RankMean: Module-Level Importance Score for Merging Fine-tuned Large Language Models

Gabriel J. Perin[1,2], **Xuxi Chen**[2], **Shusen Liu**[3],
Bhavya Kailkhura[3], **Zhangyang Wang**[2], **Brian Gallagher**[3]
[1]University of São Paulo [2]University of Texas at Austin
[3]Lawrance Livermore National Laboratory
gabrieljp@usp.br,{xxchen,atlaswang}@utexas.edu
{liu42,kailkhura1,gallagher23}@llnl.gov

Abstract

Traditionally, developing new language models (LMs) capable of addressing multiple tasks involves fine-tuning pre-trained LMs using a wide collection of datasets, a process that often incurs significant computational expenses. Model merging emerges as a cost-effective alternative, allowing the integration of existing models fine-tuned on different tasks into a single model that performs well across all tasks, eliminating the need for additional training. In this paper, we propose RankMean, an algorithm for merging fine-tuned LMs without requiring any downstream data. RankMean determines merging coefficients based on the relative rankings of weight change magnitudes and applies these coefficients for module-wise integration of various fine-tuned models. Our experimental results demonstrate that RankMean outperforms existing baseline methods on multiple benchmarks. The code is available at github.com/VITA-Group/RankMean.

1 Introduction

The LLM research community has invested significant effort in developing robust models capable of tackling diverse downstream tasks such as mathematical reasoning (Hendrycks et al., 2021; Cobbe et al., 2021), program completion (Chen et al., 2021), and general question answering (Welbl et al., 2017). One approach to craft these models is to fine-tune large pre-trained language models specifically for the targeted downstream tasks. However, such a fine-tuning process involving a large number of parameters often incurs significant training costs (Bartoldson et al., 2023), especially when constructing models intended to address multiple tasks simultaneously, where more samples from diverse datasets need to be used in training.

Instead of fine-tuning LMs for solving multiple downstream tasks, model merging seeks to amalgamate multiple already trained model weights into a single model without the need for additional training. This approach is advantageous, especially in the context of LMs, as it leverages the community's existing efforts in fine-tuning LMs across diverse tasks, while also preserving inference time. Following this streamline of research, different model merging methods have been proposed, such as Fisher Merging (Matena and Raffel, 2022), RegMean (Jin et al., 2022) and DARE (Yu et al., 2023). However, these methods either require access to training samples, which is not always cheap and feasible in practical scenarios, or suffer from performance degradation after merging compared to their fine-tuned counterpart.

In this work, we propose RankMean, an algorithm that aims to merge multiple LMs fine-tuned on downstream tasks and overcomes the aforementioned limitation. Our algorithm considers each module of LMs as a basic unit (*e.g.*, attentions, feed-forward networks, or a whole transformer block), adopts a module-wise merging scheme, and derives the merging coefficient purely based on the relative rank of the magnitude of changes in module's weights within each model. Such rank-based coefficients are straight-forward to obtain, and do not need access to any data sample. We conduct experiments on multiple architectures and datasets and evaluate the merged models on various benchmarks. Specifically, although still suffering from some level of performance degradation when compared to the fine-tuned models, RankMean outperforms the simple averaging algorithm by clear margins and the most competitive baseline method, DARE, by up to 1.82 when applied to merge models fine-tuned for coding and mathematical tasks.

2 Related Works

Model merging aims to combine multiple models into a single one that achieves higher accuracy or robustness on multiple tasks. One of the typical ways to perform the merging is to adopt a weights averaging approach. Numerous efforts have been made to study the effectiveness of weights averaging under different settings. For example, Na-

Findings of the Association for Computational Linguistics: ACL 2024, pages 1776–1782

garajan and Kolter (2019) discovered that models trained from the same initialization on MNIST can be averaged without losing much accuracy. Frankle et al. (2020) demonstrated that the two models can be averaged without accuracy drops if the two models share some common optimization trajectories. Neyshabur et al. (2020) showed that interpolating models fine-tuned from the same pre-trained weights can attain accuracies. Wortsman et al. (2022) proposes to average the weights of multiple fine-tuned models on the same dataset with different hyperparameter configurations and often leads to improved accuracy. Ainsworth et al. (2022) studies the property of mode connectivity and proposes more effective algorithms.

Several recent works have considered the application of model merging in the context of language models (Matena and Raffel, 2022; Jin et al., 2022; Ilharco et al., 2022; Ramé et al., 2023; Yadav et al., 2023). For example, Fisher Merging (Matena and Raffel, 2022) merges models by approximating the Fisher matrices. RegMean (Jin et al., 2022) proposes to use the representation of data to calculate the merging ratio of models. DARE (Yu et al., 2023) proposes a simple algorithm that prunes and rescales the delta vector for model merging. In this work, we opt for the same philosophy and propose a model merging algorithm for fine-tuned LLMs that achieves better performance than prior art on various downstream tasks.

3 Methodology

3.1 Preliminaries and Problem Formulation

We denote a language model by $f(\cdot)$ and its weights by $\boldsymbol{\theta} \in \mathbb{R}^d$ where d indicates the hidden dimension. In this paper, our objective is to combine multiple weights that correspond to the same language model f, denoted by $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_M$ into one $\boldsymbol{\theta}_{\text{merged}}$. The series of weights, *i.e.* $\boldsymbol{\theta}_i$, can be derived from different training tasks, and the merged weights model is expected to have good performance on all these tasks.

Existing works on model merging aim to find a set of coefficients, denoted as $c_1, c_2, \ldots, c_M$, that is leveraged to construct the merged weights by $\boldsymbol{\theta}_{\text{merged}} = \sum_{i=1}^{M} c_i \boldsymbol{\theta}_i$. A simple baseline would be setting $c_i = \frac{1}{M}, i = 1, 2, \ldots, M$. More advanced methods assign different c_i to different weights. For example, Fisher Merging (Matena and Raffel, 2022) assumes $c_i \in \mathbb{R}^d$ and calculates it with the following formula: $c_i^{(j)} = F_i^{(j)} / (\sum_{i=1}^{M} F_i^{(j)})$, where $F_i = \mathbb{E}_{x \sim D_i} \mathbb{E}_{y \sim p_{\boldsymbol{\theta}_i}(y|x)} \nabla_{\boldsymbol{\theta}_i} (\log p_{\boldsymbol{\theta}_i}(y|x))^2$, and obtains the merged weights by computing $\boldsymbol{\theta}_{\text{merged}} = \sum_{i=1}^{M} c_i \odot \boldsymbol{\theta}_i.$, where $\odot$ denotes the point-wise product. Instead of using the same coefficients for all the layers for a model, RegMean (Jin et al., 2022) calculates the coefficients c_i in a layer-wise manner. More specifically, for every layer j the coefficients are calculated as $c_i^j = G_i^j / (\sum_{i=1}^{M} G_i^j)$, where G_i^j is derived based on the Gram matrix of features of training data input to the j-th layer of the i-th model. The weights in each layer of the combined models are computed based on the coefficients. More recently, DARE (Yu et al., 2023) proposed a simple algorithm to merge different finetuned LLMs without access to training data. It prunes the "delta" vector (*i.e.*, the difference of weights between the fine-tuned model and the original model), and conducts a simple average between the delta vectors. In our work, we follow a similar philosophy to merge language models that are fine-tuned on different downstream datasets without resorting to the utilization of training data, aiming to build a strong model that are capable of solving multiple downstream tasks.

3.2 RankMean: Module-Level Importance Score for Merging Models

Most existing works on model merging leverage the training data to calculate the coefficients. However, the process of iterating over samples from pre-training datasets incurs a computational cost that is prohibitively expensive. These datasets contain an extensive number of samples, with a scale reaching billions of tokens. This renders the act of processing all samples through the models exceedingly costly. Similarly, downstream task datasets usually comprise thousands of samples, making the complete traverse also highly burdensome in computation, particularly as the number of parameters in models increases. To overcome these issues, we propose RankMean, a framework for merging multiple fine-tuned LLMs without the access to original training data. RankMean consists of two important designs that are different from existing methods on LLMs: (1) it adopts module-wise merging coefficients to combine the weights from multiple fine-tuned models (derived from the same pre-trained LLM); and (2) it estimates the cross-model importance scores based on relative change within the same model through rank.

Module-level Merging. Existing literature has demonstrated that different layers vary in their contributions to a specific task (Kim et al., 2024). It

has also been discovered that the attention and the FFNs modules have diversified characteristics (Dai et al., 2022). Nonetheless, employing uniform merging coefficients across all layers neglects the consideration of the varying significance of each module. To rectify this oversight, we propose the adoption of distinct coefficients for each module of models, thereby acknowledging and accommodating the different importance inherent to their contribution to the overall model performance. We achieve this by first defining a function I that maps weight vectors into a series of coefficients. More specifically, we assume $\boldsymbol{\theta}_i = \{\boldsymbol{w}_i^1 + \Delta\boldsymbol{w}_i^1, \boldsymbol{w}_i^2 + \Delta\boldsymbol{w}_i^2 \ldots, \boldsymbol{w}_i^n + \Delta\boldsymbol{w}_i^n\}$ where $\boldsymbol{w}_i^j$ indicates the **pretrained** weights of the j-th module, $\Delta\boldsymbol{w}_i^j$ indicates the **weight changes** caused by fine-tuning and n denotes the number of modules in the model. Note that a module does not necessarily mean a linear layer but can also be a Transformer block. We define I as

$$I(\boldsymbol{w}_1^1 + \Delta\boldsymbol{w}_1^1, \ldots, \boldsymbol{w}_1^n + \Delta\boldsymbol{w}_1^n; \ldots, \boldsymbol{w}_N^n + \Delta\boldsymbol{w}_N^n) = c_1^1, c_1^2, \ldots, c_1^n; \ldots; c_N^1, \ldots, c_N^n.$$

Consequently, the merged weights can be written as $\{\sum_i c_i^1(\boldsymbol{w}_i^1 + \Delta\boldsymbol{w}_i^1), \sum_i c_i^2(\boldsymbol{w}_i^2 + \Delta\boldsymbol{w}_i^2), \ldots, \sum_i c_i^n(\boldsymbol{w}_i^n + \Delta\boldsymbol{w}_i^n)\}$.

Importance Score Estimation. Essentially, I estimates the importance of different components for model merging. A straight-forward method to estimate the importance is to use the mean magnitude of parameters, which has been proven to be effective in locating essential parameters for network pruning. However, since different models can be trained on different numbers of samples with different hyperparameters such as learning rate, they can have drastically different distributions of parameters (refer to Figure 1), which undermines the effectiveness of estimated importance and hence the final performance. Using the magnitude of changes in parameters suffer from the same problem. As an alternative, we seek to use the **relative order** of the Frobenius norm of weight changes to indicate the importance of components. The mathematical expression can be written as follows:

$$c_i^j = \frac{\text{ArgSort}(\|\Delta\boldsymbol{w}_i^j\|_F; \|\Delta\boldsymbol{w}_i^1\|_F, \ldots, \|\Delta\boldsymbol{w}_i^n\|_F)}{\sum_{k=1}^{N} \text{ArgSort}(\|\Delta\boldsymbol{w}_k^j\|_F; \|\Delta\boldsymbol{w}_k^1\|_F, \ldots, \|\Delta\boldsymbol{w}_k^n\|_F)},$$

where $\text{ArgSort}(\|\Delta\boldsymbol{w}_i^j\|_F; \|\Delta\boldsymbol{w}_i^1\|_F, \ldots, \|\Delta\boldsymbol{w}_i^n\|_F)$ indicates the ranking (i.e., the index after sorting in ascending order where 1 means the smallest) of $\|\Delta\boldsymbol{w}_i^j\|_F$ among $\|\Delta\boldsymbol{w}_i^1\|_F$, $\|\Delta\boldsymbol{w}_i^2\|_F$, $\ldots, \|\Delta\boldsymbol{w}_i^n\|_F$. Intuitively, if a weight change $\Delta\boldsymbol{w}$ ranks high within a model, it has a higher chance to receive a higher coefficient in merging. In Section 4.3, we also compare with a "reversed" form of I where we observe worse performance.

In our experiments, we consider each transformer block to be a "module" as we discover this leads to the best performance (refer to Section 4.3).

4 Experiments

4.1 Implementation Details

Architectures and Baselines. We conduct experiments with `Falcon` (Almazrouei et al., 2023) and `Llama-2` (Touvron et al., 2023). We compare with multiple baseline methods: (1) *SFT*: where we use directly the fine-tuned model for evaluation; (2) *Simple Averaging*, where we directly calculate the mean weights of multiple models; (3) *DARE* (Yu et al., 2023), which prunes and rescales the weight change vectors first, and applies the average of them to the pre-trained model.

Hyper-parameters. The models fine-tuned from `Falcon-7B` are trained with LoRA (Hu et al., 2021) for 3 epochs, using a learning rate of 5×10^{-5} with a cosine annealing learning rate scheduler and a gradient accumulation steps of 4. The modules adapted by LoRA in each layer are `self_attention.query_key_value`, `mlp.dense_h_to_4h` and `mlp.dense_4h_to_h`. The models fine-tuned from `LlaMA-v2-13B` are downloaded from the huggingface hub (see Appendix A.1). In all experiments that use DARE, we have used a masking rate of 0.9 as suggested.

Evaluation. We use different benchmarks to evaluate the model's capabilities. For coding evaluation, we evaluate models on HumanEval (Chen et al., 2021) and we report the `pass@1` metric. For mathematical reasoning, we evaluate on both GSM8K (5-shot) (Cobbe et al., 2021) and MATH (zero-shot) (Hendrycks et al., 2021) and report the accuracy. We also report the performance on AlpacaEval (Dubois et al., 2023).

4.2 Experimental Results

In the first set of experiments, we fine-tune `Falcon-7B` on three different datasets, MathInstruct (Yue et al., 2023), CodeAlpaca (Chaudhary, 2023), and SciQ (Welbl et al., 2017), covering a wide range of skills. Note that we randomly sample 20K samples from the MathInstruct dataset to maintain the number of samples at the same level across tasks. The evaluation results are presented in Table 1, where we observe that RankMean achieves

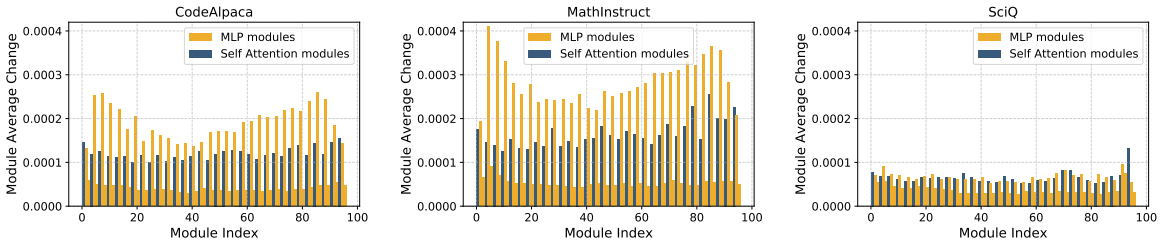

Figure 1: The average magnitude of updates to each module's weights of models fine-tuned on CodeAlpaca, MathInstruct, and SciQ. We focus on the weights in feed-forward networks (MLPs) and in self-attention modules.

the best performance. Notably, when merging models fine-tuned on maths and coding data, RankMean achieves an improvement of 0.53 on GSM8K and 0.04 on HumanEval, when compared to the second-best scores. When merging other models, we observe that RankMean achieves performance at a comparable level with DARE, the currently most competitive algorithm for LM merging.

Method	Train Sets / Source Models	Evaluation Sets GSM8K	HumanEval	SciQ
SFT	**M**	9.10	3.93	93.10
	C	5.53	8.40	94.50
	S	0.15	0.00	96.30
Simple	**M + S**	7.73	-	96.20
	C + S	-	4.02	96.30
	C + M	8.26	7.69	-
DARE	**M + S**	**8.19**	-	**96.30**
	C + S	-	**4.04**	96.30
	C + M	8.34	7.14	-
RankMean	**M + S**	6.37	-	96.20
	C + S	-	3.44	**96.40**
	C + M	**8.87**	**7.73**	-

Table 1: Performance on benchmarks by merging `Falcon-7B` fine-tuned on different training sets. **M**: MathInstruct, **C**: CodeAlpaca, **S**: SciQ. The + sign indicates the merging of two models. The best results are in **bold** and the second best are in underline.

We continue to conduct experiments on `LlaMA-v2-13B` using the setting introduced in DARE and present the performance in Table 2. It shows that RankMean out-performs substantially simple averaging on downstream benchmarks and achieves better overall performance compared to DARE when merging LMs fine-tuned on coding and mathematical data. For the other two combinations, RankMean remains a top performer, consistently achieving better performance than simple averaging and DARE especially on AlpacaEval and HumanEval.

4.3 Ablation Studies

We compare the performance of our method with different variants: (1) *Finer-Grained*: where we define the modules as weights of linear layers; (2)

Method	Train Sets / Source Models	Evaluation Sets MATH	HumanEval	AlpacaEval
No Merging	**M**	12.52	8.54	29.98
	C	0.00	24.39	22.28
	I	0.18	32.93	45.83
Simple	**M + I**	**11.10**	-	45.00
	C + I	-	**29.88**	36.09
	C + M	8.68	8.54	-
DARE	**M + I**	10.00	-	44.94
	C + I	-	26.83	37.38
	C + M	**10.14**	10.37	-
RankMean	**M + I**	9.90	-	**46.07**
	C + I	-	28.05	**38.15**
	C + M	9.54	**12.19**	-

Table 2: Performance on benchmarks by merging `LlaMA-v2-13B` adapted to different training sets. **M**: RLEIF on mathematics data, **C**: CodeAlpaca, **I**: Evol-Instruct. The + sign indicates merging between the two models. The best results are in **bold** and the second best are in underline.

Reversed, where we change the formula of c_i^j to

$$\frac{1/(1+n-\text{ArgSort}(\|\Delta \boldsymbol{w}_i^j\|_F; \|\Delta \boldsymbol{w}_i^1\|_F, \dots))}{\sum_k 1/(1+n-\text{ArgSort}(\|\Delta \boldsymbol{w}_k^j\|_F; \|\Delta \boldsymbol{w}_k^1\|_F, \dots))}.$$

The results are shown in Table 3, which demonstrates that RankMean and its finer-grained counterpart achieve comparable performance, while the reversed version under-performs the other two.

Method	Training Set	Evaluation Sets GSM8K	HumanEval	SciQ
Finer-Grained	**M + S**	**7.05**	-	**96.50**
	C + S	-	**3.66**	96.30
	C + M	8.57	**7.89**	-
Reversed	**M + S**	5.46	-	96.30
	C + S	-	2.96	96.30
	C + M	8.72	7.77	-
RankMean	**M + S**	6.37	-	96.20
	C + S	-	3.44	**96.40**
	C + M	**8.87**	7.73	-

Table 3: Performance of different variants of RankMean. The experiments are conducted with `Falcon-7B` on various downstream benchmarks.

4.4 Visualization

In Figure 1, we visualize the average magnitude of parameter changes in each module after fine-tuning `Falcon-7B` on different datasets, where we can observe that different models exhibit different distributions. Such a difference creates a bias when

using *absolute* values of weight changes for deriving merging coefficients, which we speculate to be the reason for inferior performance.

5 Conclusion

We present RankMean, an algorithm for merging language models that are fine-tuned on different downstream tasks. We propose to calculate the relative rank of the mean magnitude of weight changes in each module and use the obtained rank to derive the coefficients for model merging. Even though our algorithm still faces some level of performance degradation when compared to the fine-tuned models, we demonstrate its effectiveness and superiority in mitigating the issue on multiple benchmark datasets and architectures.

Limitations

The experiments conducted throughout this work are performed using a limited variety of tasks. Investigating if RankMean can be used to combine certain skills of the models (such as readability, conciseness, completeness, ...) could be an interesting research path, and we will explore it as future works. Besides, our algorithm explores only the merging between models with the same architectures. It would be also interesting to explore if models with different architectures can be merged.

Ethics Statement

Merging model methods allow users to cheaply combine features of different models. Even though most of the use cases are harmless, there are scenarios where these methods can facilitate the spread of malicious models. One example of such harmful use cases would be to fine-tune popular pre-trained models to be explicitly harmful and merge them with different popular task-specific fine-tuned models to create malicious versions of these models.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the Laboratory Directed Research and Development (LDRD) program under project tracking code 22-SI-007. This work is reviewed and released under: LLNL-CONF-860667. G. Jacob Perin acknowledges support from São Paulo Research Foundation (FAPESP), grants 2022/11645-1, 2023/15047-4 and 2022/15304-4.

References

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

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Brian R Bartoldson, Bhavya Kailkhura, and Davis Blalock. 2023. Compute-efficient deep learning: Algorithmic trends and opportunities. *Journal of Machine Learning Research*, 24:1–77.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

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

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*.

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

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2022. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849*.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct.

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

Vaishnavh Nagarajan and J Zico Kolter. 2019. Uniform convergence may be unable to explain generalization in deep learning. *Advances in Neural Information Processing Systems*, 32.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523.

Alexandre Ramé, Kartik Ahuja, Jianyu Zhang, Matthieu Cord, Léon Bottou, and David Lopez-Paz. 2023. Model ratatouille: Recycling diverse models for out-of-distribution generalization. In *International Conference on Machine Learning*, pages 28656–28679. PMLR.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*.

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

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Jinghan Zhang, Shiqi Chen, Junteng Liu, and Junxian He. 2023. Composing parameter-efficient modules with arithmetic operations.

A More Experimental Details

A.1 Model Architectures

The models fine-tuned from `LlaMA-v2`[1] and used in the Table 2 are `WizardLM`[2] (Xu et al., 2023), `WizardMath`[3] (Luo et al., 2023) and `llama-2-13b-code-alpaca`[4].

A.2 Dataset Statistics

Table 4: The statistics of datasets used in our experiments.

Dataset	Training Split	Testing Split
MathInstruct	20,000	-
MATH	7,500	5,000
GSM8K	7,500	1,319
CodeAlpaca	20,022	-
HumanEval	-	164
SciQ	12,679	1,000
Evol-Instruct	250,000	-
AlpacaEval	-	805

Most of the datasets we have used in our experiments are publicly available on Huggingface[5]. The usage of these datasets is consistent with their intended use. MathInstruct, MATH and GSM8K contain samples related to mathematical reasoning. CodeAlpaca and HumanEval contain samples related to programming. SciQ, Evol-Instruct and AlpacaEval are general samples.

A.3 Computational Resources.

We conduct experiments on NVIDIA RTX A6000 and NVIDIA A100 80G.

A.4 Additional results

We conduct an additional set of experiments comparing RankMean to Multi-task PEM merging (Zhang et al., 2023), which directly averages the LoRA modules without merging them to the pre-trained models first. In this experiment, Multi-task PEM merging uses uniform coefficients.

Method	Train Sets / Source Models	Evaluation Sets GSM8K	HumanEval	SciQ
Multi-task PEM Merging	**M + S**	5.31	-	96.0
	C + S	-	**3.88**	96.0
	C + M	8.11	6.95	-
RankMean	**M + S**	**6.37**	-	**96.20**
	C + S	-	3.44	**96.40**
	C + M	**8.87**	**7.73**	-

Table 5: Performance on benchmarks by merging `Falcon-7B` fine-tuned on different training sets. **M**: MathInstruct, **C**: CodeAlpaca, **S**: SciQ. The + sign indicates the merging of two models. Best results are in **bold**.

From Table 5 we can see that the performance of our algorithm still matches and outperforms these additional baselines.

[1] https://huggingface.co/meta-llama/Llama-2-13b-hf

[2] https://huggingface.co/WizardLM/WizardLM-13B-V1.2

[3] https://huggingface.co/WizardLM/WizardMath-13B-V1.0

[4] https://huggingface.co/layoric/llama-2-13b-code-alpaca

[5] https://huggingface.co/datasets/