\mathrm{KL}(G_L \,\|\, G_R) \le \sum_{i,j} w_i v_j\,\mathrm{KL}(P_i \,\|\, Q_j).$$

In practice, we use the right-hand sides of equation 1 and equation 2 as propagated *proxy* values.

H.2 ℓ_2 Parameter Distance Proxies

Let $\theta_a, \theta_b, \theta_c \in \mathbb{R}^d$ be flattened parameter vectors and $L_2(\theta, \theta') = \|\theta - \theta'\|_2$. By the triangle inequality and positive homogeneity of norms (Boyd & Vandenberghe, 2004),

$$\left\|(1-\alpha)\theta_a + \alpha\theta_b - \theta_c\right\|_2 \le (1-\alpha)\,\|\theta_a - \theta_c\|_2 + \alpha\,\|\theta_b - \theta_c\|_2, \tag{3}$$

$$\left\|\theta_c - \big((1-\alpha)\theta_a + \alpha\theta_b\big)\right\|_2 \le (1-\alpha)\,\|\theta_c - \theta_a\|_2 + \alpha\,\|\theta_c - \theta_b\|_2. \tag{4}$$

We use the right-hand sides as propagated proxy values and mark them *proxy upper*.

H.3 Cosine Similarity Proxies (Weights or Attention Patterns)

Define $\cos(u, v) = \frac{\langle u, v\rangle}{\|u\|_2\,\|v\|_2}$. Because the denominator is nonlinear in mixtures, an exact cosine with an intermediate proxy would require dot products and norms that are typically not logged for every possible intermediate. We therefore use a simple, stable proxy:

$$\cos\big((1-\alpha)u_a + \alpha u_b,\ u_c\big) \approx (1-\alpha)\,\cos(u_a, u_c) + \alpha\,\cos(u_b, u_c), \tag{5}$$

and similarly for $\cos\big(u_c,\ (1-\alpha)u_a + \alpha u_b\big)$. We clip the resulting values to $[-1, 1]$. This rule is used both for weight-vector cosines (with $u_\cdot = \theta_\cdot$) and for attention-pattern cosines (with $u_\cdot = \mathrm{vec}(A_\cdot^{(\ell,h)}(x))$ after summarization).

These propagated proxy values are aggregated using the same robust statistics as in Appendix C and inserted into the plan representation $X(\pi, t)$ whenever a similarity involving an intermediate step is required. This allows SimMerge to score candidate multi-step merge sequences using a precomputed pairwise similarity table, without recomputing similarities for every hypothetical intermediate merge.

I Evaluation Benchmarks and Metrics

We provide additional details on the benchmarks and evaluation metrics used for each task domain.

Math reasoning. We evaluate mathematical reasoning on *MATH* (Hendrycks et al., 2021) and *GSM8K* (Cobbe et al., 2021). Both benchmarks consist of grade-school to competition-level math problems that require multi-step reasoning and symbolic manipulation. Models are evaluated using *exact match* accuracy, where a prediction is considered correct only if the final answer exactly matches the reference solution. For GSM8K, answers are normalized following standard evaluation protocols to account for formatting differences.

Multilingual question answering. Multilingual performance is measured using an internal multilingual QA suite together with *MGSM* (Shi et al., 2022), which extends GSM-style math reasoning to multiple languages. MGSM evaluates cross-lingual generalization and reasoning robustness. Performance is reported using accuracy and win-rate metrics, where win-rate measures the fraction of examples on which a model's answer is preferred over a baseline under automatic or human evaluation, depending on the benchmark. These metrics capture both correctness and relative answer quality across languages.

Code generation. Code generation is evaluated on *HumanEval_Python* (Chen, 2021) and *MBPP+* (Liu et al., 2023). Both benchmarks assess functional correctness of generated programs against unit tests. We report *pass@1*, which measures the probability that the first generated solution passes all test cases. This metric reflects single-sample code generation quality and is standard in code evaluation.

Retrieval-augmented generation (RAG). RAG performance is evaluated on *TauBench* (Yao et al., 2025) and *BFCL* (Patil et al., 2025), which test a model's ability to integrate retrieved evidence into accurate responses. We report accuracy and F1 score, depending on the benchmark, following their official evaluation protocols. These metrics assess both answer correctness and overlap with reference responses, capturing retrieval grounding quality.

Instruction following. For instruction-following experiments used in the bandit setting, we evaluate on *IFEval* (Zhou et al., 2023). IFEval measures a model's ability to follow explicit instructions and

constraints. Performance is reported using the benchmark's standard instruction-compliance score, which aggregates binary success indicators across multiple instruction types.

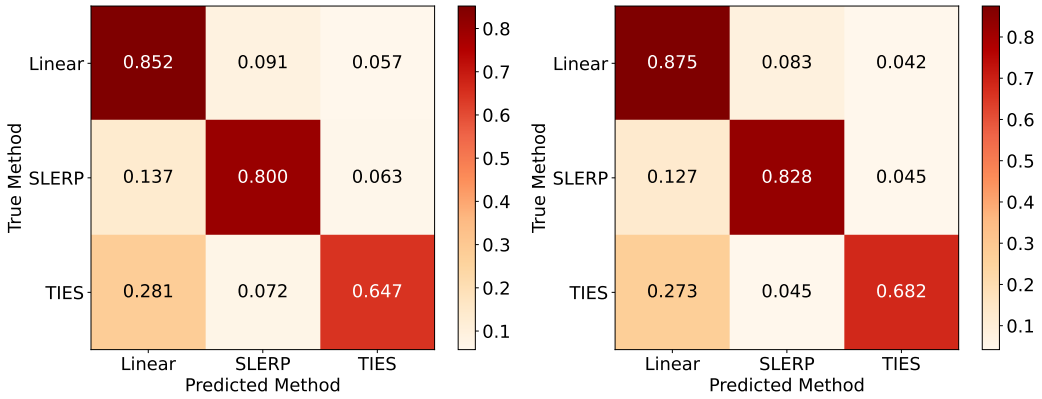
All evaluations are run three times with different random seeds, and we report the mean score. This reduces variance due to stochastic decoding and ensures stable comparisons across merge methods.

J Additional Results

J.1 Classifier Accuracy and Task-Encoding Ablation

To quantify predictive accuracy, we report confusion matrices for the offline selector on the held-out pairwise test set of 60 merges. We compare our default task-conditioned representation, which appends a task encoding $c(t)$ to the similarity features, against a task-agnostic variant that omits $c(t)$.

Figure 6 shows that task conditioning yields a small but consistent improvement across all classes. With the task encoding, the selector correctly identifies Linear in 87.5% of cases, SLERP in 82.8%, and TIES in 68.2%. Without the task encoding, accuracy drops to 85.2% for Linear, 80.0% for SLERP, and 64.7% for TIES. Across both settings, most errors occur when the true operator is TIES, reflecting that TIES occupies a narrower regime and is easier to confuse with Linear or SLERP. Overall, these results support that similarity features capture the relationships that drive operator preference, and that a lightweight task encoding provides an additional, modest gain.



(a) Without task encoding

(b) With task encoding

Figure 6: Confusion matrices of the offline selector on the held-out pairwise test set. Appending a task encoding improves per-class accuracy from 85.2% to 87.5% for Linear, from 80.0% to 82.8% for SLERP, and from 64.7% to 68.2% for TIES.

J.2 Detailed Per-Task Results Across Merge Sizes

Tables 1–3 provide detailed per-task summaries for pairwise, three-way, and four-way merges of 7B checkpoints. For each task and merge method, we report the mean task performance, absolute differences from the expert and auxiliary baselines, and the corresponding relative changes. Highlighting the best fixed operator per task makes explicit how the strongest baseline varies across settings, while bolding SimMerge emphasizes its consistent advantage across tasks and merge sizes.

In the pairwise setting (Table 1), the best fixed operator differs substantially across tasks: Linear performs best on Code and Multilingual, TIES on Math, and SLERP on RAG. This variability confirms that no single merge operator dominates even in the simplest two-model regime. Across all tasks, SimMerge consistently achieves higher mean performance than the best fixed operator, simultaneously reducing degradation relative to the expert and improving more over the auxiliary baseline. These results establish that similarity features are predictive of operator choice and motivate learning instance-specific merge decisions.

Table 1: Pairwise (2-way) merges: per-task performance summary. Mean is the task-level average score. Diff.Exp and Diff.Aux denote absolute differences from the expert and auxiliary baselines, %Exp and %Aux are the corresponding relative changes. The best fixed operator (among Linear, SLERP, and TIES) is highlighted in blue for each task, SIMMERGE is bold.

Task	Method	Mean	Diff.Exp	Diff.Aux	%Exp	%Aux
Code	Linear	0.60	-0.03	0.07	-4.70	12.31
	SLERP	0.59	-0.04	0.05	-7.07	9.52
	TIES	0.56	-0.07	0.02	-11.66	4.11
	SIMMERGE	**0.62**	**-0.02**	**0.08**	**-2.40**	**15.02**
Math	Linear	0.68	-0.07	0.11	-9.15	19.20
	SLERP	0.70	-0.05	0.13	-6.61	22.53
	TIES	0.72	-0.03	0.15	-3.98	25.98
	SIMMERGE	**0.74**	**-0.01**	**0.16**	**-1.90**	**28.80**
Multilingual	Linear	0.44	-0.16	0.09	-26.40	26.77
	SLERP	0.38	-0.22	0.03	-36.55	9.29
	TIES	0.32	-0.28	-0.03	-46.24	-7.40
	SIMMERGE	**0.46**	**-0.13**	**0.12**	**-22.27**	**33.89**
RAG	Linear	0.21	-0.20	-0.00	-49.46	-0.38
	SLERP	0.25	-0.15	0.05	-37.65	22.88
	TIES	0.22	-0.19	0.01	-46.09	6.26
	SIMMERGE	**0.28**	**-0.13**	**0.07**	**-32.11**	**33.81**

Table 2: Three-way (k=3) merges: per-task performance summary using the same metrics as Table 1. The best fixed operator (among Linear, SLERP, and TIES) is highlighted in blue for each task; SIMMERGE is bold.

Task	Method	Mean	Diff.Exp	Diff.Aux	%Exp	%Aux
Code	Linear	0.60	-0.03	0.06	-5.39	11.77
	SLERP	0.57	-0.06	0.04	-9.18	7.29
	TIES	0.51	-0.12	-0.02	-18.68	-3.94
	SIMMERGE	**0.61**	**-0.02**	**0.08**	**-2.90**	**14.71**
Math	Linear	0.64	-0.09	0.04	-12.48	7.33
	SLERP	0.70	-0.03	0.11	-3.89	17.86
	TIES	0.71	-0.02	0.11	-2.76	19.25
	SIMMERGE	**0.73**	**-0.00**	**0.13**	**-0.02**	**22.61**
Multilingual	Linear	0.42	-0.15	0.08	-25.79	22.82
	SLERP	0.36	-0.21	0.02	-36.29	5.45
	TIES	0.34	-0.23	-0.00	-39.76	-0.30
	SIMMERGE	**0.44**	**-0.13**	**0.10**	**-22.20**	**28.77**
RAG	Linear	0.20	-0.20	-0.03	-49.76	-13.81
	SLERP	0.22	-0.18	-0.01	-45.38	-6.30
	TIES	0.21	-0.18	-0.02	-46.09	-7.52
	SIMMERGE	**0.24**	**-0.15**	**0.01**	**-38.88**	**4.85**

For three-way merges (Table 2), overall performance decreases relative to the pairwise setting, reflecting the increased difficulty of composing multiple models. Nevertheless, the same qualitative patterns persist: the identity of the strongest fixed operator remains task-dependent, and fixed baselines occasionally fail to improve over auxiliaries, particularly on RAG and Multilingual. In contrast, SIMMERGE consistently yields the highest mean performance across all tasks, incurring the smallest expert degradation while maintaining positive gains over auxiliary models in every domain.

Four-way merges (Table 3) further amplify the shortcomings of fixed operator choices. For several tasks, all fixed baselines incur large expert degradation and, in some cases, negative gains relative to auxiliaries, indicating harmful merges. Despite this increased complexity, SIMMERGE consistently remains the top-performing method across tasks, often matching or exceeding the best fixed operator while substantially reducing expert degradation. These results demonstrate that similarity-driven operator selection becomes increasingly important as the number of merged models grows.

Table 3: Four-way (k=4) merges: per-task performance summary using the same metrics as Table 1. The best fixed operator (among Linear, SLERP, and TIES) is highlighted in blue for each task; SIMMERGE is bold.

Task	Method	Mean	Diff.Exp	Diff.Aux	%Exp	%Aux
Code	Linear	0.53	-0.08	0.01	-13.30	1.10
	SLERP	0.51	-0.10	-0.02	-16.89	-3.09
	TIES	0.53	-0.08	0.01	-13.00	1.44
	SIMMERGE	**0.59**	**-0.02**	**0.07**	**-3.36**	**12.68**
Math	Linear	0.69	-0.03	0.15	-4.70	27.01
	SLERP	0.68	-0.04	0.14	-5.54	25.67
	TIES	0.71	-0.02	0.17	-2.09	31.59
	SIMMERGE	**0.71**	**-0.02**	**0.17**	**-2.09**	**31.59**
Multilingual	Linear	0.38	-0.16	0.04	-29.95	11.54
	SLERP	0.33	-0.22	-0.01	-39.72	-4.01
	TIES	0.29	-0.26	-0.05	-47.19	-15.91
	SIMMERGE	**0.40**	**-0.14**	**0.06**	**-25.63**	**18.42**
RAG	Linear	0.18	-0.18	-0.03	-49.98	-14.11
	SLERP	0.19	-0.16	-0.01	-45.07	-5.69
	TIES	0.18	-0.17	-0.02	-48.22	-11.09
	SIMMERGE	**0.20**	**-0.15**	**-0.00**	**-42.01**	**-0.42**

J.3 PER-TASK TRENDS ACROSS MERGE SIZES

Tables 1–3 report the exact per-task results for $k \in \{2, 3, 4\}$. Here we complement those tables with per-task trend plots that visualize how performance evolves with merge size under two reference points: (i) *expert-relative* change (Δ_{expert}), measuring preservation of task specialization, and (ii) *auxiliary-relative* change (Δ_{aux}), measuring retention of useful off-domain capability.

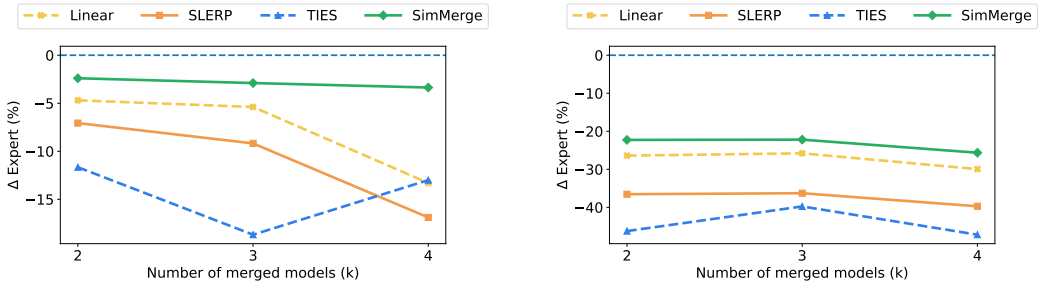


Figure 7: **Expert-relative trends.** Percentage change relative to the task expert (Δ_{expert}) as the number of merged models increases. Left: Code. Right: Multilingual. The effect of merge size can be non-monotonic (often a drop at $k = 3$ followed by partial recovery at $k = 4$), reflecting higher-order interactions between fine-tuned updates. Across both domains, SIMMERGE remains closest to the expert across merge sizes.

Expert-relative trends (Code and Multilingual). Figure 7 shows Δ_{expert} as a function of k for Code and Multilingual. Across both domains, SIMMERGE remains closest to the expert at every merge size, forming the upper envelope among all methods. Notably, the effect of increasing k is not strictly monotonic: several fixed operators exhibit a pronounced degradation from $k = 2$ to $k = 3$ followed by partial recovery at $k = 4$. This non-monotonicity is consistent with multi-way composition dynamics, where the third model can introduce the first strong conflict between specialized updates, while adding a fourth model can partially cancel harmful directions under equal-weight merging. The Multilingual domain is particularly sensitive: fixed operators separate more dramatically as k increases, while SIMMERGE remains consistently closer to the expert.

Auxiliary-relative trends (Math and RAG). Figure 8 plots Δ_{aux} for Math and RAG. As k grows, auxiliary gains can shrink or even become negative for fixed operators, indicating that naive multi-way merges can underperform the auxiliary baseline. This behavior is especially visible in RAG, where interference is strong and fixed operators often yield weakly positive or negative auxiliary

percentage change. In contrast, SIMMERGE more reliably maintains positive (or near-zero) auxiliary gains across merge sizes, suggesting better retention of off-domain capability while still limiting expert degradation (Figure 7).

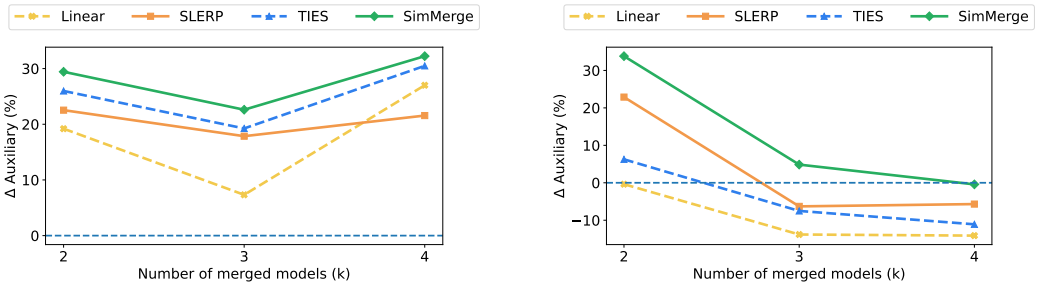


Figure 8: **Auxiliary-relative trends.** Percentage change relative to auxiliary baselines (Δ_{aux}) as the number of merged models increases. Left: Math. Right: RAG. Fixed operators can exhibit diminishing or negative auxiliary gains at larger k, particularly in RAG. SIMMERGE more consistently preserves improvements over auxiliaries across merge sizes, indicating more robust retention of off-domain capability under multi-way composition.

J.4 OVERALL GAP CLOSED SUMMARY

Figure 2 reports domain-level averages of *Gap Closed*. To summarize overall performance with a single scalar, we take an unweighted macro-average across the four domain means on Code, Math, Multilingual, RAG.

Table 4: Domain-averaged GAPCLOSED from Figure 2 and macro-average across domains.

Method	Code	Math	Multilingual	RAG	Macro avg.
LINEAR	69.0	61.5	37.1	-0.4	41.8
SLERP	53.3	72.2	12.9	23.0	40.4
TIES	23.6	83.3	-10.2	6.4	25.8
SIMMERGE	84.2	94.3	46.9	34.8	**65.0**

To complement Table 4 and Figure 2, Figure 9 reports the corresponding average performance for the auxiliary, expert and merged models.

Figure 9 makes two points explicit on the original task scales. First, the expert-auxiliary gap differs substantially by domain. For instance, the expert improves over the auxiliary from 0.538 to 0.634 on Code, from 0.570 to 0.748 on Math, from 0.346 to 0.596 on Multilingual, and from 0.207 to 0.408 on RAG. Second, SIMMERGE consistently produces merged models that move toward the expert while improving over the auxiliary baseline across all four domains. In the pairwise setting, SIMMERGE achieves the highest mean performance in every domain, reaching 0.62 on Code, 0.74 on Math, 0.46 on Multilingual, and 0.28 on RAG, outperforming the best fixed operator in each case. By contrast, the strongest fixed operator is domain-dependent: Linear on Code and Multilingual, TIES on Math, and SLERP on RAG, reinforcing that no single merge rule dominates across tasks. This absolute view also clarifies the trade-off: SIMMERGE improves off-domain performance while incurring smaller degradation relative to the expert than fixed baselines.

J.5 111B TASK-LEVEL RESULTS FOR 3-WAY MERGES

Figure 10 breaks down 3-way merging results at 111B by domain, reporting both auxiliary-relative gains Δ_{aux} and expert-relative degradation Δ_{expert} as defined in Section D.4. Across all four domains, SIMMERGE achieves the strongest expert-auxiliary trade-off. It produces the smallest degradation relative to the expert in every domain, and it also delivers the largest gain over auxiliaries. For example, on Code it reduces expert-relative degradation to -6.8% while improving over auxiliaries by $+20.4\%$. On RAG, it achieves a large auxiliary gain of $+90.4\%$ while keeping expert degradation at -10.4%.

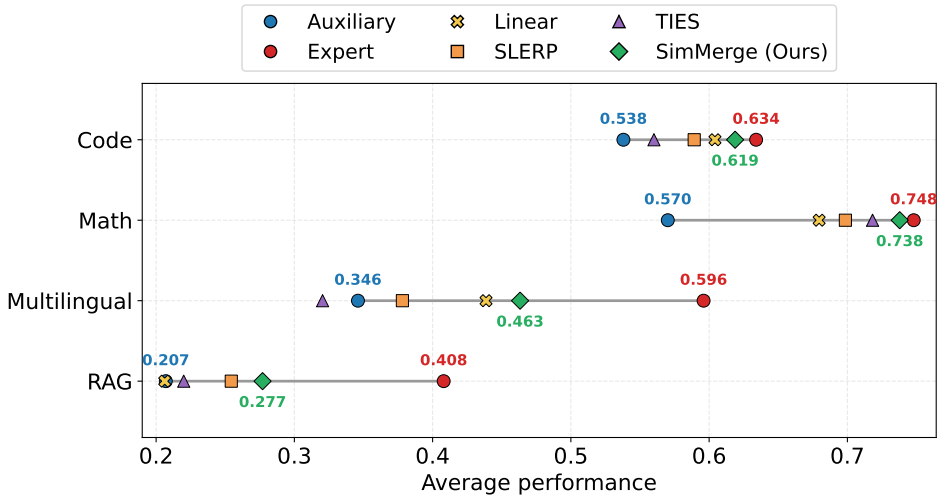


Figure 9: Absolute average performance per domain for auxiliary, expert, and merged models. This anchors the normalized metrics to the original task scales.

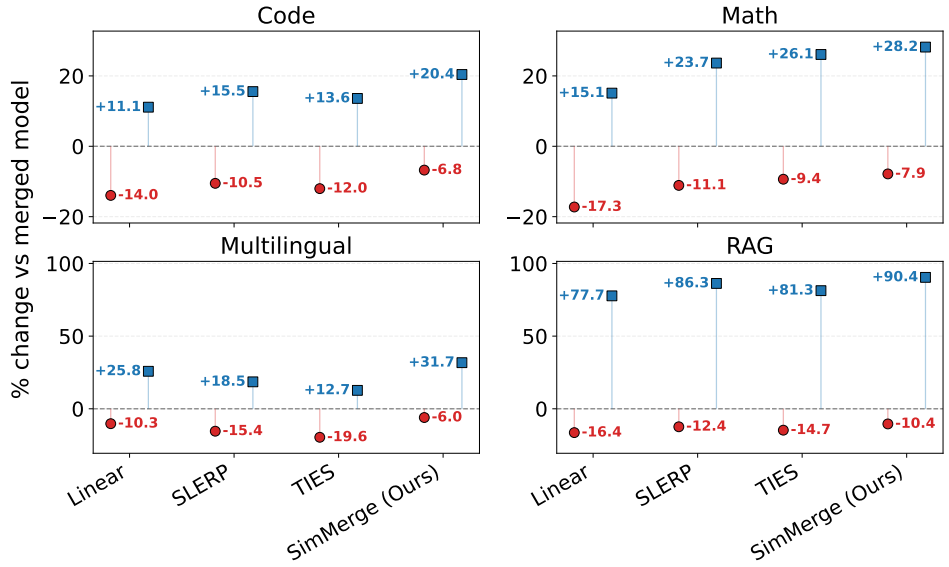


Figure 10: Per-task percentage change in performance for each merge method for 3-way merges at 111B. Blue markers show Δ_{aux} (change vs. auxiliary; higher is better) and red markers show Δ_{expert} (change vs. task expert; closer to 0 indicates less degradation), as defined in Section D.4.

Figure 11 reports the same comparison using GAPCLOSED, which normalizes performance so that 0% corresponds to the auxiliary baseline and 100% corresponds to the expert. SIMMERGE achieves the highest GAPCLOSED in every domain. The best fixed operator varies by domain, but SIMMERGE remains best overall, reaching 69.0 on Code, 76.6 on Math, 70.0 on Multilingual, and 80.3 on RAG.

K TAIL EFFECTS AND SIMILARITY CORRELATIONS

We begin by examining how similarity signals correlate with merge operator performance. Figure 12 shows Pearson and Spearman correlations between similarity features and performance outcomes

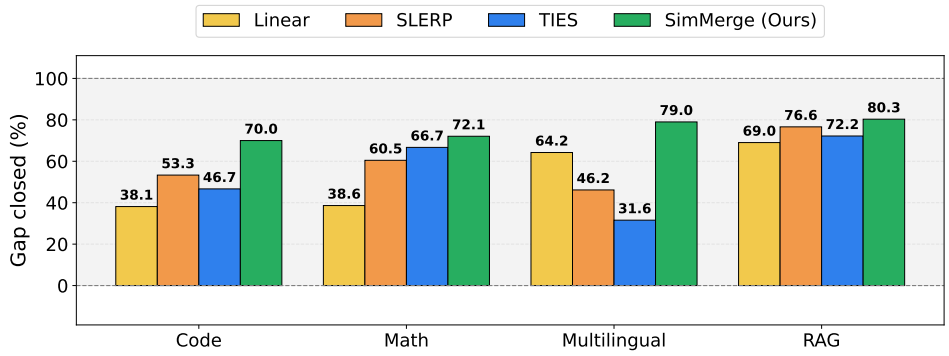
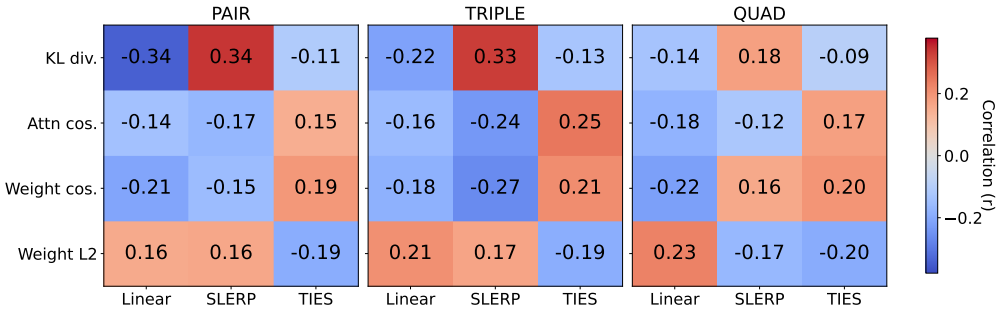


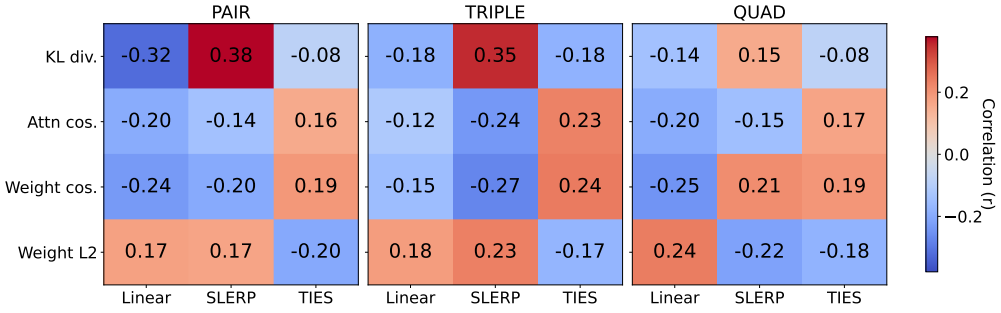
Figure 11: GAPCLOSED for 3-way merges at 111B across Code, Math, Multilingual, and RAG.

for LINEAR, SLERP and TIES across pairwise (PAIR), triple (TRIPLE), and quadruple (QUAD) merges.

Several consistent patterns emerge. Across all merge settings, different similarity features are predictive of success for different operators, often with opposing signs. KL divergence is positively correlated with SLERP performance but negatively correlated with LINEAR, while weight cosine similarity exhibits the opposite pattern. Attention-based cosine similarity shows positive correlation with TIES, whereas weight ℓ_2 distance is most predictive of LINEAR's success.



(a) Spearman



(b) Pearson

Figure 12: Correlation between similarity signals and merge performance for LINEAR, SLERP and TIES across pairwise, triple, and quadruple merges. Spearman and Pearson correlations exhibit consistent patterns, indicating robust, operator-specific similarity regimes.

The close agreement between Pearson and Spearman correlations indicates that these relationships are robust and largely monotonic rather than being driven by a small number of outliers. Importantly, no single similarity feature correlates positively with all operators, suggesting that merge quality is inherently regime-dependent.

Trends in percentile-bin. The percentile-bin curves in Figure 13 visualize how each merge operator's win probability ($P(\text{win})$) varies as a function within-case percentiles. Overall, the directions of these trends are consistent with the winner-metric correlations shown in Figure 12.

Figure 13a shows that moving from low to high KL percentiles increases $P(\text{win} = \text{SLERP})$ while decreasing $P(\text{win} = \text{LINEAR})$ while TIES is weakly decreasing or near-flat.

In Fig. 13b, $P(\text{win} = \text{TIES})$ increases monotonically with attention cosine similarity percentiles, while $P(\text{win} = \text{LINEAR})$ decreases, SLERP tends to decrease, most clearly in the triple merge setting.

Figure 13c shows that higher weight-cosine percentiles favor TIES and disfavor LINEAR across merge settings. Notably, SLERP decreases with weight cosine in pairwise and triple merges but increases in QUAD, consistent with the corresponding sign flip observed in Figure 12. This pattern is consistent with spherical interpolation becoming more reliable when strong mutual parameter alignment is present in four-way merges.

Figure 13d shows that increasing weight ℓ_2 percentiles increase $P(\text{win} = \text{LINEAR})$ and decrease $P(\text{win} = \text{TIES})$ across merge settings. SLERP increases with ℓ_2 distance in PAIR and TRIPLE but decreases in QUAD, again mirroring the sign changes in Figure 12. This pattern is consistent with a geometric interpretation of the merge operators. Large weight ℓ_2 distance reflects substantial parameter magnitude mismatch between models. In such regimes, Linear interpolation, which does not rely on directional alignment, tends to be more robust, while TIES degrades due to increased trimming under magnitude differences. SLERP improves with increasing ℓ_2 distance in pairwise and triple merges but degrades in the quadruple setting, where averaging across multiple directions becomes less stable.

Effects of merge size. Comparing the panels within each subfigure in Figure 13, QUAD trends are often flatter, indicating weaker dependence on similarity percentiles. Additionally, SLERP exhibits two notable merge-size-dependent reversals: (i) a positive association with weight cosine similarity in QUAD (Figure 13c), and (ii) a negative association with weight ℓ_2 distance in QUAD (Figure 13d). These effects indicate that multi-model geometry introduces interactions beyond those captured by pairwise relationships.

Tail effects and operator robustness. While similarity-conditioned trends describe average behavior across similarity regimes, they do not capture how performance is distributed across individual merge instances. In particular, an operator may perform well on average while still failing catastrophically on a nontrivial fraction of cases. To characterize this behavior, we analyze *tail effects*, which quantify whether a method's wins are concentrated in favorable regimes or whether it frequently appears among the worst-performing outcomes.

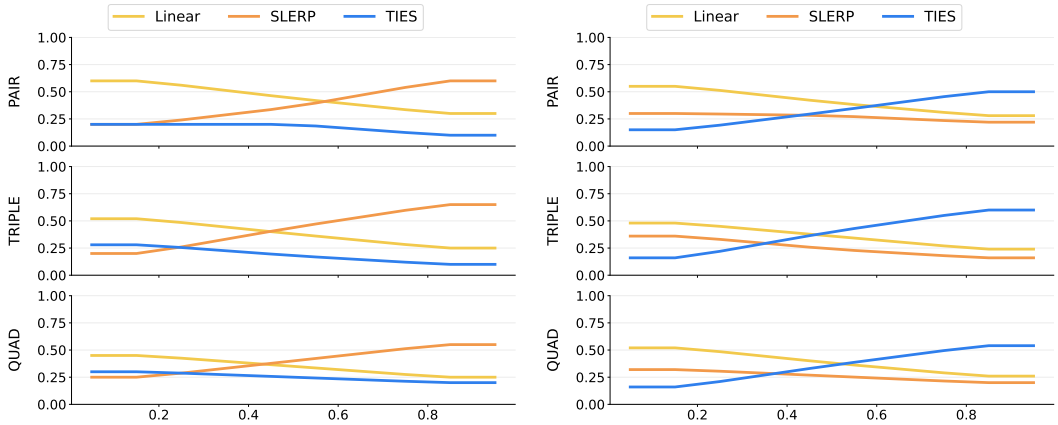
For each merge method, metric, and merge setting, we define the tail effect as

$$\Delta P(\text{win}) = P(\text{top } 20\%) - P(\text{bottom } 20\%) \tag{6}$$

where $P(\text{top } 20\%)$ denotes the probability that the method ranks in the top quintile of outcomes, and $P(\text{bottom } 20\%)$ denotes the probability of ranking in the bottom quintile. A large positive value indicates that a method consistently wins in favorable regimes while rarely failing badly, whereas values near zero or negative indicate brittle behavior with frequent severe failures.

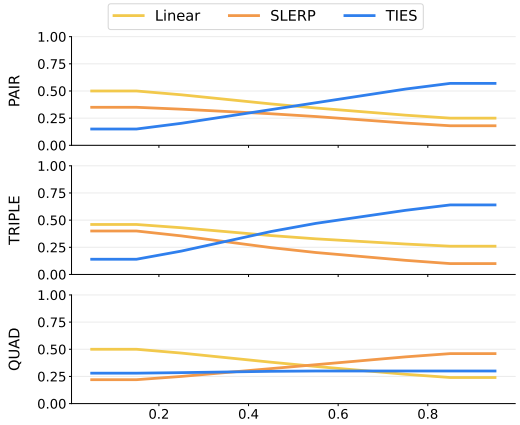
Figure 14 visualizes tail effects across similarity metrics and merge settings. Fixed operators exhibit strong and highly metric-dependent tail behavior. For example, LINEAR shows positive tail effects in regimes characterized by small weight distances, but negative or near-zero tail effects under KL divergence and attention-based similarity. Conversely, TIES concentrates wins when weight cosine or attention similarity is high, but frequently occupies the bottom tail outside these regimes. SLERP exhibits mixed behavior, with tail effects that change sign depending on both the similarity metric and the merge setting.

As merge complexity increases from pairwise to quad settings, tail effects generally become more pronounced. This indicates that applying a single operator uniformly across increasingly heterogeneous collections of models amplifies the risk of severe failures, even when average performance remains competitive. These tail failures explain why fixed operators can appear strong under aggregate metrics yet behave unreliably in practice.

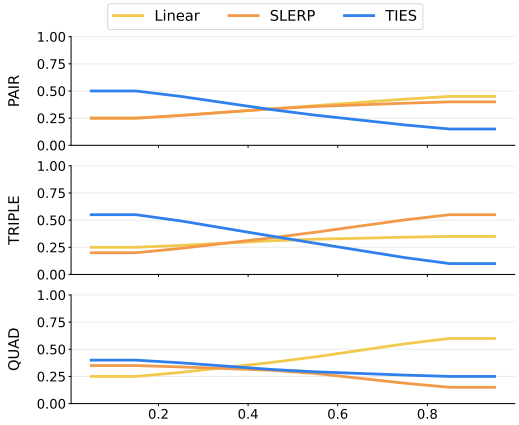


(a) KL divergence (logits)

(b) Attention cosine similarity



(c) Weight cosine similarity



(d) Weight ℓ_2 distance

Figure 13: **Percentile-bin win trends.** For each case (PAIR/TRIPLE/QUAD), we map each probe metric to its empirical percentile and compute $P(\text{win})$ for each merge operator within equal-mass percentile bins. This makes trends comparable across merges and across metrics with different raw scales.

Taken together, similarity-conditioned trends and tail effects show that merge operator effectiveness is inherently regime-dependent. Each operator succeeds only within specific similarity regimes and exhibits sharp failures outside them, leading to brittle behavior when a single rule is applied universally. By identifying these regimes through similarity signals and selecting operators on a per-instance basis, SIMMERGE avoids unfavorable tails and achieves robust merging behavior across tasks and merge settings.

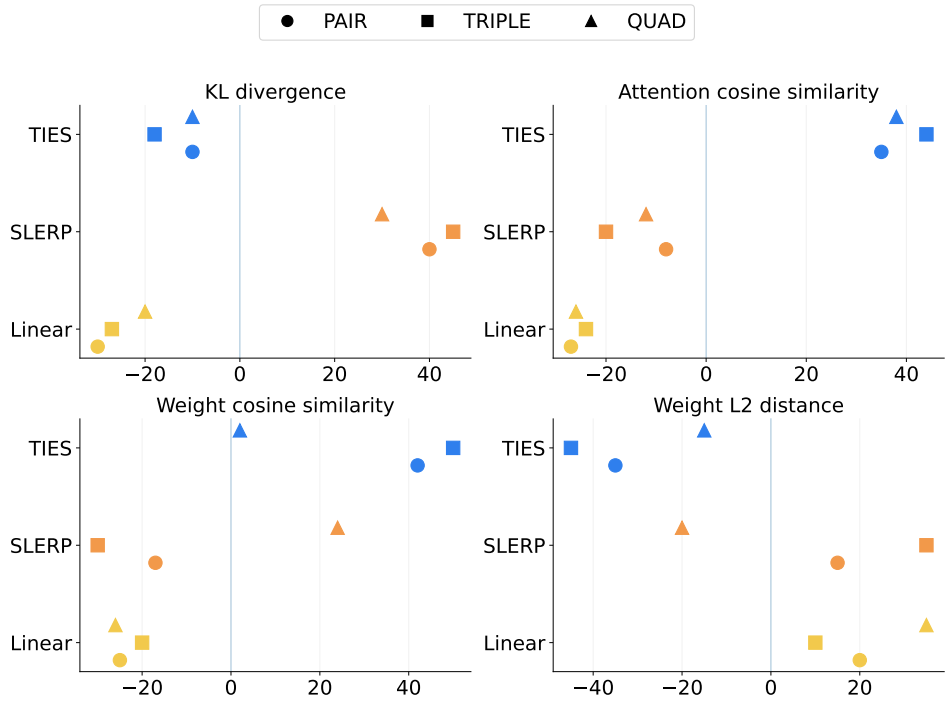


Figure 14: Tail effects of merge operators across similarity metrics and merge settings. Each point shows the tail-effect score (Eq. 6). Markers indicate pairwise, triple, and quad merges.