DogeRM: Equipping Reward Models with Domain Knowledge through Model Merging

Tzu-Han Lin Chen-An Li Hung-yi Lee Yun-Nung Chen
National Taiwan University, Taipei, Taiwan
{r12944034,r13942069,hungyilee}@ntu.edu.tw y.v.chen@ieee.org

Abstract

Reinforcement learning from human feedback (RLHF) is a popular strategy for aligning large language models (LLMs) with desired behaviors. Reward modeling is a crucial step in RLHF. However, collecting paired preference data for training reward models is often costly and time-consuming, especially for domain-specific preferences requiring expert annotation. To address this challenge, we propose the **Do**main knowled**ge** merged **R**eward **M**odel (DogeRM), a novel framework that integrates domain-specific knowledge into a general reward model by model merging. The experiments demonstrate that DogeRM enhances performance across different benchmarks and provide a detailed analysis showcasing the effects of model merging, showing the great potential of facilitating model alignment.[1]

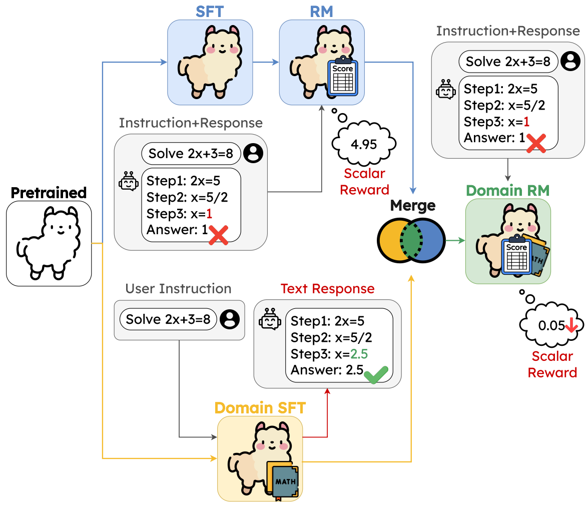


Figure 1: The framework of **DogeRM**, illustrating the merging of a general RM with a domain-specific LM to create a domain-specific RM. All icons used in this figure are sourced from https://www.flaticon.com/.

1 Introduction

Modern large language models (LLMs), such as GPT-4 (Achiam et al., 2023) and Gemini (Team et al., 2023), showcase impressive capabilities across various tasks (Gao et al., 2023; Beeching et al., 2023), with aligning their behavior with human preferences. Reinforcement learning from human feedback (RLHF) is a prominent technique for enhancing the alignment of desired behaviors in LLMs (Christiano et al., 2017; Ziegler et al., 2020; Ouyang et al., 2022). A key component of RLHF is its reward models (RMs), which assess entire sentences generated by policy models. The reward signals produced by these RMs are instrumental in adjusting the parameters of the policy models, thus directly impacting the policy models' effectiveness.

RMs are developed by training LLMs on *paired* preference data to simulate human judgment (Ouyang et al., 2022). This preference data consists of two responses to a given user input, accompanied by a human-assigned label indicating which response is more preferred. However, gathering such preference data can be costly and time-consuming due to the requirement of human annotation (Stiennon et al., 2020). This challenge becomes more pronounced when handling domain-specific preference data, as it necessitates expertise from domain specialists.

Recent developments have demonstrated the effectiveness of model merging techniques in strategically integrating multiple domain-specific models into a multi-domain model without requiring additional training (Wortsman et al., 2022; Ilharco et al., 2023). Furthermore, domain-specific SFT data is relatively more accessible compared to preference data. Moreover, many high-quality domain-specific models are available on open-source platforms (Wolf et al., 2020), which can be directly employed in the merging process. This brings us to consider a novel approach: *Is it possible to equip reward models with domain knowledge through*

[1]The source code and trained models are released at https://github.com/MiuLab/DogeRM.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 15506–15524
November 12-16, 2024

merging with domain-specific language models?

In this work, we propose **Do**main knowled**ge** merged **R**eward **M**odel (DogeRM), exploring the potential of merging a reward model trained on a general open-sourced preference dataset with a language model fine-tuned on domain-specific datasets, such as math and coding. An illustration of DogeRM is presented in Figure 1. We evaluate DogeRM using RewardBench (Lambert et al., 2024), Auto-J Eval (Li et al., 2024) and Best-of-N sampling on GSM8K (Cobbe et al., 2021) and MBPP (Austin et al., 2021). Our results demonstrate that DogeRM improves performance and can be generalized to different model architectures. We also conduct a comprehensive analysis to demonstrate the impact of model merging.

2 Related Work

Reward Modeling RMs are crucial for aligning language models with human preferences, providing proxy rewards as training signals for policy models. Previous work has employed RL algorithms with RMs to guide language models towards human preferences in various NLP tasks (Ziegler et al., 2020; Stiennon et al., 2020; Wu et al., 2021; Nakano et al., 2022; Menick et al., 2022) and instruction-following (Ouyang et al., 2022; Ramamurthy et al., 2023). In RLHF literature, RMs evaluate the quality of instructions and responses based on criteria like helpfulness and harmlessness (Bai et al., 2022) or more fine-grained objectives (Wu et al., 2023).

Several open-source paired preference datasets are available for training RMs for RLHF, such as OpenAI Summarization (Stiennon et al., 2020), HH-RLHF (Bai et al., 2022), SHP (Ethayarajh et al., 2022), Ultrafeedback (Cui et al., 2024), PKU-SafeRLHF (Ji et al., 2023), HelpSteer (Wang et al., 2024c), Nectar (Zhu et al., 2023), and UltraInteract (Yuan et al., 2024). However, most datasets are not *domain-specific*. To address this, our work focuses on merging RMs with domain-specific language models, aiming to equip RMs with domain knowledge.

Model Merging Model merging integrates multiple task-specific models into a single unified model without additional training. A straightforward approach involves averaging parameters from models fine-tuned from the same initial model (Wortsman et al., 2022). Another method employs weighted averaging of model parameters (Matena and Raffel, 2022; Jin et al., 2023).

Another innovative approach involves creating task vectors by subtracting the weights of a pre-trained model from those of the same model after fine-tuning for a specific task. This method showcases the flexibility and composability of these vectors through arithmetic operations (Ilharco et al., 2023; Yadav et al., 2024; Huang et al., 2024).

Some recent work focused on model merging to align with user preferences. They interpolated model parameters fine-tuned on diverse rewards (Rame et al., 2024a; Jang et al., 2023; Wang et al., 2024a), or merging RMs for combining different aspects of rewards (Rame et al., 2024b).

However, these methods still rely heavily on substantial domain-specific preference data to integrate domain knowledge. In contrast, our approach significantly reduces the need for such data by focusing on incorporating domain-specific knowledge into RMs through model merging.

3 Methodology

3.1 Reward Modeling

To train a reward model, we replace the decoding layer of a transformer-based pre-trained language model with a linear regression layer. This new layer projects the logits from the final transformer layer to a scalar, representing the reward of a given input.

Given an input prompt x, the chosen response y_c, and the rejected response y_r, we use the following loss function to optimize our reward model:

$$\mathcal{L}_{\text{RM}} = -\log\left[\sigma(r(x, y_c)) - \sigma(r(x, y_r))\right] \quad (1)$$

where $r(x, y_c)$ is the reward of chosen response, $r(x, y_r)$ is the reward of rejected response, and $\sigma(\cdot)$ is the logistic function.

3.2 Model Merging

Our proposed method merges the parameters of a supervised fine-tuned language model, denoted as θ^{SFT}, with those of a reward model, θ^{RM}, both initialized from the same pre-trained model θ.

We divide θ^{SFT} into three disjoint parts:

$$\theta^{\text{SFT}} = \{\theta^{\text{SFT}}_{\text{emb}}, \theta^{\text{SFT}}_{\text{trans}}, \theta^{\text{SFT}}_{\text{dec}}\} \quad (2)$$

where $\theta^{\text{SFT}}_{\text{emb}}, \theta^{\text{SFT}}_{\text{trans}}, \theta^{\text{SFT}}_{\text{dec}}$ represent the embedding, transformer, and decoding layers' parameters, respectively.

Similarly, we also divide θ^{RM} into three parts:

$$\theta^{\text{RM}} = \{\theta^{\text{RM}}_{\text{emb}}, \theta^{\text{RM}}_{\text{trans}}, \theta^{\text{RM}}_{\text{reg}}\} \quad (3)$$

Model	Reward Bench					Auto-J Eval			Best-of-16	
	Chat	Chat-Hard	Safety	Reasoning		Code	Math	Others	GSM8K	MBPP
				Code	Math					
(a) LLaMA-2 RM	95.8	47.6	**44.6**	78.9	68.2	76.2	84.4	79.2	35.3	**17.2**
(b) FT on Auto-J Math	94.7	**48.5**	44.4	79.1	68.7	76.2†	**90.2**†	79.2†	35.2	-
(c) FT on Auto-J Code	94.7	48.2	44.3	78.8	66.9	**89.3**†	84.4†	79.4†	-	**17.2**
(d) Ours (+ MetaMath)	95.8	44.5	43.5	**85.7**	79.6	79.8	87.5	79.3	**40.7**	-
(e) Ours (+ MAmmoTH)	**96.1**	44.7	43.8	84.1	**85.2**	79.8	87.5	**79.7**	40.5	-
(f) Ours (+ Code Model)	**96.1**	45.6	43.9	84.3	71.8	82.1	87.5	**79.7**	-	**17.2**

Table 1: Performance comparison across various benchmarks. Row (a) represents our base LLaMA-2 7B (Touvron et al., 2023) reward model. Rows (b) and (c) show results after fine-tuning the LLaMA-2 RM using the test data from Auto-J Eval (Li et al., 2024) Math and Code subsets, respectively. We use † to denote training accuracy, as these values are derived from benchmark testing data used during training. Rows (d) to (f) demonstrate the performance of LLaMA-2 RM when merged with MetaMath-7B (Yu et al., 2024), MAmmoTH-7B (Yue et al., 2024a), and the Code Model, each with a weight factor of $\lambda = 0.35$.

where $\theta^{\text{RM}}_{\text{emb}}, \theta^{\text{RM}}_{\text{trans}}, \theta^{\text{RM}}_{\text{reg}}$ denote the parameters for the embedding, transformer, and regression layer, respectively.

For embedding layer parameters, we apply a weighted average to common token embeddings:

$$\theta^{\text{MERGE}}_{\text{emb},t_i} = \lambda \cdot \theta^{\text{SFT}}_{\text{emb},t_i} + (1-\lambda) \cdot \theta^{\text{RM}}_{\text{emb},t_i} \tag{4}$$

where t_i is a common token to both models, θ_{emb,t_i} is the corresponding embedding, and λ is a hyperparameter controlling the weight of the SFT parameters, ranging from 0 to 1.

As for the unshared tokens, we directly use the embedding from their corresponding source model.

$$\theta^{\text{MERGE}}_{\text{emb},t_i} = \begin{cases} \theta^{\text{SFT}}_{\text{emb},t_i} & \text{If } t_i \text{ is unique to SFT} \\ \theta^{\text{RM}}_{\text{emb},t_i} & \text{If } t_i \text{ is unique to RM} \end{cases} \tag{5}$$

For the transformer layers, we perform a weighted average directly since both models are initialized from the same pre-trained model:

$$\theta^{\text{MERGE}}_{\text{trans}} = \lambda \cdot \theta^{\text{SFT}}_{\text{trans}} + (1-\lambda) \cdot \theta^{\text{RM}}_{\text{trans}} \tag{6}$$

Finally, we derive the merged reward model θ^{MERGE} by combining $\theta^{\text{MERGE}}_{\text{emb}}$, $\theta^{\text{MERGE}}_{\text{trans}}$, and the reward model's regression layer $\theta^{\text{RM}}_{\text{reg}}$:

$$\theta^{\text{MERGE}} = \{\theta^{\text{MERGE}}_{\text{emb}}, \theta^{\text{MERGE}}_{\text{trans}}, \theta^{\text{RM}}_{\text{reg}}\} \tag{7}$$

4 Experiments

4.1 Experimental Setup

Reward Model To fine-tune the backbone of our reward model, we utilize the 10k SFT split from Alpacafarm (Dubois et al., 2023). For reward modeling, we employ the UltraFeedback (Cui et al., 2024). The details of training and these datasets are presented in Appendix A and D, respectively.

Domain-Specific SFT For the math LLMs, we adopt the open-source models, MetaMath-7B (Yu et al., 2024) and MAmmoTH-7B (Yue et al., 2024a), both of which are fine-tuned from LLaMA-2-7B. For code generation LLM, since we could not find open-source models with detailed training information, we use OSS-Instruct and Magicoder-Evol-Instruct (Wei et al., 2024) to fine-tune LLaMA-2-7B ourselves. We refer to this model as the Code Model in the following sections. The details of code fine-tuning datasets and math models are presented in Appendix D, E, and F.

4.2 Evaluation

We evaluate the reward models using two benchmarks, RewardBench (Lambert et al., 2024) and Auto-J Eval (Li et al., 2024). These benchmarks provide paired instruction-completion data, with the preferred completion annotated as chosen and the other as rejected, using accuracy as the evaluation metric. We use the core set of RewardBench, focusing primarily on the reasoning category to evaluate the model's abilities in code and mathematical reasoning. For Auto-J Eval, we use pairwise testing data and categorize the dataset into three categories: code, math, and others, following Yuan et al. 2024. Additionally, to further test our reward model's effectiveness in enhancing model performance through reranking, we conduct best-of-N sampling on zero-shot prompted responses from `llama-2-7B-chat` for GSM8K (Cobbe et al., 2021) and MBPP (Austin et al., 2021). None of the models used in the experiment were trained on these data sources. Details about the models and datasets are provided in Appendix D and E, while the hyperparameters used for best-of-N sampling are outlined in Appendix B.

Additionally, in DogeRM, determining an appropriate weight factor λ depends on a small in-domain validation set. This raises an important question: *Can fine-tuning the reward model on this small dataset match or even exceed the performance of our method in the target domain?* To investigate this, we performed continuous fine-tuning of our LLaMA-2 RM using test data from Auto-J Eval and evaluated the newly fine-tuned RM on the remaining benchmarks to assess its effectiveness.

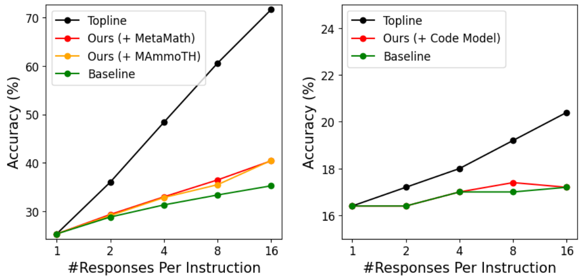
Lastly, We present results using a weight factor of $\lambda = 0.35$ for the main findings, with an analysis of the impact of λ detailed in section 4.4 and results for different values of λ presented in Appendix H.

4.3 Results

RM Benchmarks The main results of RM benchmarks are shown in Table 1. Merging the LLaMA-2 RM with MetaMath-7B (Row (d)) and MAmmoTH-7B (Row (e)) improves math performance on RewardBench by 11.4% and 17%, respectively, and coding performance by 5.2% and 5.8%, respectively. Similar enhancements are seen on Auto-J Eval, with gains in both math and coding. Merging our LLaMA-2 RM with the Code Model (Row (f)) further improves coding performance on RewardBench and Auto-J Eval by 5.4% and 6%, respectively, along with noticeable improvements in math performance on both benchmarks. Although DogeRM enhances performance in the reasoning domain, there is no significant degradation in other domains. The specific role of domain knowledge is evident, as merging with the math model leads to greater improvements in the math domain than merging with the Code model, and vice versa.

Best-of-N Sampling Figure 2 and Table 1 show the results, with accuracy improvements on GSM8K. At the best-of-16 setting, merging with math models (Rows (d) and (e)) improves GSM8K by 5%, while merging with the Code Model (Row (f)) maintains performance on MBPP without degradation. We attribute the modest improvement on MBPP to the low upper bound of reranking performance (indicated by the black line in Figure 2b), which constrains the potential gains from reranking in this task.

Fine-tuning on Small Validation Dataset Rows (b) and (c) of Table 1 show the results of fine-tuning



(a) + MetaMath/MAmmoTH on GSM8K. (b) + Code Model on MBPP.

Figure 2: Best-of-N results. Merging with domain-specific models improves reranking accuracy. Topline: Pass@N, the probability of obtaining at least one correct solution out of N responses. Baseline: LLaMA-2 RM.

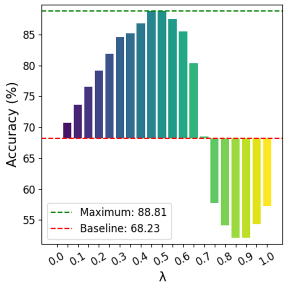
our LLaMA-2 RM on the Auto-J Eval Math and Code test subsets, respectively. While fine-tuning improved performance on Auto-J Eval, it did not generalize well to other benchmarks. In contrast, using these datasets as a validation set to determine an appropriate λ for merging resulted in better overall performance. For a detailed analysis of λ's impact across different benchmarks, see Appendix H.

4.4 Analysis

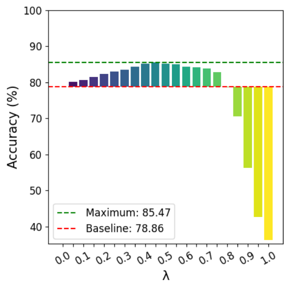
Effect of Weight Factor λ To further investigate how the weight factor λ affects our method's performance, we test various values of λ ranging from 0 to 1 in increments of 0.05, observing the performance changes across these values on RewardBench. Figure 3 shows that performance degrades when λ is large. We suggest setting λ between 0.2 and 0.5 to achieve better results.

Reward Differences We delve deeper into how model merging affects the output of reward models by examining the value of the reward signals corresponding to chosen and rejected prompts in RewardBench. Figure 3 illustrates the distribution before and after merging. In the math subset, we notice that the difference in reward scores between the chosen and rejected prompts initially increases and then decreases as λ varies from 0 to 1. Conversely, in the code subset, this difference consistently decreases. We hypothesize that this discrepancy arises because the original reward model inherently excels in the code subset.

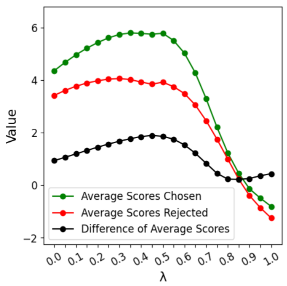
Generalizability To test the adaptability of DogeRM to different model architectures, we use an open-source Mistral-based (Jiang et al., 2023) RM (Ray2333, 2024) merging with Misrtral-based



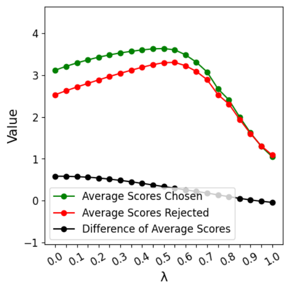
(a) + MAmmoTH on RewardBench math subset.



(b) + Code Model on RewardBench code subset.



(c) + MAmmoTH on RewardBench math subset.



(d) + Code Model on RewardBench code subset.

Figure 3: The impact of different value of λ on RewardBench math and code subsets. (a)(b): Accuracy; (c)(d): Reward difference between chosen and rejected prompts.

Model	**Reward Bench**		**Auto-J Eval**		**Best-of-16**
	Code	Math	Code	Math	GSM8K
Mistral RM	**93.5**	55.0	**88.1**	87.5	44.2
+ MAmmoTH2-Plus	92.6	**85.0**	**88.1**	**90.6**	**46.6**

Table 2: Performance of Mistral-based models on various benchmarks and best-of-16 results. Our methods show improvements across RM benchmarks and in best-of-16 sampling on GSM8K.

MAmmoTH2-7B-Plus (Yue et al., 2024b). Details of these models are presented in Appendix E. The results for $\lambda = 0.35$ in reasoning domains on RM benchmarks and best-of-N sampling on GSM8K, with N=16, are presented in Table 2. Our method improves math performance by 30% on RewardBench and 3% on Auto-J Eval. Additionally, we enhance reranking performance on GSM8K by 2.4%. These results demonstrate the adaptability of our methods to different model architectures. The results for different λ are presented in Appendix H.

Integrating Multiple Domains To evaluate DogeRM's capability of integrating knowledge from multiple domains, we experimented by merging MAmmoTH (Yue et al., 2024a) and the Code model into LLaMA-2 RM. We heuristically set the weight factors for MAmmoTH, the Code model,

Model	**Reward Bench**		**Auto-J Eval**		**Best-of-16**	
	Code	Math	Code	Math	GSM8K	MBPP
LLaMA-2 RM	78.9	68.2	76.2	84.2	35.3	**17.2**
+ Math & Code	**83.0**	**85.2**	**81.0**	**87.5**	**39.5**	17.0

Table 3: Performance of merging LLaMA-2 RM with MAmmoTH-7B and the Code model on various benchmarks and best-of-16 results. Our methods show improvements across RM benchmarks and in best-of-16 sampling on GSM8K.

and LLaMA-2 RM at 0.2, 0.2, and 0.6, respectively. The evaluation results, presented in Table 3, indicate that merging models from multiple related domains can indeed enhance performance in those domains.

5 Conclusion

In this work, we introduce a novel approach, DogeRM, which integrates domain knowledge into RM by merging it with the domain-specific SFT models. We demonstrate that DogeRM enhances performance on math and coding benchmarks and can be generalized to different model architectures. A series of analyses show that DogeRM effectively affects the reward signal corresponding to chosen and rejected prompts. The results highlight DogeRM's potential to enhance model alignment and generation verification through model merging, offering promising results across various benchmarks.

Limitations

There are several limitations in our work: (1) Our framework has been tested exclusively in the math and coding domains, leaving other areas such as medicine, finance, and law unexplored. In Section 4.4, we demonstrated the effectiveness of merging domain-specific models for math and coding into RM. However, the integration of models from multiple orthogonal domains remains an area for future investigation. (2) Our method was tested exclusively on 7B models, and we have not evaluated its performance on models of larger or smaller sizes. (3) While our framework is compatible with various merging techniques, such as TIES-Merge (Yadav et al., 2024), we have not thoroughly examined the impact of these more advanced methods. In this work, we focused on demonstrating the core idea—improving RM performance in a target domain by merging with a domain-specific language model. To keep our approach straightforward, we used weighted averaging, which, in the case of two models, can be understood as a form of task arithmetic. Despite the simplicity of this method, it has already led to notable performance gains. However, the effectiveness of more sophisticated merging techniques remains unexplored, and we leave this investigation for future work. (4) The models being merged must share the same architecture, a limitation common to most model merging algorithms (Wortsman et al., 2022; Ilharco et al., 2023; Yadav et al., 2024). Recently, evolutionary model merging (Akiba et al., 2024) has been proposed as a solution for merging models with different architectures. Investigating the merging of models with varying architectures remains a topic for future research. (5) Due to the sensitivity of RLHF to hyperparameter choices and our limited computational resources, we did not implement RLHF algorithms such as PPO (Schulman et al., 2017) or RLOO (Ahmadian et al., 2024) in this work. Exploring the integration of DogeRM within RLHF frameworks is left for future work.

Ethics Statement

While our method effectively equips reward models with domain knowledge, it does not eliminate the inherent biases within these models. Further investigation is needed to explore the impact of these inherited biases in the original reward models.

Acknowledgments

We thank the reviewers for their insightful comments. This work was financially supported by the National Science and Technology Council (NSTC) in Taiwan, under Grants 111-2222-E-002-013-MY3 and 112-2223-E002-012-MY5. We thank to National Center for High-performance Computing (NCHC) of National Applied Research Laboratories (NARLabs) in Taiwan for providing computational and storage resources. We are also grateful to Yen-Ting Lin, Wei-Lin Chen, Chao-Wei Huang and Wan-Xuan Zhou from National Taiwan University for their insightful discussions and valuable advice on Figure 1.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, Bangkok, Thailand. Association for Computational Linguistics.

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. 2024. Evolutionary optimization of model merging recipes. *Preprint*, arXiv:2403.13187.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. 2021. A general language assistant as a laboratory for alignment. *Preprint*, arXiv:2112.00861.

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

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *Preprint*, arXiv:2204.05862.

Alvaro Bartolome, Gabriel Martin, and Daniel Vila. 2023. Notus. `https://github.com/argilla-io/notus`.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. 2023. Open llm leaderboard. `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion

Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

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*.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. 2024. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36.

Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. 2022. Understanding dataset difficulty with $\mathcal{V}$-usable information. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5988–6008. PMLR.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Alex Havrilla. 2023. synthetic-instruct-gptj-pairwise. `https://huggingface.co/datasets/Dahoas/synthetic-instruct-gptj-pairwise`.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Shengding Hu, Yifan Luo, Huadong Wang, Xingyi Cheng, Zhiyuan Liu, and Maosong Sun. 2023. Won't get fooled again: Answering questions with false premises. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5626–5643, Toronto, Canada. Association for Computational Linguistics.

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

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

Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. 2023. Personalized soups: Personalized large language model alignment via post-hoc parameter merging. *arXiv preprint arXiv:2310.11564*.

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. In *Advances in Neural Information Processing Systems*, volume 36, pages 24678–24704. Curran Associates, Inc.

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

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2023. Dataless knowledge fusion by merging weights of language models. In *The Eleventh International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. 2024. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*.

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, hai zhao, and Pengfei Liu. 2024. Generative judge for evaluating alignment. In *The Twelfth International Conference on Learning Representations*.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pages 22631–22648. PMLR.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

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

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, and Nat McAleese. 2022. Teaching language models to support answers with verified quotes. *Preprint*, arXiv:2203.11147.

Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. 2024. Octopack: Instruction tuning code large language models. In *The Twelfth International Conference on Learning Representations*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

OpenAI. 2023. Openai gpt-3 api `text-davinci-003` (deprecated). Accessed: 2023-06-15. Available at: `https://beta.openai.com/docs/models/gpt-3`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. 2023. Is reinforcement learning (not) for natural language processing: Benchmarks, baselines, and building blocks for natural language policy optimization. In *The Eleventh International Conference on Learning Representations*.

Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. 2024a. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36.

Alexandre Rame, Nino Vieillard, Leonard Hussenot, Robert Dadashi, Geoffrey Cideron, Olivier Bachem, and Johan Ferret. 2024b. WARM: On the benefits of weight averaged reward models. In *Forty-first International Conference on Machine Learning*.

Ray2333. 2024. reward-model-mistral-7b-instruct-unified-feedback. Ray2333/reward-model-Mistral-7B-instruct-Unified-Feedback. Accessed: June 2024.

Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. 2024. XSTest: A test suite for identifying exaggerated safety behaviours in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5377–5400, Mexico City, Mexico. Association for Computational Linguistics.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

theblackcat102. 2023. The evolved code alpaca dataset. https://huggingface.co/datasets/theblackcat102/evol-codealpaca-v1.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl.

Haoxiang Wang, Yong Lin, Wei Xiong, Rui Yang, Shizhe Diao, Shuang Qiu, Han Zhao, and Tong Zhang. 2024a. Arithmetic control of LLMs for diverse user preferences: Directional preference alignment with multi-objective rewards. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8642–8655, Bangkok, Thailand. Association for Computational Linguistics.

Yuxia Wang, Haonan Li, Xudong Han, Preslav Nakov, and Timothy Baldwin. 2024b. Do-not-answer: Evaluating safeguards in LLMs. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 896–911, St. Julian's, Malta. Association for Computational Linguistics.

Zhilin Wang, Yi Dong, Jiaqi Zeng, Virginia Adams, Makesh Narsimhan Sreedhar, Daniel Egert, Olivier Delalleau, Jane Scowcroft, Neel Kant, Aidan Swope, and Oleksii Kuchaiev. 2024c. HelpSteer: Multi-attribute helpfulness dataset for SteerLM. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3371–3384, Mexico City, Mexico. Association for Computational Linguistics.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and LINGMING ZHANG. 2024. Magicoder: Empowering code generation with OSS-instruct. In *Forty-first International Conference on Machine Learning*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

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.

Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively summarizing books with human feedback. *Preprint*, arXiv:2109.10862.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. 2023. Fine-grained human feedback gives better rewards for language model training. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.

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

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

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024. Advancing LLM reasoning generalists with preference trees. In *AI for Math Workshop @ ICML 2024*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024a.

MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. 2024b. Mammoth2: Scaling instructions from the web. *Preprint*, arXiv:2405.03548.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2024. Evaluating large language models at evaluating instruction following. In *The Twelfth International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Yaowei Zheng, Richong Zhang, Junhao Zhang, YeYanhan YeYanhan, and Zheyan Luo. 2024. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. 2023. Starling-7b: Improving llm helpfulness & harmlessness with rlaif.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-tuning language models from human preferences. *Preprint*, arXiv:1909.08593.

A Training Details

We use V100 GPUs for training models. We spent 2 hours training the backbone of our LLaMA-2 RM, 8 hours training our LLaMA-2 RM, and 12 hours training our Code Model. Since V100 did not support `bf16`, we adopted mixed precision training (`fp16`) for both SFT and Reward Modeling.

A.1 Supervised Fine-Tuning (SFT)

We use LlamaFactory (Zheng et al., 2024) for supervised fine-tuning (SFT). For fine-tuning the backbone of our LLaMA-2 RM, we use Alpacafarm (Dubois et al., 2023) with a learning rate of 1e-5 and a batch size of 128. For code generation, we follow a training procedure similar to Wei et al. (2024). First, we use OSS-Instruct to fine-tune LLaMA-2-7B (Touvron et al., 2023) for 2 epochs. Then, we continuously fine-tune the model with Magicoder-Evol-Instruct for 1 epoch. The learning rate for both stages is 1e-5, and the effective batch size is 128.

A.2 Reward Modeling

For reward modeling, we modify the sample code provided by TRL (von Werra et al., 2020). We trained the backbone model described in the previous section on UltraFeedback (Cui et al., 2024) for 1 epoch, using a learning rate of 1e-5 and a batch size of 32.

For continuous fine-tuning of LLaMA-2 RM on Auto-J Eval (Li et al., 2024) math and code test data, we set the learning rate to 1e-6, the batch size to 8, and the number of epochs to 1.

B Best-of-N Sampling

We use vLLM (Kwon et al., 2023) to generate responses for reranking. For the GSM8K dataset (Cobbe et al., 2021), we set the temperature to 1.0, top-p to 1.0, and a maximum token length of 512. In the case of MBPP (Austin et al., 2021), we adjust the temperature to 0.1, top-p to 0.95, and maintain the same max length of 512, aligning with the hyperparameters from the `bigcode-evaluation-harness`[2] repository (Zhuo et al., 2024).

C Prompt Template

For LLaMA-2 based models, we use the same prompt template as LLaMA-2-Chat model, as shown below:

```
<s>[INST] <<SYS>>
{System Prompt}
<</SYS>>

{Instruction} [/INST] {Response}<\s>
```

We use this template for both SFT and reward modeling. For Mistral-based models, the prompt template is modified by removing the system prompt part:

[2] https://github.com/bigcode-project/bigcode-evaluation-harness

```
<s>[INST] {Instruction} [/INST] {Response}<\s>
```

The default system prompt we used in SFT and reward modeling aligns with the original system prompt for LLaMA-2-Chat model:

```
You are a helpful, respectful and honest
assistant. Always answer as helpfully as
possible, while being safe. Your answers
should not include any harmful, unethical,
racist, sexist, toxic, dangerous, or illegal
content.

Please ensure that your responses are socially
unbiased and positive in nature. If a question
does not make any sense, or is not factually
coherent, explain why instead of answering
something not correct. If you don't know the
answer to a question, please don't share false
information.
```

The system prompt used in prompting LLaMA-2-7B-Chat for Best-of-N sampling on GSM8K is:

```
You are a math problem solver. Please think
step by step and demonstrate your calculation
steps. After your reasoning steps, you should
generate the answer by following the format
starting with 'The answer is'
```

The system prompt used in prompting LLaMA-2-7B-Chat for Best-of-N sampling on MBPP is:

```
Write Python code to solve the task.
```

D Dataset Details

Alpacafarm (Dubois et al., 2023) The Alpacafarm dataset consists of 52k instructions as well as response generated by text-davinci-003 model (OpenAI, 2023) from the original Alpaca dataset (Taori et al., 2023). Alpacafarm splits the datasets into 10k 'sft' subset for instruction fine-tuning, 10k 'pref' subset for preference learning, 20k 'unlabeled subset for training such as PPO, and 2k 'val' subset for validation. We only utilize the 10k 'sft' subset for fine-tuning the backbone of our reward model.

UltraFeedback (Cui et al., 2024) This dataset consists of 64k prompts from sources including UltraChat (Ding et al., 2023), ShareGPT (Chiang et al., 2023), Evol-Instruct (Xu et al., 2024), TruthfulQA (Lin et al., 2022), FalseQA (Hu et al., 2023), and FLAN (Longpre et al., 2023). The responses are generated by a pool of different LLMs. The preferences are generated by GPT-4 (Achiam et al., 2023). In our experiment, we use a cleaned version of UltraFeedback[3] (Bartolome et al., 2023), which removes TruthfulQA contamination and uses the average of the preference ratings.

OSS-Instruct & Magicoder Evol-Instruct (Wei et al., 2024) OSS-Instruct consists of 75k synthesized data collected by prompting ChatGPT (Achiam et al., 2023) to generate a coding problem and solution based on a seed code snippet from an open-sourced platform. The Magicoder Evol-Instruct dataset, based on the work in (Luo et al., 2024), uses an open-source implementation (theblackcat102, 2023) that has been further decontaminated, resulting in 110k data points for fine-tuning. Both OSS-Instruct and Magicoder Evol-Instruct are used to fine-tune the Code Model for merging.

RewardBench (Lambert et al., 2024) RewardBench is a benchmark designed to evaluate reward models (RMs). The datasets are categorized into core sets and prior sets. The prior sets consist of testing sets from open-sourced preference dataset such as OpenAI Summarization (Stiennon et al., 2020), Anthropic Helpful split (Bai et al., 2022), Anthropic HHH (Askell et al., 2021), and Stanford Human Preference (SHP) (Ethayarajh et al., 2022).

We utilize the core sets for evaluation, which include four categories: chat, chat-hard, reasoning, and safety. The chat category collects data from AlpacaEval (Li et al., 2023) and MT Bench (Zheng et al., 2023) to assess RMs' basic ability to discern correct responses in open-ended dialogue. Chat-Hard incorporates data from MT Bench (Zheng et al., 2023) with similar ratings and LLMBar (Zeng et al., 2024) data designed to challenge LLM-based judges. The reasoning category includes math data selected from PRM800K (Lightman et al., 2024), where the prompt is the reference answer and the rejected prompt is a wrong solution generated by GPT-4 (Achiam et al., 2023). The coding data utilizes HumanEvalPack (Muennighoff et al., 2024), augmenting HumanEval (Chen et al., 2021) across six programming languages, with the prompt being the reference solution and the rejected prompt being buggy solutions. Safety category comprises data from XSTest (Röttger et al., 2024), Do-Not-Answer (Wang et al., 2024b), and

[3] https://huggingface.co/datasets/argilla/ultrafeedback-binarized-preferences-cleaned

an in-development refusals dataset at AI2, aiming to accurately test models' ability to refuse dangerous content and avoid incorrect refusals triggered by similar words.

Auto-J Eval (Li et al., 2024) Auto-J Eval's pairwise testing set includes examples from various sources: OpenAI Summarization (Stiennon et al., 2020), WebGPT (Nakano et al., 2022), Stanford SHP (Ethayarajh et al., 2022), Synthetic GPT-J (Havrilla, 2023), and PKU-SafeAlignment (Ji et al., 2023). GPT-4 (Achiam et al., 2023) serves as the annotator. The dataset consists of categories including Summarization, Exam Questions, Code, Creative Writing, Functional Writing, Rewriting, General Communication, and NLP Tasks. We exclude the tied examples and re-group the data into Code, Math (extract from Exam Questions category), and Others, following Yuan et al. 2024.

GSM8K (Cobbe et al., 2021) This dataset consists of 8.5K grade school-level math problems. We use the prompt from the testing set to perform Best-of-N sampling in a zero-shot manner.

MBPP (Austin et al., 2021) This dataset consists of 1,000 crowd-sourced Python programming problems, which are entry-level problems covering standard libraries, programming, and so on. We use the testing set to perform Best-of-N sampling in a zero-shot manner.

E Open-Source Model Details

MetaMath (Yu et al., 2024) We use the LLaMA-2-7B based model fine-tuned by the authors for merging. The MetaMath-7B models are trained on the MetaMathQA dataset, which the authors curated by bootstrapping problems from GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). According to the original paper, the model did not trained on any data from the testing set of GSM8K and MATH.

MAmmoTH (Yue et al., 2024a) We merge our RM with the MAmmoTH-7B model, a LLaMA-2-7B based model fine-tuned on the MathInstruct dataset. This dataset combines a diverse range of math problems and hybrid rationales curated by the author. According to the original paper, the model did not trained on any data from the testing set of GSM8K as well as MATH.

Mistral-RM (Ray2333, 2024) We use a Mistral-based RM, initialized from `Mistral-7B-Instruct-v0.2`, trained on diverse preference datasets to evaluate our framework's adaptability. Detailed information about the training setup can be found in the author's blog.[4]

MAmmoTH2-Plus (Yue et al., 2024b) To test the adaptability of our framework across different model architectures, we use the MAmmoTH2-7B-Plus and merge it with the Mistral RM. This model is fine-tuned from the MAmmoTH2-7B, which is fine-tuned from `Mistral-7B-Instruct-v0.2`, on public instruction tuning datasets to further enhance performance. According to the original paper, the model did not trained on any data from the testing set of GSM8K as well as MATH.

Model	MBPP	Humaneval
LLaMA-2	18.6	12.2
FT on Alpacafarm	21.0	15.9
Code Model	**26.2**	**31.7**

Table 4: Performance on two code benchmarks.

F Code Model Details

To showcase the capabilities of the fine-tuned code model, we assess its performance on two benchmarks: MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021), utilizing BigCodeBench (Zhuo et al., 2024) for evaluation. Table 4 demonstrates that our fine-tuned Code model outperforms the original Llama2 and the backbone of our reward model in the code benchmarks.

G Case Study

We conducted a case study on the predictions of the reward model. We examined the changes in chosen/rejected prompt rewards on the coding subset of RewardBench before and after merging the reward models. We identified that the problems initially predicted incorrectly but corrected after merging can be categorized into math, string operations, and bit operations. The passed and failed cases are shown in Table 5 and Table 6 respectively. Of the four problems correctly predicted before merging but incorrectly predicted after merging, two were related to list/sorting operations, while the others involved mathematical operations such as modular arithmetic and derivatives.

[4] `https://www.notion.so/abe03f9afdac42b9a5bee746844518d0`

Instruction (Math)

Write a Python function ‘max_fill(grid, capacity): import math‘ to solve the following problem: You are given a rectangular grid of wells. Each row represents a single well, and each 1 in a row represents a single unit of water. Each well has a corresponding bucket that can be used to extract water from it, and all buckets have the same capacity. Your task is to use the buckets to empty the wells. Output the number of times you need to lower the buckets.

```
Example 1:
Input:
grid : [[0,0,1,0], [0,1,0,0], [1,1,1,1]]
bucket_capacity: 1
Output: 6
Example 2:
Input:
grid : [[0,0,1,1], [0,0,0,0], [1,1,1,1], [0,1,1,1]]
bucket_capacity : 2
Output: 5
Example 3:
Input:
grid: [[0,0,0], [0,0,0]]
bucket_capacity : 5
Output: 0
Constraints:
* all wells have the same length
* 1 <= grid.length <= 10^2
* 1 <= grid[:,1].length <= 10^2
* grid[i][j] -> 0 | 1
* 1 <= capacity <= 10
```

Chosen Response

```
return sum([math.ceil(sum(arr)/capacity) for arr in grid])
```

Rejected Response

```
return sum([math.floor(sum(arr)/capacity) for arr in grid])
```

Instruction (String)

Write a Python function ‘concatenate(strings: List[str]) -> str‘ to solve the following problem: Concatenate list of strings into a single string >>> concatenate([]) ’’ >>> concatenate([’a’, ’b’, ’c’]) ’abc’

Chosen Response

```
return ''.join(strings)
```

Rejected Response

```
return ' '.join(strings)
```

Instruction (Bit Operation)

Write a Python function ‘string_xor(a: str, b: str) -> str‘ to solve the following problem: Input are two strings a and b consisting only of 1s and 0s.
Perform binary XOR on these inputs and return result also as a string. >>> string_xor(’010’, ’110’) ’100’

Chosen Response

```
def xor(i, j):
    if i == j:
        return '0'
    else:
        return '1'
    return ''.join(xor(x, y) for x, y in zip(a, b))
```

Rejected Response

```
def xor(i, j):
    if i == j:
        return '1'
    else:
        return '0'
    return ''.join(xor(x, y) for x, y in zip(a, b))
```

Table 5: Passed cases after merging.

Instruction (List/Sorting)
`Write a Python function ‘sort_array(array)‘ to solve the following problem:`
`Given an array of non-negative integers, return a copy of the given array after sorting, you will sort the given array in ascending order if the sum( first index value, last index value) is odd, or sort it in descending order if the sum( first index value, last index value) is even.`
`Note: * don’t change the given array.`
`Examples:`
`* sort_array([]) => []`
`* sort_array([5]) => [5]`
`* sort_array([2, 4, 3, 0, 1, 5]) => [0, 1, 2, 3, 4, 5]`
`* sort_array([2, 4, 3, 0, 1, 5, 6]) => [6, 5, 4, 3, 2, 1, 0]`
Chosen Response
`return [] if len(array) == 0 else sorted(array, reverse= (array[0]+array[-1]) % 2 == 0)`
Rejected Response
`return [] if len(array) == 0 else sorted(array, reverse= (array[0]+array[-1]) % 2 != 0)`

Instruction (Math)
`Write a Python function ‘derivative(xs: list)‘ to solve the following problem:`
`xs represent coefficients of a polynomial. xs[0] + xs[1] * x + xs[2] * x^2 + ....`
`Return derivative of this polynomial in the same form.`
`>>> derivative([3, 1, 2, 4, 5]) [1, 4, 12, 20] >>> derivative([1, 2, 3]) [2, 6]`
Chosen Response
`return [(i * x) for i, x in enumerate(xs)][1:]`
Rejected Response
`return [(i * x) for i, x in enumerate(xs)]`

Table 6: Failed cases after merging.

H Full Results

Full results with different values of λ on Best-of-N sampling and RM benchmarks are presented here.

H.1 Best-of-N

Figure 4 and 5 demonstrate the results of Best-of-N sampling on GSM8K when merging our LLaMA-2 RM with MetaMath-7B (Yu et al., 2024) and MAmmoTH-7B (Yue et al., 2024a), respectively. DogeRM shows consistent improvement across different models being merged.

Figure 6 shows the result of Best-of-N sampling on MBPP when merging our LLaMA-2 RM with the Code Model. While merging did not lead to a performance decline, the observed improvement is modest. We suspect this is attributable to the low upper bound of reranking performance (represented by the black line), which limits the potential gains from reranking in this task.

Finally, Figure 7 shows the results when merging the Mistral RM (Ray2333, 2024) with MAmmoTH2-7B-Plus (Yue et al., 2024b). DogeRM improves the reranking accuracy at an N=16 setting by 2.88%, indicating that our method can be generalized to different model architectures.

H.2 RewardBench

Figure 8 and 9 shows the results on different categories. We further split the reasoning category into math and coding. Merging LLaMA-2 RM with math models shows consistent improvement in both Math and Coding. The performance drop in chat-hard and safety categories can be observed.

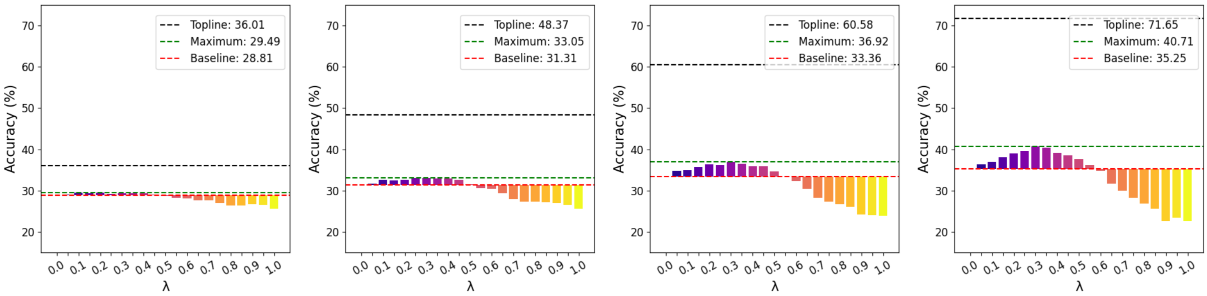
Figure 10 shows the result of merging LLaMA-2 RM with the Code Model. We observe improvements in both the Math and Coding, with a performance drop in both the chat-hard and safety categories.

Finally, Figure 11 shows the result of merging Mistral RM with MAmmoTH2-7B-Plus. We improve accuracy on the math subset by 30%, while the improvement on the coding subset is minor, likely because the original RM already achieved high accuracy on this subset. An improvement in the chat-hard category can also be observed, contrary to previous cases, but a performance degradation in the safety category is found.

We believe that the performance degradation in safety aligns with observations from Yuan et al. 2024, which indicate that removing safety data from the RM training set improves reasoning performance, suggesting that modeling safety may hurt reasoning. As for the chat-hard category, we did not observe consistent performance degradation across all combinations. A deeper investigation into this is left for future work. Despite these issues, our method can effectively equip the LLaMA-2 RM with domain-specific knowledge, a finding that holds across different domains as well as different model architectures.

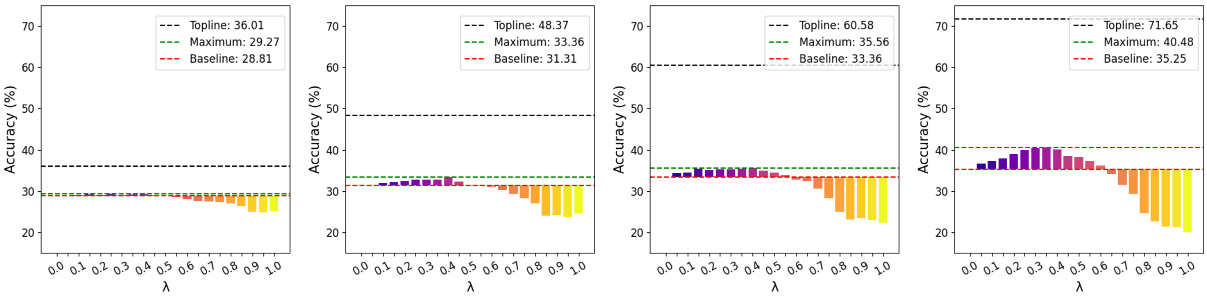
H.3 Auto-J Eval

The results of merging LLaMA-2 RM with math models are presented in Figure 12 and 13, showing improvements in both the Code and Math subsets. A similar observation can be found in Figure 14, which shows the result of merging LLaMA-2 RM with the Code Model, and Figure 15, which shows the result of merging Mistral RM with MAmmoTH-2-7B-Plus. These results support the conclusion that DogeRM can equip RMs with domain-specific knowledge.



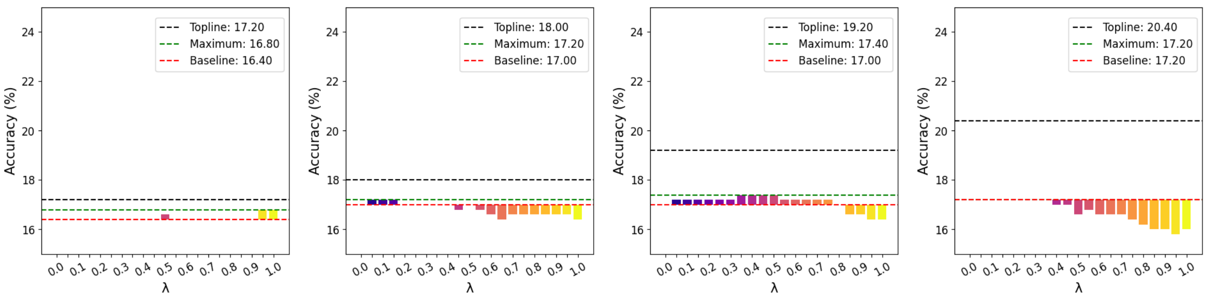
(a) Best-of-2 (b) Best-of-4 (c) Best-of-8 (d) Best-of-16

Figure 4: Full results of LLaMA-2 RM + MetaMath on GSM8K.



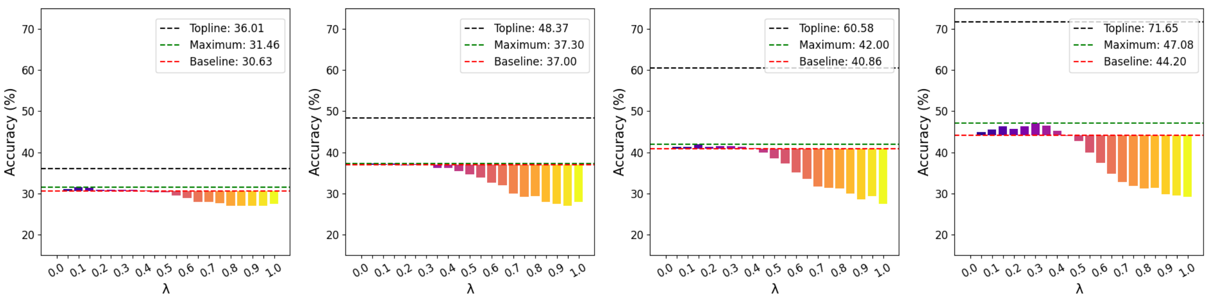
(a) Best-of-2 (b) Best-of-4 (c) Best-of-8 (d) Best-of-16

Figure 5: Full results of LLaMA-2 RM + MAmmoTH on GSM8K.



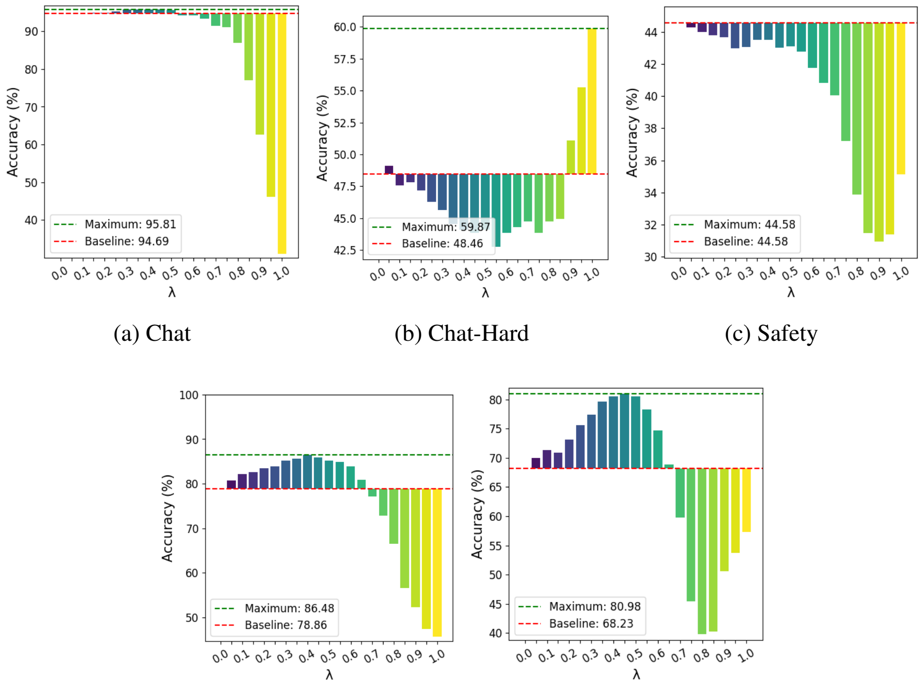
(a) Best-of-2 (b) Best-of-4 (c) Best-of-8 (d) Best-of-16

Figure 6: Full results of LLaMA-2 RM + Code Model on MBPP.



(a) Best-of-2 (b) Best-of-4 (c) Best-of-8 (d) Best-of-16

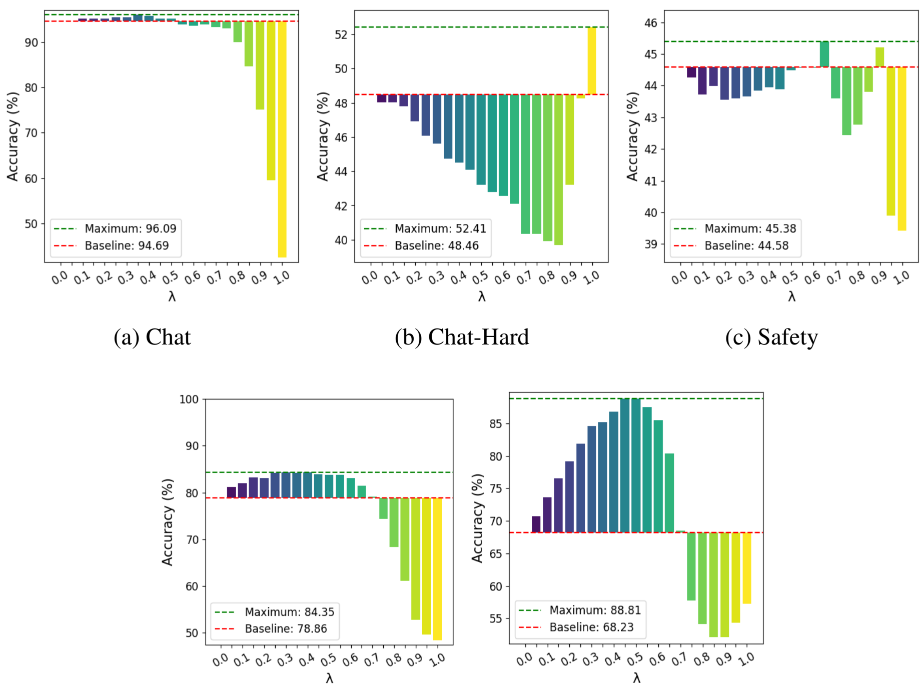
Figure 7: Full results of Mistral RM + MAmmoTH2-Plus on GSM8K.



(a) Chat (b) Chat-Hard (c) Safety

(d) Code (e) Math

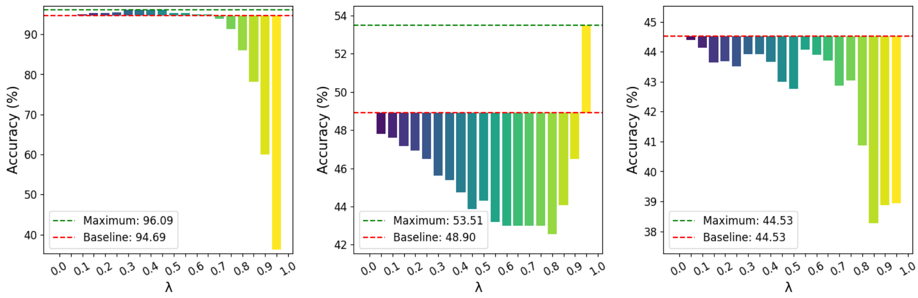
Figure 8: Full results of LLaMA-2 RM + MetaMath on Reward Bench.



(a) Chat (b) Chat-Hard (c) Safety

(d) Code (e) Math

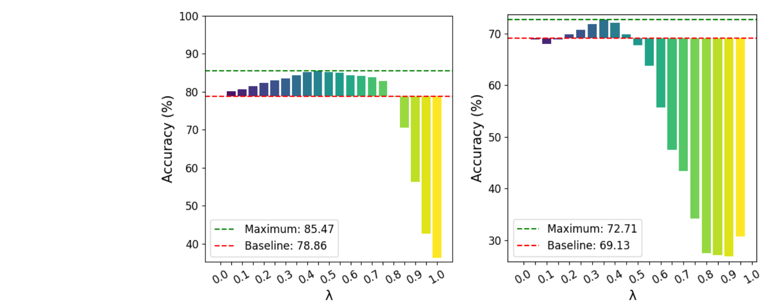
Figure 9: Full results of LLaMA-2 RM + MAmmoTH on Reward Bench.



(a) Chat (b) Chat-Hard (c) Safety

(d) Code (e) Math

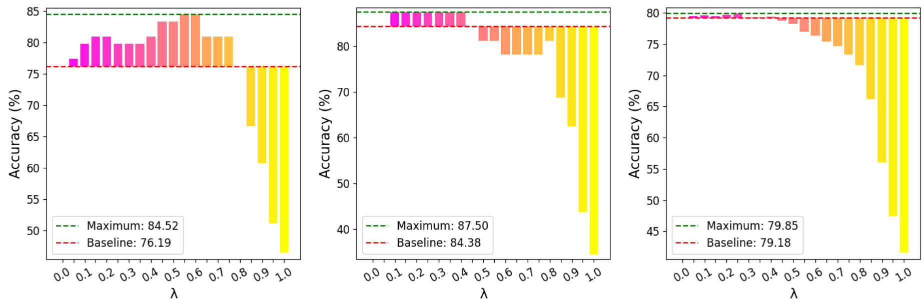
Figure 10: Full results of LLaMA-2 RM + Code Model on Reward Bench.



(a) Chat (b) Chat-Hard (c) Safety

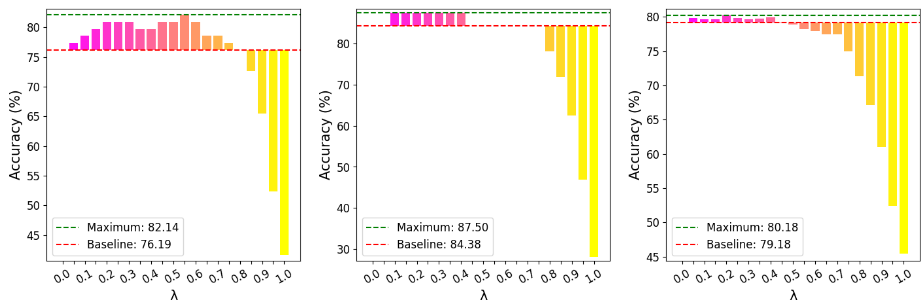
(d) Code (e) Math

Figure 11: Full results of Mistral RM + MAmmoTH2-Plus on Reward Bench.



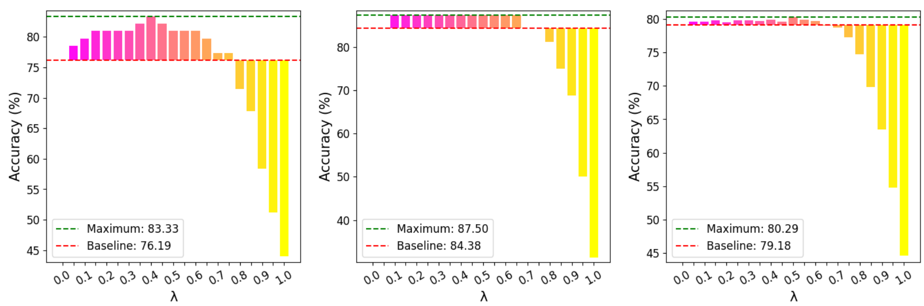
(a) Code (b) Math (c) Others

Figure 12: Full results of LLaMA-2 RM + MetaMath on Auto-J Eval.



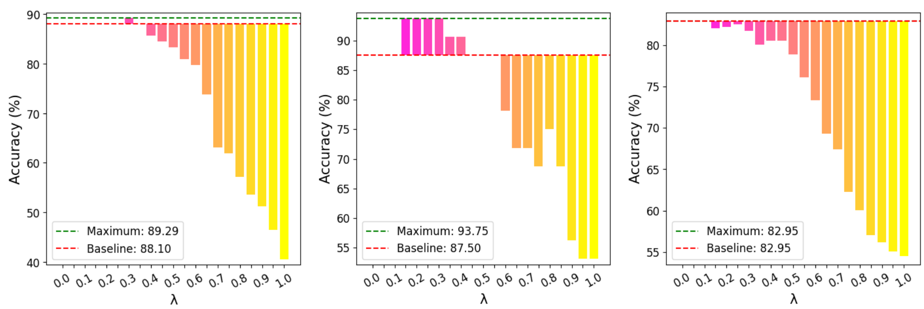
(a) Code (b) Math (c) Others

Figure 13: Full results of LLaMA-2 RM + MAmmoTH on Auto-J Eval.



(a) Code (b) Math (c) Others

Figure 14: Full results of LLaMA-2 RM + Code Model on Auto-J Eval.



(a) Code (b) Math (c) Others

Figure 15: Full results of Mistral RM + MAmmoTH2-Plus on Auto-J Eval.