Fig. 16 Fig. 18 If You Can't Use Them, Recycle Them: Optimizing Merging at Scale Mitigates Performance Tradeoffs

Muhammad Khalifa ¹² Yi-Chern Tan ² Arash Ahmadian ³ Tom Hosking ² Honglak Lee ¹ Lu Wang ¹ Ahmet Üstün ³ Tom Sherborne ² Matthias Gallé ²

Abstract

Model merging has shown great promise at combining expert models, but the benefit of merging is unclear when merging "generalist" models trained on many tasks. We explore merging in the context of large (~100B) models, by recycling checkpoints that exhibit tradeoffs among different tasks. Such checkpoints are often created in the process of developing a frontier model, and the suboptimal ones are usually discarded. Given a pool of model checkpoints obtained from different training runs (e.g., different stages, objectives, hyperparameters, and data mixtures), which naturally show tradeoffs across different language capabilities (e.g., instruction following vs. code generation), we investigate whether merging can recycle such suboptimal models into a Pareto-optimal one. Our optimization algorithm tunes the weight of each checkpoint in a linear combination, resulting in such an optimal model that outperforms both individual models and merge-based baselines. Further analysis shows that good merges tend to include almost all checkpoints with nonzero weights, indicating that even seemingly bad initial checkpoints can contribute to good final merges.

1. Introduction

Model merging is gaining traction as a cost-effective alternative to joint multi-task learning (Wortsman et al., 2022; Yu et al., 2024). While the research here has rapidly advanced in the last few years, it remains limited in terms of both *model scale* and *the nature of considered checkpoints*. On one hand, most work has studied merging fairly small models ($\leq 10B$) by today's standards, and it remains unclear how much benefit merging brings with larger models ($\geq 100B$). On the other hand, the setup where merging was

Preprint.

mostly applied involved two or more *expert* models, where experts are independently optimized for specialized tasks, merged to combine their capabilities (Yadav et al., 2024a; Yu et al., 2024; Akiba et al., 2024). The primary motivation for such expert merging is to eliminate the cost of multitask training—each expert can be trained separately, and then later merged for combined expertise (Li et al., 2022). However, expert merging is only reasonable when expert models are available. Departing from that setting, modern large language model (LLM) development scenarios tend to produce a large number of multi-task models.

Training general-purpose LLMs relies on training a single model on many tasks during supervised finetuning (SFT)/instruction tuning (Chung et al., 2024; Jiang et al., 2023; Achiam et al., 2023; Dubey et al., 2024; Team et al., 2024; 2023). A prevalent issue here is that different tasks may conflict with each other (Lin et al., 2019), resulting in tradeoffs among different capabilities. Parameters well suited to one task may conflict or combine poorly with parameters specialized for another (Gueta et al., 2023). For example, aligning a model with human preferences can hurt performance on other evaluations (Bai et al., 2022; Vijjini et al., 2024). Similarly, improving math capabilities may come at the cost of other reasoning skills (Fu et al., 2023). One strategy to alleviate these tradeoffs is to carefully tune different training choices until a good enough (e.g., a Pareto-optimal) model is obtained. This approach is not only computationally expensive, but also discards suboptimal models under the assumption that they constitute "failed" experiments.

Adopting a realistic setting where we use 16 104B checkpoints that exhibit such tradeoffs, we ask: *How to merge multi-tasked models to reduce performance tradeoffs among different tasks?* We apply iterative search to optimize the contribution of each checkpoint to the mixture (see Fig. 1)

This paper extends model merging approaches by (i) utilizing models orders of magnitudes larger than previous work and, more importantly, (ii) leveraging it to minimize tradeoffs among multiple "generalist" checkpoints, as opposed to merging only "experts". This practical setup considers how to optimally re-use suboptimal intermediate checkpoints in

¹University of Michigan ²Cohere ³Cohere For AI. Correspondence to: Muhammad Khalifa <khalifam@umich.edu>.

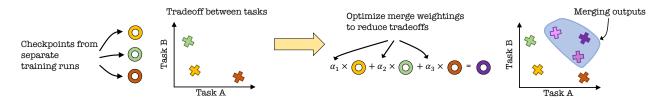


Figure 1. An overview of our setup. Given models obtained from different LLM training runs, we optimize linear merging weightings $(\alpha_1, \alpha_2, \alpha_3)$ via iterative search to obtain a model with minimal task tradeoffs. Each \circ represents a single model, with a \circ to designate its performance on the two tasks. The purple color indicates a Pareto-optimal model, achieving a good balance between the two tasks without being dominated by other models. We show tradeoffs between only two tasks since it is easier to visualize.

the style of model recycling (Choshen et al., 2022).

We make the following contributions:

- We explore an understudied setup where the goal is to optimize task tradeoffs by merging multiple models obtained from different training runs, with different data mixtures, post-training stages, and objectives (instead of only random seeds or hyperparamerts). This setting mirrors well the real-life scenario of developing a frontier model (§3.1).
- We propose to use evolutionary optimization to find such optimal merges instead of relying on manual tuning the weights, a process that can be tedious, unreliable and costly (§3.3). We show that such evolutionary optimization reduces task tradeoffs in two- and three-task setups, outperforming uniform and greedy merging without hurting out-of-domain performance on held-out tasks (§5).
- We analyze the optimal merges obtained via search and identify key lessons (§5.3): Merges with the best tradeoffs are the result of merging almost all checkpoints, suggesting that linear merging benefits from seemingly bad checkpoints that perform poorly on the tasks in question. In other words, merging can be optimized to recycle existing checkpoints, enabling training-free optimization of task tradeoffs. We also observe that the performance of a merge cannot be determined solely based on the isolated performance of the merged models, where seemingly low-performing checkpoints can lead to optimal merges.

2. Related Work

Prior work on model merging has mainly considered two axes: merging *how to merge* (methods); and *when to merge* (context).

Merging methods Recent efforts on model merging methods aim to overcome the limitations of simple averaging

(Utans, 1996; Wortsman et al., 2022) with more involved techniques. For instance, Matena & Raffel (2022) use the Fisher information matrix to approximate each model's posterior beyond the isotropic assumption made by averaging. Ilharco et al. (2023) propose to merge "task vectors" instead of full models. Yadav et al. (2024a) innovated upon averaging by proposing a 3-step process to resolve parameter interference among merged models while Daheim et al. (2023) do so by reducing mismatch in parameter gradients. A major downside of most of these methods is that they require access to gradient information, which becomes expensive as the models scale. In addition, other techniques such as DARE (Yu et al., 2024) assume access to a shared base model from which experts are fine-tuned, which is limiting in cases with heterogeneous models as in our setup.

Merging context Model merging has been mainly exploited in the context of transfer learning (Matena & Raffel, 2022; Jin et al., 2022; Yadav et al., 2024a; Hammoud et al., 2024; Akiba et al., 2024), where merging is done across a set of experts, specialized models that are fine-tuned separately on different tasks or domains, and which are then merged to combine their capabilities into a single multi-task model (Rame et al., 2024). Merging can also be exploited to ablate on dataset mixtures across domains during pre-training (Na et al., 2024). This does not fully capture typical large-scale LLM training scenarios. In practice, LLMs are simultaneously trained on a multitude of tasks during a single training run—suggesting that training is generally not approached as a multiple single-task optimization problem but rather as a single multi-task optimization problem (Ouyang et al., 2022; Jiang et al., 2023; Chung et al., 2024; Bai et al., 2023; Achiam et al., 2023; Team et al., 2023). It is more common to have multiple models exhibiting different performance tradeoffs than to have expert models specialized in certain domains. Team et al. (2024) demonstrate this setup and discuss the tuning of SFT hyperparameters (including training data mixture weightings) to minimize tradeoffs between helpfulness, safety, and faithfulness. Our work further explores optimizing merging across such checkpoints to minimize performance tradeoffs across critical benchmark tasks. **Contrast with prior work** In a similar spirit to our work, Wortsman et al. (2022) average models trained across different runs. However, their approach is applied to computer vision models and, more importantly, their goal is to maximize performance on a single task rather than minimize task tradeoffs. Ramé et al. (2024) similarly average LLM policies across different runs to improve alignment reward, but their singular goal is improving reward, contrasting to our exploration of tradeoffs. Their setup is also limited to merging a maximum of 5 smaller models, while we explore merging up to 16 104B checkpoints. Dubey et al. (2024) mention that they average checkpoints trained with different hyperparameters and data mixtures, but they do not provide any additional details. Another adjacent recent work is Akiba et al. (2024), using evolutionary optimization techniques to optimize merging across two expert models. They explore parameter-space merging (i.e., merging weights) and data-space merging (i.e., merging information flow across different layers). However, their setup is limited to merging only two 7B models. To summarize, our work sheds light on the underexplored setup where model merging can help recycle existing models beyond the top-few experts. Our work differs from prior work in at least three ways. First, our objective is to minimize performance tradeoffs across different training runs, as opposed to merging expert models. Second, our setup involves merging up to 16 different 104B, making our setup more realistic and aligned with the current scales of LLMs. Third, as opposed to relying on simple averaging (Wortsman et al., 2022) or tuning the weights manually, we leverage evolutionary search to optimize the merging weightings.

3. Optimizing LLM Merging

3.1. Task Conflict

Different tasks conflict with each other in required expertise, resulting in performance tradeoffs where improvement observed over some tasks incurs performance degradation on other tasks (Lin et al., 2019; Wang et al., 2021). For instance, in language modeling, aligning a model with human preferences could result in an *alignment tax*, where the model performance degrades on held-out tasks (Bai et al., 2022). Another example is that instruction tuning could hurt the performance on tasks such as question answering and reasoning (Dou et al., 2023). It is well known in practice that different decisions related to hyperparameters, training data mixtures, or long-context training yield such tradeoffs.

One way to minimize these tradeoffs is by carefully tuning the training choices. This involves running several training runs with different hyperparameters and choosing Paretooptimal runs with minimal tradeoffs (Team et al., 2024). However, this is cumbersome and expensive, and becomes less tractable as model parameters increase. We investigate whether it is possible to recycle the models obtained from such "failed" runs, which are usually abandoned or discarded altogether. We propose to use training-free model merging as an efficient and compute-minimal strategy to combining models.

3.2. The Optimization Problem

Given T tasks $\{t_1,\cdots,t_T\}$ and N checkpoints $\{\theta_1,\theta_2,...,\theta_N\}$ which tradeoff across the tasks, we aim to output a model θ_{mrg} by merging the checkpoints such that all tradeoffs are minimized. We focus on weight interpolation of the model weights, or model soups (Wortsman et al., 2022), which yield a merged model by linearly interpolating the model parameters with non-negative weightings. Precisely, the resulting model is computed as a weighted sum of the individual model parameters i.e., $\theta_{mrg} = \sum_{i=1}^{N} \alpha_i \theta_i$, where α_i represents the weighting assigned to θ_i , subject to $\sum_{i=1}^{N} \alpha_i = 1$.

We limit our study to linear merging (i.e., model soups) for three reasons. First, linear merging is easy to implement and is generally considered a strong baseline (Wortsman et al., 2022; Ilharco et al., 2022b; Yadav et al., 2024a). Second, it makes few assumptions about the merged models (e.g., assumes no shared base model (Ilharco et al., 2022a), or access to gradient information (Matena & Raffel, 2022)) enabling easy scaling to large models. Third, recent work (Yadav et al., 2024b) has shown that different merging methods, including linear merging, may converge approximately to the same performance at large model scales.

Let $P_t(\theta')$ represent the performance of the model θ' on task t. Given a candidate model θ' , we quantify the performance tradeoffs using a fitness function $R(\theta') = R(P_{t_1}(\theta'), \cdots, P_{t_T}(\theta'))$ which assesses the balance across tasks to capture all tradeoffs. That is, higher fitness means less severe tradeoffs. Thus, our objective is to find the optimal weightings vector $a^* = \{\alpha_1^*, \alpha_2^*, \cdots, \alpha_N^*\}$ that minimizes the task tradeoffs and therefore maximizes R: $a^* = \arg\max_a R(P_{t_1}(\theta_{\mathrm{mrg}}), \dots, P_{t_T}(\theta_{\mathrm{mrg}}))$.

This optimization formulation seeks a balanced performance across tasks by adjusting the weights in a.

Fitness Function We rely on single-objective optimizations, and therefore we require a single score that reflects the performance tradeoff between the given tasks. One choice could be the product of all task metrics or a weighted average of the metrics—if some tasks are considered more important than others. We will use the **unweighted macroaverage** of task performances of the merging output θ_{mrg} , matching the majority of LLM evaluations for comparison to prior work (Dubey et al., 2024; Team et al., 2024). Over the tasks $t \in \{t_1, \cdots t_T\}$, the fitness function of a model θ'

is computed as $R(\theta') = \frac{1}{T} \sum_{t \in \{t_1, \cdots t_T\}} P_t(\theta')$.

3.3. Search Algorithm

There exists a plethora of techniques to solve such problems, including Bayesian Optimization (Snoek et al., 2012), random search (Bergstra & Bengio, 2012), and genetic algorithms (Young et al., 2015; Alibrahim & Ludwig, 2021). For the purpose of our study, we focus on Covariance Matrix Adaptation Evolution Strategy (Hansen & Ostermeier, 2001), which has proven its usefulness in hyperparameter optimization (Loshchilov & Hutter, 2016, CMA-ES), including in model merging literature (Akiba et al., 2024), as well as requiring minimal hyperparameter configuration.

CMA-ES is an optimization technique suited for continuous, non-linear optimization. It improves solutions by iteratively sampling candidates from a multivariate normal distribution, with a mean representing the instantaneous optimal solution. At each step, CMA-ES adapts the covariance matrix of the normal distribution over time, based on successful solutions, to assign higher probability to good solutions. CMA-ES is suitable for cases where gradient information is not available or is expensive to obtain. Details of CMA-ES are shown in Algorithm 1 in App. A.

4. Experimental Setup

Merge candidates As candidate inputs to the merge, we select 16 models from a 104B development pipeline, where each model is the result of a separate training run. To ensure these models are representative of different training stages, we select them such that:

- 50% are sourced from the supervised fine-tuning (SFT) training stage, and the other 50% from preference optimization (PO) stage.
- The SFT checkpoints involve different training data: some are trained purely on code, some on code and academic data, and some on the general multi-task SFT setup, etc.,
- The PO checkpoints include variation of the training objective and hyperparameters (e.g., warmup ratio).
- One checkpoint is sourced from an earlier version of this model's family.

Table 2 in App. B includes details about each model. We first note that the PO models are not necessarily based on the SFT models; some are based on different SFT models not included in our candidate models. Aside from the criteria above, the checkpoints were selected without any specific

bias; we do not intentionally pick checkpoints that exhibit high performance tradeoffs between tasks, as these tradeoffs appear naturally anyway. Our expectation is that our merging strategy can easily apply to other researchers and practitioners who may have many underexploited checkpoints "laying around".

Tasks We evaluate the merged models on both heldin and held-out tasks. The heldin tasks are the ones for which we wish to minimize the tradeoffs. The heldin tasks serve to verify that the merged model still performs well on tasks outside the heldin tasks. As heldin tasks, we select five different tasks that are representative of different model capabilities. We use MBPP² to evaluate code performance, GSM8K (Cobbe et al., 2021) for math, IFEval (Zhou et al., 2023) for instruction following, MMLUPro (Wang et al., 2024)³ for multitask language understanding, and MUSR (Sprague et al., 2024) for multistep reasoning. Held-out tasks include MT-bench (Zheng et al., 2023) and LBPP (Matton et al., 2024).

Evaluation We report pass@1 for code tasks i.e., MBPP and LBPP. We use accuracy for GSM8K, MMLU Pro, and MUSR. For IFEval, we report the prompt-level strict accuracy (Zhou et al., 2023), which measures the percentage of prompts for which all verifiable instructions are followed. All evaluations use greedy decoding.

Search details We use the CMA-ES implementation from Optuna (Akiba et al., 2019). The population size is set to $4+3\ln N$ (default in the implementation), which aligns with prior work (Akiba et al., 2024). We initialize the search with uniform weights, i.e., $\alpha_i=1/N$ for all i. We set the initial standard deviation $\sigma_0=1.0$ and run CMA-ES for 50 iterations in total. At each iteration, CMA-ES proposes a single weighting vector $\{\alpha_1',\alpha_2',\cdots,\alpha_N'\}$, which we use to merge the individual models and then evaluate the fitness over the resulting merge. To make sure the weightings sum to 1, we first sample unnormalized weights and then normalize by their sum. To avoid cold starting the CMA-ES optimization, we initialize the CMA-ES history with the performance from individual checkpoints results and the two merged baselines detailed below.

Baselines To validate how optimal the found solution is, we compare it against the following baselines:

• **Best single model:** The individual model with the highest fitness function in addition to the individual model with

¹A model achieving the best average task performance is necessarily Pareto-optimal, as no other model can dominate it without achieving a better average.

²https://github.com/google-research/ google-research/tree/master/mbpp.

³To speed up experimentation, we use a subset of 2K examples from MMLU-Pro. We found the performance on this subset to correlate highly with performance on the full dataset.

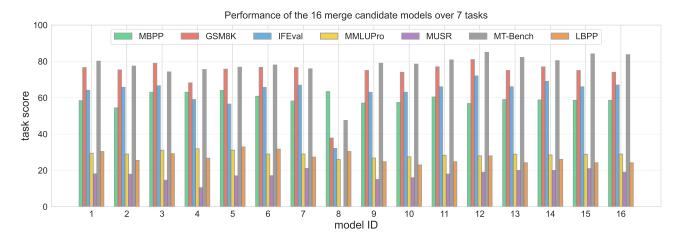


Figure 2. Performance of individual models over the seven tasks covering different capabilities. Models 1-8 are the result of supervised finetuning runs, while 8-16 from preference optimization. Held-out tasks (MT-Bench and LBPP) are used to evaluate the resulting merges to make sure the merge optimization process does not overfit to the held-in tasks that we aim to minimize tradeoffs over. MT-Bench rating is scaled by a factor of 10 for better visualization. The exact numbers are in Table 2 in App. B.

the highest performance on each task.

- Uniform soup: Averaging all checkpoints using equal weights: $\alpha_i = \frac{1}{N}$ (Wortsman et al., 2022).
- Merge-best: We average the top-performing checkpoints for each task. Specifically, we merge the checkpoints $\{\theta_{t_1}^*,\cdots,\theta_{t_T}^*\}$ where $\theta_t^*=\arg\max_{\theta}P_t(\theta)$ with uniform weight. This corresponds to assigning the highest weights to the best performing model on each task and zero weights to the remaining models.

5. Results and Discussion

Before merging any models, we first look at the performance of individual models across different tasks to get a sense of the existing tradeoffs. Fig. 2 shows the performance of each of the 16 merge candidates over both held-in and held-out tasks. Table 2 in App. B shows exact numbers. Interesting tradeoffs show up when looking at individual model performances. For instance, SFT models (1-8) exhibit better code performance (i.e., on MBPP and LBPP) compared to PO models (9-16), while PO models seem to perform better than SFT ones on MT-Bench and IFEval. This is a tradeoff likely caused by alignment training, which could hurt some other model capabilities (Bai et al., 2022).

Now we zoom in on *pairwise* tradeoffs between two tasks. In this case, it is fairly straightforward to measure the severity of such tradeoffs by computing performance correlation across the models. Fig. 4 shows Spearman's rank correlation ρ between all task pairs over our 16 selected models. We observe a few strong pairwise tradeoffs between task pairs, such as MBPP-IFEval ($\rho=-0.35$) and MBPP-MUSR ($\rho=-0.40$). Throughout this section, we will refer to the

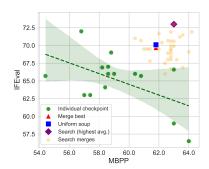
tasks with tradeoffs as *held-in* tasks. The main question we aim to answer here through our experiments is: Can optimizing $\{\alpha_1,\alpha_2,\cdots,\alpha_N\}$ yield $\theta_{\rm mrg}$ with minimal tradeoffs over the held-in tasks without a hurting performance over the held-out ones?

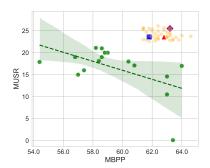
5.1. Optimizing Pairwise Tradeoffs

We apply our merge optimization recipe over three task pairs with relatively strong tradeoffs: **MBPP-IFEval**, **MBPP-MUSR**, and **MMLU Pro-IFEval**.

Fig. 3 shows a plot for each pair of tasks. Each plot includes a best-fit line to the performances of each pair (shown in green), which exhibits a negative slope in all three cases due to the respective tradeoff. We observe that baselines such as 'Uniform Soup' and 'Merge-Best' seem to reduce tradeoffs in a few cases. However, search-based optimization of the merge always yields a model that falls on the Pareto frontier—occasionally outperforming baselines by up to 2.2 average points, as with MBPP-IFEval. In addition, we can see that over MBPP-IFEval and MBPP-MUSR, our search has yielded a model that is better than the best individual model on both IFEval (+1.0) and MUSR (+4.4), showing that search optimized merges could improve over the initial candidate models. We also note that 'Merge-Best' baseline, which averages the two best performing models on each task, performs comparably well in some cases (MMLU Pro-IFEval) but performs below search in the other task pairs. This suggests that merging based on individual model performance sometimes results in suboptimal merges. We dive deeper into the weightings found via search in Sect. 5.3.

Looking at the held-in tasks alone however does not give us the full picture. It is possible that search has overfit to the





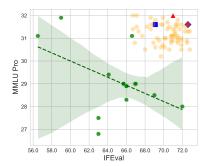


Figure 3. Performance tradeoffs with different merging approaches over different pairwise combinations. Shaded areas represent 95% confidence interval of the best-fit line computed over individual checkpoint scores (shown in green).

	Не	eld-In	d-In Held-O			Avg. All Tasks	
	MBPP	IFEval	Avg.	MT-Bench	LBPP		
Highest fitness model	63.0	65.7	60.1	7.42	30.4	41.6	
Best on MBPP	64.0	56.5	60.3	7.68	32.9	40.3	
Best on IFEval	56.8	72.0	64.4	8.50	28.0	41.3	
Uniform Soup	61.8	70.1	66.0	8.13	29.8	42.5	
Merge-best	61.8	69.7	65.8	8.38	32.3	43.0	
Search-optimized (ours)	63.0	73.0	68.0	8.07	32.3	44.1	
	MBPP	MUSR	Avg.	MT-Bench	LBPP		
Highest fitness model	64.0	17.0	40.5	7.68	32.9	30.4	
Best on MBPP	64.0	17.0	40.5	7.68	32.9	30.4	
Best on MUSR	58.2	21.1	39.7	7.59	27.3	28.6	
Uniform Soup	61.8	23.6	42.7	8.17	31.7	31.3	
Merge-best	62.8	23.6	43.2	7.88	30.4	31.2	
Search-optimized (ours)	63.2	25.5	44.4	8.27	30.4	31.8	
	MMLU Pro	IFEval	Avg.	MT-Bench	LBPP		
Highest fitness model	28.0	72.0	50.0	8.50	28.0	34.1	
Best on MMLUPro	31.9	59.0	45.5	7.55	26.7	31.3	
Best on IFEval	28.0	72.0	50.0	7.55	26.7	33.6	
Uniform Soup	31.6	70.1	50.9	8.19	29.2	34.8	
Merge-best	32.0	71.0	51.5	8.13	31.1	35.6	
Search-optimized (ours)	31.6	72.6	52.1	8.15	31.1	35.9	

Table 1. Performance of different baselines compared to search optimized merge in the pairwise case. Held-in tasks refer to tasks included in the fitness function (§ 3.2) and held-out tasks are used to validate the quality of the search optimized models. Search yields models with the lowest tradeoffs over the held-in tasks without sacrificing performance on held-out tasks. We highlight single models and merges differently. Evaluations on MMLU Pro use only 2K test examples.

held-in tasks by yielding merges with minimal tradeoffs, but could perform significantly worse on tasks that were not incorporated into the fitness function, i.e., out-of-distribution tasks. To verify whether this is the case, we evaluate the resulting merge on held-out tasks, namely, MT-Bench and LBPP. As shown in Table 1 the search-optimized merges exhibit comparable performance on the held-out tasks—and in some cases even better than baselines. This means search

has minimized task tradeoffs over the held-in tasks without compromising performance on other tasks.

5.2. Optimizing Three-task Tradeoffs

In practice, production LLMs are expected to be performant at more than two tasks. We consider balancing performance across three tasks: code generation, instruction

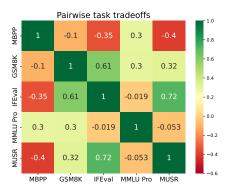


Figure 4. Spearman's rank correlation between task pairs. It is easy to see how some tasks exhibit strong performance tradeoffs, such as MBPP-IFEval and MMLU-Pro/MUSR.

following, and math reasoning, by using **MBPP-IFEval-GSM8K** as held-in tasks. We target this combination since IFEval correlates both negatively with MBPP, and positively with GSM8K (as shown in Fig. 4), making it a challenging combination to optimise. The goal is to identify how our approach scales with more than 2 tasks, due to the exponential growth in choices (and search space) with respect to the number of tasks.

Looking at Fig. 5, it is evident that the best fitness single model (i.e., highest average performance) performs well on IFEval and GSM8K, but comparably poor on MBPP. The other two baselines, 'Merge-Best' and 'Uniform Soup', were able to improve the tradeoffs by some degree but exhibit noticeable performance drop on IFEval. While the search-optimized merge is a Pareto-optimal model – maintaining – and only slightly underperforms the best single model for each task. These results suggest that search-optimized merging can perform well when scaling the number of tasks. Our approach maintains comparable performance on the held-out tasks, and even improves performance on LBPP compared to the baselines as shown in Table 3 in App. C.

5.3. Analysis

We perform further analysis into the dynamics of linear merging:

Most models contribute to the best merges. The first question we ask is: What fraction of the initial 16 models contribute to the best solutions? Fig. 6 shows a heatmap of the top 5 solutions on each task combination. We observe that CMA-ES identifies good solutions which distribute the weightings among almost all checkpoints (dense solution), instead of assigning high weights to a small subset of the models (sparse). For example, the top solution assigns very few zero weightings (shown in black in Fig. 6) for MBPP-

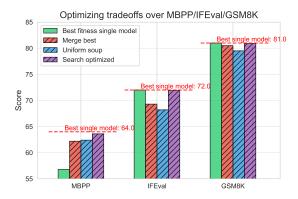


Figure 5. Performance of different merge approaches when minimizing the tradeoffs across three tasks: MBPP, IFEVal, and GSM8K. Dashed red lines represent the best individual model at the corresponding task. It is clear that search-optimized merging can well balance the performance over the three tasks. Bars corresponding to merging are hatched to differentiate from individual models.

MUSR and MBPP-IFEval. Also, while the top solutions for MBPP-IFEVal-GSM8K are slightly sparser in the pairwise case, at least 9/16 weightings are non-zero for the top 5 solutions. This indicates that almost all checkpoints have contributed to the optimal merge.

Low performing models may lead to optimal merges.

Since the CMA-ES optimization process evaluates many solutions, we can investigate the solutions found through search along with their quality, as measured by their standalone performance. Intuitively, one would expect that high fitness solutions found through search will assign higher weights to the checkpoints that perform well on the held-in tasks, compared to checkpoints that perform poorly.

Interestingly, this *is not* necessarily the case for the top solutions found by CMA-ES. For instance, the top performing solution for MBPP-MUSR has a relatively high weight for $\alpha_8 = 0.09$, even if model #8 performs extremely poorly at MUSR.⁴ The same holds for MBPP-IFEval pair, where the top solution does not assign a particularly high weight to any of the best performing models on MBPP (models #3, #4, or #5). Similarly, over MBPP-IFEval-GSM8K, α_8 is relatively high in the top solution, while model #8 actually performs the worst on GSM8K. An individual model's performance on given task(s) does not reflect the performance of a merge that assigns high/low weight to this model. An optimization procedure to find good merges is therefore necessary, since simply assigning the weightings based on the model's isolated performance is suboptimal.

Similarly, an individual model performance across all held-

⁴In fact, this particular code trained model has 0% accuracy, as shown in Fig. 2; It responds to all queries by generating code.

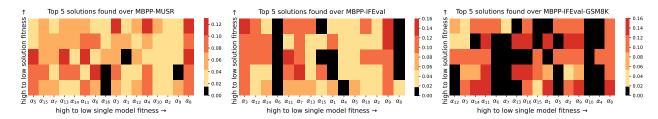


Figure 6. Best solutions found via CMA-ES search when optimizing tradeoffs over the pairs MBPP-MUSR (left) MBPP-IFEval (mid) and MBPP-IFEval-GSM8K. We order the weightings over the x-axis based on the fitness of the individual model they correspond to. We observe that top-solutions do not necessarily assign high weights to high-fitness individual checkpoints. For instance, the top solution on MBPP-IFEval assigns considerably high weight to model #9, which exhibits a relatively bad tradeoff on the task pair.

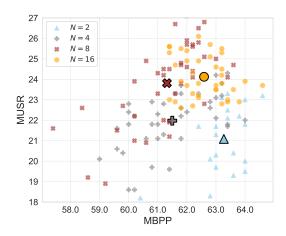


Figure 7. Merges found via CMA-ES when optimizing MBPP-MUSR tradeoffs over 2, 4, 8, and 16 checkpoints. We also show the centroid of each set of experiments (in large markers). Optimizing over more checkpoints (8 and 16) tends to yield less tradeoffs compared to fewer checkpoints (2,4), showing how recycling more models can outperform recycling fewer checkpoints.

in tasks is not predictive of its importance in the found solution. In Fig. 6, we sort the α_i weightings based on the fitness of the corresponding model for each experiment. For example, in the leftmost heatmap corresponding to MBPP-MUSR, model #5 has the highest average performance and model #8 has the lowest. If the performance of an individual model would be indicative of its weight in the final merge, we would expect higher weights (more reddish) to be concentrated on the left of each heatmap. The fact that this does not happen suggests that hand tuning the merge weightings based on heuristics (e.g., individual model performance, fitness, etc.) is suboptimal, further supporting the perspective of approaching model merging as an optimization problem.

Fitness improves with more iterations. We inspect whether CMA-ES effectively optimizes the fitness function as the search progresses by looking at the fitness function development over the course of search. Fig. 9 in App. C in plots the fitness function vs the number of iterations.

Each point in the graph is a weightings vector proposed by CMA-ES, and the fitness is the average of the held-in task performances of the resulting merge. Over the three pairwise task combinations, it is clear that the average solution fitness improves with more CMA-ES iterations.

Recycling benefits from more checkpoints. Including more initial checkpoints obviously extends the search space, but may lead to better solutions. To study how the merge quality changes with the number of checkpoints, we run CMA-ES over the N checkpoints with the best fitness scores, with $N \in \{2,4,8,16\}$. The results over MBPP-IFEval, and MBPP-MUSR are shown in Fig. 7 and Fig. 8 (in App. C), where we highlight the centroid for each N. As can be seen, with larger N the search space is explored more exhaustively and results in centroid checkpoints with better fitness.

Computational cost. In App. D, we estimate the computational cost required during both SFT and PO stages of a single model and compare it to the cost of our merge optimization recipe. We find that our recipe requires only about 10% compute of that needed for training, showing that search-optimized merging provides a significantly cheaper and training-free approach for reducing task tradeoffs.

Conclusion

In this paper, we present an approach to recycle checkpoints obtained during a typical training run of a frontier model. While the vast majority of those checkpoints are in general discarded, in this paper we show how to leverage them via search-optimized merging. We show that a simple search algorithm focusing on linear merging can yield better, and often Pareto-optimal models with respect to the existing checkpoints. Our research show that it is possible to leverage merging when we have many multi-task trained checkpoints, as opposed to the standard setup of merging experts. A surprising finding is that even checkpoints which perform relatively bad on subtasks can contribute to an overall better model. While we relied on a simple merging approach,

we hope future development will further investigate more involved merging techniques in a similar setup to ours via merging as a cheaper and training-free approach.

Impact Statement

This paper presents work whose goal is to utilize suboptimal large language model checkpoints by merging them into better models. Potential societal consequences of our work include accelerating frontier model training and development. There may be other potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.
- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.
- Akiba, T., Shing, M., Tang, Y., Sun, Q., and Ha, D. Evolutionary optimization of model merging recipes. *arXiv* preprint arXiv:2403.13187, 2024.
- Alibrahim, H. and Ludwig, S. A. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 1551–1559. IEEE, 2021.
- Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., Das-Sarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.
- Bergstra, J. and Bengio, Y. Random search for hyperparameter optimization. *Journal of machine learning* research, 13(2), 2012.
- Choshen, L., Venezian, E., Don-Yehia, S., Slonim, N., and Katz, Y. Where to start? analyzing the potential value of intermediate models. *arXiv preprint arXiv:2211.00107*, 2022.

- Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv* preprint arXiv:2110.14168, 2021.
- Daheim, N., Möllenhoff, T., Ponti, E. M., Gurevych, I., and Khan, M. E. Model merging by uncertainty-based gradient matching. *arXiv preprint arXiv:2310.12808*, 2023.
- Dou, S., Zhou, E., Liu, Y., Gao, S., Zhao, J., Shen, W., Zhou, Y., Xi, Z., Wang, X., Fan, X., et al. The art of balancing: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment. arXiv preprint arXiv:2312.09979, 2023.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.
- Fu, Y., Peng, H., Ou, L., Sabharwal, A., and Khot, T. Specializing smaller language models towards multi-step reasoning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10421–10430. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/fu23d.html.
- Gueta, A., Venezian, E., Raffel, C., Slonim, N., Katz, Y., and Choshen, L. Knowledge is a region in weight space for fine-tuned language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 1350–1370, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023. findings-emnlp.95. URL https://aclanthology.org/2023.findings-emnlp.95.
- Hammoud, H. A. A. K., Michieli, U., Pizzati, F., Torr, P., Bibi, A., Ghanem, B., and Ozay, M. Model merging and safety alignment: One bad model spoils the bunch. *arXiv* preprint arXiv:2406.14563, 2024.
- Hansen, N. and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

- Ilharco, G., Ribeiro, M. T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022a.
- Ilharco, G., Wortsman, M., Gadre, S. Y., Song, S., Hajishirzi, H., Kornblith, S., Farhadi, A., and Schmidt, L. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems*, 35: 29262–29277, 2022b.
- Ilharco, G., Ribeiro, M. T., Wortsman, M., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. arXiv preprint arXiv:2310.06825, 2023.
- Jin, X., Ren, X., Preotiuc-Pietro, D., and Cheng, P. Dataless knowledge fusion by merging weights of language models. arXiv preprint arXiv:2212.09849, 2022.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.
- Li, M., Gururangan, S., Dettmers, T., Lewis, M., Althoff, T., Smith, N. A., and Zettlemoyer, L. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.
- Lin, X., Zhen, H.-L., Li, Z., Zhang, Q.-F., and Kwong, S. Pareto multi-task learning. Advances in neural information processing systems, 32, 2019.
- Loshchilov, I. and Hutter, F. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.
- Matena, M. S. and Raffel, C. A. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.
- Matton, A., Sherborne, T., Aumiller, D., Tommasone, E., Alizadeh, M., He, J., Ma, R., Voisin, M., Gilsenan-McMahon, E., and Gallé, M. On leakage of code generation evaluation datasets. *arXiv preprint arXiv:2407.07565*, 2024.

- Na, C., Magnusson, I., Jha, A. H., Sherborne, T., Strubell, E., Dodge, J., and Dasigi, P. Scalable data ablation approximations for language models through modular training and merging. In *The 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Rame, A., Couairon, G., Dancette, C., Gaya, J.-B., Shukor, M., Soulier, L., and Cord, M. Rewarded soups: towards pareto-optimal alignment by interpolating weights finetuned on diverse rewards. *Advances in Neural Informa*tion Processing Systems, 36, 2024.
- Ramé, A., Ferret, J., Vieillard, N., Dadashi, R., Hussenot, L., Cedoz, P.-L., Sessa, P. G., Girgin, S., Douillard, A., and Bachem, O. Warp: On the benefits of weight averaged rewarded policies. *arXiv preprint arXiv:2406.16768*, 2024.
- Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.
- Sprague, Z., Ye, X., Bostrom, K., Chaudhuri, S., and Durrett, G. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=jenyYQzue1.
- Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. Gemma: Open models based on gemini research and technology. arXiv preprint arXiv:2403.08295, 2024.
- Utans, J. Weight averaging for neural networks and local resampling schemes. In *Proc. AAAI-96 Workshop on Integrating Multiple Learned Models. AAAI Press*, pp. 133–138. Citeseer, 1996.
- Vijjini, A. R., Chowdhury, S. B. R., and Chaturvedi, S. Exploring safety-utility trade-offs in personalized language models. arXiv preprint arXiv:2406.11107, 2024.

- Wang, Y., Wang, X., Beutel, A., Prost, F., Chen, J., and Chi, E. H. Understanding and improving fairness-accuracy trade-offs in multi-task learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1748–1757, 2021.
- Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024.
- Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965– 23998. PMLR, 2022.
- Yadav, P., Tam, D., Choshen, L., Raffel, C. A., and Bansal, M. Ties-merging: Resolving interference when merging models. Advances in Neural Information Processing Systems, 36, 2024a.
- Yadav, P., Vu, T., Lai, J., Chronopoulou, A., Faruqui, M., Bansal, M., and Munkhdalai, T. What matters for model merging at scale? *arXiv preprint arXiv:2410.03617*, 2024b.
- Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., and Patton, R. M. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the workshop on machine learning in high-performance computing environments*, pp. 1–5, 2015.
- Yu, L., Yu, B., Yu, H., Huang, F., and Li, Y. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.
- Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.
- Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D., and Hou, L. Instruction-following evaluation for large language models. *arXiv* preprint *arXiv*:2311.07911, 2023.

A. Optimization Algorithm

Algorithm 1 Shows a pseudocode of CMA-ES used to optimize the merge weightings, as explained in Sect. 3.2.

Algorithm 1 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

```
1: Input: Objective function f, population size \lambda, initial mean
        \mathbf{m}_0, initial step size \sigma_0, maximum iterations T
 2: Output: Optimized solution mopt
 3: Initialize:
 4: \mathbf{m} \leftarrow \mathbf{m}_0
                                                                                            // Initial mean
 5: \sigma \leftarrow \sigma_0
                                                                                      // Initial step size
 6: \mathbf{C} \leftarrow \mathbf{I}
                                                                    // Initial covariance matrix
 7: \mathbf{p}_{\sigma} \leftarrow \mathbf{0}, \mathbf{p}_{c} \leftarrow \mathbf{0}
                                                                                     // Evolution paths
 8: Define recombination weights w_i for \lambda offspring
 9: for t = 1 to T do
10:
             Sample offspring:
11:
             for k=1 to \lambda do
                  \mathbf{x}_k \sim \mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})
12:
13:
             end for
14:
             Evaluate fitness:
15:
             for k=1 to \lambda do
16:
                  f_k \leftarrow f(\mathbf{x}_k)
             end for
17:
             Sort offspring by fitness:
18:
19:
             Sort \{\mathbf{x}_k\} by ascending f_k
20:
             Update mean:
             \mathbf{m} \leftarrow \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}
21:
                                                                                     // μ best offspring
22:
             Update evolution paths:
             \mathbf{p}_{\sigma} \leftarrow (1 - c_{\sigma})\mathbf{p}_{\sigma} + \sqrt{c_{\sigma}(2 - c_{\sigma})\mu_{\text{eff}}}\mathbf{C}^{-1/2}\frac{\mathbf{m} - \mathbf{m}_{\text{prev}}}{\tau}
23:
             \mathbf{p}_c \leftarrow (1 - c_c)\mathbf{p}_c + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{\mathbf{m} - \mathbf{m}_{\text{prev}}}{\sigma}
24:
25:
             Update covariance matrix:
26:
             \mathbf{C} \leftarrow (1 - c_1 - c_\mu)\mathbf{C} + c_1\mathbf{p}_c\mathbf{p}_c^\top + c_\mu\sum_{i=1}^\mu w_i(\mathbf{x}_{i:\lambda} - c_\mu)\mathbf{p}_c^\top + c_\mu\sum_{i=1}^\mu w_i(\mathbf{x}_{i:\lambda} - c_\mu)\mathbf{p}_c^\top
             \mathbf{m})(\mathbf{x}_{i:\lambda} - \mathbf{m})^{\top}
             Update step size:
27:
             \sigma \leftarrow \sigma \cdot \exp\left(\frac{c_{\sigma}}{d_{\sigma}}\left(\frac{\|\mathbf{p}_{\sigma}\|}{\mathbb{E}[\|\mathcal{N}(0,\mathbf{I})\|]}-1\right)\right)
28:
29: end for
```

B. Checkpoint details

30: return $\mathbf{m}_{opt} \leftarrow \mathbf{m}$

We show the details and task performance of the 16 individual checkpoint we use in the paper in table 2.

C. Additional Results and Analysis

Table 3 shows results from our three-task experiment in Sect. 5.2 and Fig. 8 includes results with search over different number of checkpoints over MBPP-IFEval from Sect. 5.3.

Fig. 9 shows fitness improvement over the course of CMA-ES. While the improvement in the fitness is not monotonic due to the sampling nature of CMA-ES, the average fitness shows a positive trend with more iterations.

Model ID	Info	MBPP	GSM8K	IFEval	MMLUPro	MUSR	MT-Bench	LBPP
Supervised	Finetuning							
1	Without MuP	58.4	76.6	64.1	29.4	18.1	8.01	30.4
2	Two stage SFT	54.4	75.3	65.7	29.0	17.9	7.74	25.5
3	Academic + Code data only. 2 epochs	63.0	79.0	66.6	31.1	14.6	7.42	29.2
4	Academic + Code data only.	63.0	68.2	59.0	31.9	10.5	7.55	26.7
5	Academic + Code data only. 2 epochs	64.0	75.7	56.5	31.1	17.0	7.68	32.9
6	Two stage SFT	60.8	76.7	65.7	29.0	17.1	7.80	31.7
7	Two stage SFT	58.2	76.6	66.9	29.0	21.1	7.59	27.3
8	Only Code	63.4	37.8	32.1	26.0	0.0	4.75	30.4
Preference	Optimization							
9	Light offline Pref	57.0	75.0	63.0	26.8	15.0	7.90	24.8
10	Data-filtered. Offline Pref	57.4	74.0	63.0	27.5	16.0	7.85	23.0
11	Different Preamble	60.4	77.0	66.0	28.3	18.0	8.08	24.8
12	Light offline Pref, with different margin scaling	56.8	81.0	72.0	28.0	19.0	8.50	28.0
13	Full offline Pref	59.0	75.0	66.0	28.9	20.0	8.22	24.2
14	Specific data mix. Of- fline Pref	58.8	77.0	69.0	28.5	20.0	8.04	26.1
15	Specific data mix. Of- fline Pref. With warmup	58.6	75.0	66.0	28.9	21.0	8.41	24.2
16	Specific data mix. Of- fline Pref	58.6	74.0	67.0	29.0	19.0	8.37	24.2

Table 2. Details and performance of the different initial checkpoints used for merging in our experiments. Supervised Finetuning and Preference Optimization models are shown with their respective performance across various benchmarks.

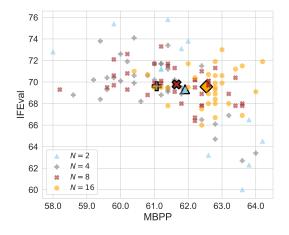


Figure 8. Merges found via CMA-ES when optimizing MBPP-MUSR tradeoffs over 2, 4, 8, and 16 checkpoints. We also show the centroid of each set of experiments. We find that optimizing over more checkpoints (8 and 16) outperforms optimization over fewer checkpoints (2,4), showing how recycling more models can outperform recycling fewer checkpoints.

D. Computational Cost Comparison

Following Kaplan et al. (2020), the total training cost in FLOPs is estimated by:

$${\rm Train\ FLOPs}\ =\ 6\,N\,B\,S,$$

where N is the number of non-embedding parameters, B is the batch size, and S is the number of training steps. In our case, $N \approx 10^{11}$, so we can estimate the cost of a single stage of supervised finetuning (SFT) and preference optimization (PO) training stages cost as follows:

$$\begin{split} \text{SFT:} \quad & 6\times 100\times 10^{9}\times 64\times 1554\ =\ 6\times 10^{16},\\ \text{PO:} \quad & 6\times 100\times 10^{9}\times 64\times 1182\ =\ 4.57\times 10^{16},\\ \text{Total:} \quad & 6\times 10^{16}+4.57\times 10^{16}\ =\ 1.057\times 10^{17}. \end{split}$$

In contrast, inference costs are substantially lower. From the same scaling laws paper, inference takes about

Inference FLOPs
$$= 2N \times \text{\#samples}$$
,

leading to the following cost (with $N\approx 10^{11})$ on different tasks:

Model	Held-in				Held out		Avg. All Tasks	
	MBPP	IFEval	GSM8K	Avg.	MT-Bench	LBPP		
Highest fitness model	56.8	72.0	81.0	69.9	7.42	30.4	49.52	
Best on MBPP	64.0	56.5	75.7	65.4	7.68	32.9	47.36	
Uniform Soup Merge best Optimized Merge	62.4	68.2	79.5	70.0	8.24	32.3	50.13	
	62.2	69.3	80.5	70.7	8.16	32.3	50.49	
	63.6	71.9	80.9	72.1	8.21	33.5	51.62	

Table 3. Comparison of model performance across different task pairs. Held-in tasks refer to tasks included in the fitness function (§ 3.2).

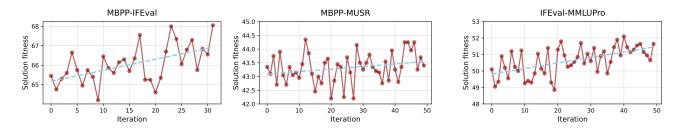


Figure 9. Fitness vs. CMA-ES iterations when optimizing tradeoffs over task pairs (see Sect. 5.1). CMA-ES explores the search space to find merge weightings with high fitness or low task tradeoffs.

Since we run the search for 50 iterations, the total compute cost for a full merge optimization over two tasks (e.g., MBPP and IFEval) amounts to:

$$50 \times 1.01 \times 10^{14} + 50 \times 1.09 \times 10^{14} = 1.05 \times 10^{16} \text{ FLOPs.}$$

That means that optimized model merging only needs at most 10% compute as that needed for a single stage of SFT + PO training. In practice, multiple SFT stages are applied over many training runs to explore different hyperparameter choices, further amplifying the overall training cost compared to search-optimized merging.