Process-based Self-Rewarding Language Models

Shimao Zhang[♣*] **Xiao Liu**[★†] **Xin Zhang**[★] **Junxiao Liu**[♣] **Zheheng Luo**[◇]
Shujian Huang[♣†] **Yeyun Gong**[★]
[♣]National Key Laboratory for Novel Software Technology, Nanjing University
[◇]The University of Manchester [★]Microsoft Research Asia
smzhang@smail.nju.edu.cn, huangsj@nju.edu.cn,
xiao.liu.msrasia@microsoft.com

Abstract

Large Language Models have demonstrated outstanding performance across various downstream tasks and have been widely applied in multiple scenarios. Human-annotated preference data is used for training to further improve LLMs' performance, which is constrained by the upper limit of human performance. Therefore, Self-Rewarding method has been proposed, where LLMs generate training data by rewarding their own outputs. However, the existing self-rewarding paradigm is not effective in mathematical reasoning scenarios and may even lead to a decline in performance. In this work, we propose the Process-based Self-Rewarding pipeline for language models, which introduces long-thought reasoning, step-wise LLM-as-a-Judge, and step-wise preference optimization within the self-rewarding paradigm. Our new paradigm successfully enhances the performance of LLMs on multiple mathematical reasoning benchmarks through iterative Process-based Self-Rewarding, demonstrating the immense potential of process-based self-rewarding to achieve LLM reasoning that may surpass human capabilities. [1]

1 Introduction

Large language models (LLMs) acquire powerful multi-task language capabilities through pretraining on extensive corpus (Radford et al., 2019; Brown et al., 2020). Additionally, supervised fine-tuning (SFT) can further effectively improve the model's performance on end-tasks. However, it is found that models after SFT are prone to hallucinations (Lai et al., 2024) due to the simultaneous increasing of the probabilities of both preferred and undesirable outputs (Hong et al., 2024). Therefore, to further enhance the language capabilities of LLMs to align with human-level performance effectively, researchers often utilize human-annotated preference data for training. A representative approach is Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017), which utilizes RL algorithms and external reward signals to help LLMs learn specific preferences.

However, most reward signals rely on human annotations or reward models, which is expensive and bottlenecked by human capability and reward model quality. So the Self-Rewarding Language Models paradigm (Yuan et al., 2024) is proposed to address situations with insufficient labeled data, which integrates the reward model and the policy model within the same model. In this framework, a single model possesses the ability to both perform the target task and provide reward feedback. The model can execute different tasks based on the scenario and conduct iterative updates. This paradigm is effective in instruction-following scenarios, where the model achieves performance improvement solely through self-rewarding and iterative updates.

Although the self-rewarding algorithm performs well in the instruction-following tasks, it is also demonstrated that LLMs perform poorly on the mathematical domain data based on the existing self-rewarding algorithm. In fact, model's performance may even degrade as the number of iterations increases (Yuan et al., 2024). We notice two main limitations in the self-rewarding framework: (a) The existing self-rewarding algorithm is not able to provide fine-grained and accurate reward signals for complex reasoning tasks involving long-thought chains; (b) For a complex mathematical solution, it's hard to design the criterion for generating specific scores. It means that assigning scores to complex long-thought multi-step reasoning for LLMs is more challenging than performing pairwise comparisons, with lower consistency and agreement with humans, which is proven by the

[*]Work done during his internship at MSRA.
[†]Corresponding authors.
[1]Our code and data are available at: https://github.com/Shimao-Zhang/Process-Self-Rewarding.

results in Appendix B.

In this work, we propose the paradigm of *Process-based Self-Rewarding Language Models*, where we introduce the step-wise LLM-as-a-Judge and step-wise preference optimization into the traditional self-rewarding framework. In a nutshell, we enable the LLMs to simultaneously conduct step-by-step complex reasoning and perform LLM-as-a-Judge for individual intermediate steps. For the limitation (a) above, to get finer-grained and more accurate rewards, Process-based Self-Rewarding paradigm allows LLMs to perform step-wise LLM-as-a-Judge for the individual reasoning step. Since producing the correct final answer does not imply that LLMs can generate correct intermediate reasoning steps, it is crucial to train the model to learn not only to produce the correct final answer but also to generate correct intermediate reasoning steps. By using model itself as a reward model to generate step preference pairs data, we further perform step-wise preference optimization. For the limitation (b) above, we design a LLM-as-a-Judge prompt for step-wise pairwise comparison rather than directly assigning scores to the answer for more proper and steadier judgments based on the observations in Appendix B.

We conduct the experiments on models in different parameter sizes (7B and 72B) and test across a wide range of mathematical reasoning benchmarks. Our results show that Process-based Self-Rewarding can effectively enhance the mathematical reasoning capabilities of LLMs, which indicates that LLMs are able to perform effective self-rewarding at the step level. Our models that iteratively trained based on the Process-based Self-Rewarding paradigm demonstrate an increasing trend in both mathematical and LLM-as-a-judge capabilities. These results suggest this framework's immense potential for achieving intelligence that may surpass human performance.

2 Background

2.1 Reinforcement Learning from Human Feedback

Supervised Fine-tuning is an effective method to improve LLMs' performance across many different downstream tasks. But it has been evidenced that SFT potentially exacerbates LLMs' hallucination (Hong et al., 2024). So RLHF is further utilized to align LLMs with human preference. In the RLHF paradigm, the model is trained based on reward signals provided by external reward models and humans by reinforcement learning algorithms, such as PPO (Schulman et al., 2017), DPO (Rafailov et al., 2024), SimPO (Meng et al., 2024), and so on. Direct Preference Optimization (DPO) is a preference learning algorithm which directly uses pairwise preference data, including chosen and rejected answers for optimization. Furthermore, the step-wise preference optimization has also been investigated for long-chain reasoning and has shown great performance (Lai et al., 2024; Chen et al., 2024). In our work, we introduce the step-wise preference optimization into our Process-based Self-Rewarding paradigm for more fine-grained learning.

2.2 LLM-as-a-Judge

LLM-as-a-Judge technique has been widely used for evaluation tasks because of LLMs' scalability, adaptability, and cost-effectiveness (Gu et al., 2024). In the LLM-as-a-Judge scenarios, LLMs are prompted to mimic human reasoning and evaluate specific inputs against a set of predefined rules. To improve the performance of LLM-as-a-Judge, the LLM acting as the evaluator is trained to align with human preferences. When conducting LLM-as-a-Judge, LLMs can play many different roles depending on the given prompt. Typical applications include tasks where LLMs are prompted to generate scores (Li et al., 2023; Xiong et al., 2024), perform pairwise comparisons (Liu et al., 2024; Liusie et al., 2023), rank multiple candidates (Yuan et al., 2023), and so on. However, the investigation of the LLM-as-a-Judge for individual mathematical reasoning steps remains highly limited. In our work, we design the step-wise LLM-as-a-Judge for rewarding and analyze its performance.

2.3 Self-Rewarding Language Models

Although RLHF has been widely utilized to align LLMs with human-level performance and has achieved impressive performance, the existing methods heavily rely on high-quality reward models or human feedback, which bottlenecks these approaches. To avoid this bottleneck, Yuan et al. (2024) propose the Self-Rewarding Language Models paradigm, which uses a single model as both instruction-following model and reward model simultaneously. The iterative self-rewarding algorithm operates by having the model generate responses and reward the generated response candidates, then selecting preference pairs for training.

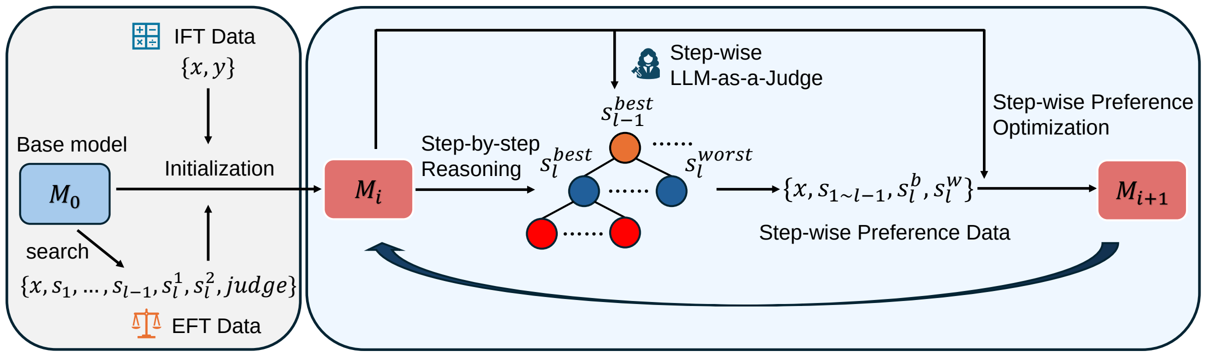


Figure 1: **Illustration of our Process-based Self-Rewarding paradigm.** (1) We get EFT data by tree-search, initial data filtering and data annotation. And we get IFT data by step segmentation. (2) The model is initialized on EFT and IFT data. (3) The model conducts step-by-step search-based reasoning and performs step-wise LLM-as-a-Judge to select the chosen step and generate the step-wise preference pair at each step. (4) We perform step-wise preference optimization on the model. (5) The model enters the next iteration cycle.

Based on this, Wu et al. (2024) further improve the judgment agreement by adding the LLM-as-a-Meta-Judge action into the self-rewarding pipeline, which allows the model to evaluate its own judgments. But the existing self-rewarding methods mainly focus on the instruction-following tasks and perform poorly in the mathematical domain data (Yuan et al., 2024). And evaluating the entire response makes it difficult for the model to learn fine-grained preference information. For some long-thought reasoning tasks, it is important to enable LLMs to focus on and learn the fine-grained reasoning step preference information.

2.4 Step-by-step Reasoning

Complex reasoning tasks are still great challenges for LLMs now. Chain-of-Thought (Wei et al., 2022) methods prompt LLMs to solve the complex problems by reasoning step by step rather than generating the answer directly, which leads to significant improvements across many reasoning tasks (Yoran et al., 2023; Fu et al., 2022; Zhang et al., 2022). Furthermore, recent studies investigate the test-time scaling paradigm which allows the LLMs to use more resources and time for inference to achieve better performance (Lightman et al., 2023) typically based on search and step selecting (Yao et al., 2024; Wang et al., 2024b). These results highlight the importance of conducting high-quality long-thought step-by-step reasoning for LLMs in solving complex reasoning problems.

3 Process-based Self-Rewarding Language Models

In this section, we propose our new Process-based Self-Rewarding Language Models pipeline. We first review the existing self-rewarding algorithm and our motivation as a preliminary study in §3.1. Then we introduce our novel paradigm for more fine-grained step-wise self-rewarding and self-evolution. The entire pipeline consists of sequential stages: model initialization (§3.2), reasoning and preference data generation (§3.3), and model preference optimization (§3.4). Finally, we provide a summarized overview of our algorithm (§3.5). We illustrate the entire pipeline in Figure 1.

3.1 Preliminary Study

Most existing preference optimization algorithms rely on reward signals from external reward models or human-annotated data. However, deploying an external reward model or getting ground truth gold reward signals from human annotators is expensive (Gao et al., 2023). Moreover, due to the inherent limitations and implicit biases of both humans and reward models, these model optimization strategies are bottlenecked (Lambert et al., 2024; Yuan et al., 2024). Thus, Self-Rewarding algorithm is proposed to mitigate this limitation by enabling the model to provide reward signals for its own outputs and perform self-improvement, showing the feasibility of achieving models that surpass human performance (Yuan et al., 2024).

There are still many aspects waiting for further research and improvement in the self-rewarding framework. The original method is primarily designed for instruction-following tasks and performs poorly on mathematical reasoning data. Step-by-step long-chain reasoning is widely used for complex mathematical reasoning, which allows the models to conduct more detailed thinking and fine-grained verification of the reasoning steps (Lightman et al., 2023; Wang et al., 2024b; Lai et al., 2024). Given the effectiveness of step-by-step reasoning, we further propose Process-based Self-Rewarding, introducing LLM-as-a-judge and preference optimization for individual steps.

3.2 Model Initialization

To perform Process-based Self-Rewarding, models need to possess two key abilities:

- Step-by-step mathematical reasoning: When faced with a complex reasoning problem, the model needs to think and reason step by step, outputting the reasoning process in a specified format. (Each step is prefixed with "Step n: ", where n indeicates the step number.)
- LLM-as-a-Judge for individual steps: The model should be able to assess the quality of the given next reasoning steps based on the existing problem and partial reasoning steps and provide a detailed explanation.

We construct data separately for the two tasks to perform cold start. Following Yuan et al. (2024), we refer to them as Instruction Fine-Tuning (IFT) data and Evaluation Fine-Tuning (EFT) data. For IFT data, we divide the given solution steps into individual steps logically without altering any information in the original solution by using OpenAI o1 (Jaech et al., 2024).

For EFT data, since there is no available step-wise LLM-as-a-Judge dataset, we first train Qwen2.5-72B (Yang et al., 2024a) on PRM800k (Lightman et al., 2023) following Wang et al. (2024a). After getting a Process Reward Model (PRM) by this, which can output a single label "+" or "-" for a reasoning step based on the question and the previous steps, we conduct Monte Carlo Tree Search (MCTS) on a policy model. We use the probability of label "+" of the above PRM to compare the relative quality of all candidate steps at the same layer, and choose the best and the worst step as a data pair. After the initial data filtering process, we use GPT-o1 to generate judgments and detailed explanations for the obtained data pairs. The pairs whose judgments align with the previous PRM assessments are selected as the final EFT data. Additionally, to enhance consistency, we evaluate each pair twice using GPT with different input orders and select only the pairs that have consistent results.

3.3 Step-by-step Long-chain Reasoning and Preference Data Generation

After the "EFT + IFT" initialization stage, the model is able to conduct both step-wise LLM-as-a-Judge and step-by-step mathematical reasoning in the specified formats. Because we conduct pairwise comparison rather than single answer grading, we utilize the following search strategy:

$$S_l = \{s_{l,1},\ s_{l,2},\ s_{l,3},\ \ldots,\ s_{l,w-1},\ s_{l,w}\} \tag{1}$$

where S_l is all candidates for the next step, l is the step number starting from 1, w is a hyperparameter to specify the search width for each step.

$$\text{Score}_{l,i} = \sum_{1 \le j \le w,\ j \ne i} \mathbf{O}(s_{l,i},\ s_{l,j} \mid x,\ s_1, s_2, \ldots, s_{l-1}) \tag{2}$$

where l is the next step number, $s_{l,i}$ indicates the i-th candidate for the next l-th step, x is the prompt, and $\mathbf{O}$ is a function that takes 1 when $s_{l,i}$ is considered better than $s_{l,j}$ and 0 otherwise.

$$s_l^{\text{best}} = S_l[\max(\text{Score}_l)] \tag{3}$$

$$s_l^{\text{worst}} = S_l[\min(\text{Score}_l)] \tag{4}$$

$$s_l = s_l^{best} \tag{5}$$

where $\max(\text{Score}_l)$ is the index of the candidate with the highest score and $\min(\text{Score}_l)$ corresponds to the lowest score. s_l is the final chosen l-th step. $(s_l^{\text{best}},\ s_l^{\text{worst}})$ will be chosen as a chosen-rejected preference pair.

This process will be repeated continuously until generation is complete. It is important to note that to enhance the effectiveness of preference data, if $\max(\text{Score}_l)$ is equal to $\min(\text{Score}_l)$, we will discard the existing s_{l-1} and $(s_{l-1}^{\text{best}},\ s_{l-1}^{\text{worst}})$ and roll back to the previous step.

3.4 Step-wise Model Preference Optimization

With preference data collected in the Section 3.3, we conduct preference optimization training on the model. We choose Direct Preference Optimization (DPO) as the training algorithm (Rafailov et al., 2024). The difference is that we conduct a more fine-grained step-wise DPO in our work. The similar method has also been investigated by Lai et al. (2024). We can calculate the training loss as:

$$A = \beta \log \frac{\pi_\theta(s_l^b \mid x, s_1, ..., s_{l-1})}{\pi_{ref}(s_l^b \mid x, s_1, ..., s_{l-1})} \quad (6)$$

$$B = \beta \log \frac{\pi_\theta(s_l^w \mid x, s_1, ..., s_{l-1})}{\pi_{ref}(s_l^w \mid x, s_1, ..., s_{l-1})} \quad (7)$$

$$\mathcal{L}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x, s_1, ..., s_l^b, s_l^w) \sim \mathcal{D}}[\log \sigma(A - B)] \quad (8)$$

where x is the prompt, $s_1, ..., s_{l-1}$ is the previous steps, s_l^b and s_l^w are the best and worst steps respectively for the l-th step, β is a hyperparameter controlling the deviation from the base reference policy, π_θ and π_{ref} are the policies to be optimized and the reference policy respectively.

After the preference optimization stage, we have the model for the next cycle. In the next iteration, we sequentially repeat the steps in §3.3 and §3.4.

3.5 Iteration Pipeline

We show the entire pipeline of our algorithm. Following Yuan et al. (2024), we refer to the model after n iterations as M_n. And we refer to the Pair-wise Preference Data generated by M_n as PPD(M_n). Then the sequence in our work can be defined as:

- M_0: The base model.
- M_1: The model obtained by supervised fine-tuning (SFT) M_0 on "EFT + IFT" data.
- M_2: The model obtained by training M_1 on PPD(M_1) using step-wise DPO.
- ······
- M_n: The model obtained by training M_{n-1} on PPD(M_{n-1}) using step-wise DPO.

In summary, we initialize the base model using well-selected step-wise LLM-as-a-Judge data (EFT) and step-by-step long-thought reasoning data (IFT). Once the model possesses the corresponding two abilities, we select preference pairs through search and reward signals provided by the model itself, and train the model using step-wise DPO. Then we iterate the model by repeatedly performing the above operations.

4 Experimental Setup

We conduct our experiments on models in different parameter sizes and several representative mathematical reasoning benchmarks. In this section, we introduce our experimental settings in detail.

Models We choose the base model from Qwen2.5-Math series (Yang et al., 2024b) in our experiments, which is one of the most popular open-source LLM series. Specifically, we choose Qwen2.5-Math-7B and Qwen2.5-Math-72B. Additionally, we choose OpenAI GPT-o1 (Jaech et al., 2024) for our initialization data processing (§3.2).

Datasets In our experiments, we mainly focus on two capabilities of the model:

- **Step-by-step Mathematical Reasoning:** We choose a subset of NuminaMath (LI et al., 2024) for IFT data construction, whose solutions have been formatted in a Chain of Thought (CoT) manner. We extract a subset of 28,889 samples and prompt GPT-o1 (Jaech et al., 2024) to logically segment the solutions into step-by-step format without altering any original content. The corresponding prompt is presented in Figure 3. And the instruction format for step-by-step long-thought reasoning is presented in Figure 4.
- **Step-wise LLM-as-a-Judge:** As described in the Section 3.2, we first filtrate some preference pairs using the trained PRM. Then we utilize GPT-o1 and get a total of 4,679 EFT data with judgments and detailed explanations. Finally we split the whole dataset into 4,167 samples as the training set and 500 samples as the test set. The instruction format for step-wise pairwise LLM-as-a-Judge is presented in Figure 5, which is following the basic format of Zheng et al. (2023).

And for mathematical task evaluation, following Yang et al. (2024b), we evaluate the LLMs' mathematical capabilities across some representative benchmarks. We choose the widely used benchmarks GSM8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). We also choose some complex and challenging competition benchmarks, including Gaokao2023En (Liao et al., 2024), Olympiadbench (He et al., 2024), AIME2024[2], and AMC2023[3].

Evaluation Metrics We use accuracy as the evaluation metric for both the mathematical performance and LLM-as-a-Judge quality. For accuracy

[2]https://huggingface.co/datasets/AI-MO/aimo-validation-aime
[3]https://huggingface.co/datasets/AI-MO/aimo-validation-amc

Model	GSM8k	MATH	Gaokao2023En	OlympiadBench	AIME2024	AMC2023	Avg.
GPT-4o	92.9	76.6	67.5	43.3	10.0	47.5	56.3
7B Base Model							
M_0	70.1	51.7	51.2	21.3	0.0	22.5	36.1
SRLM - M_1	88.2	69.0	61.6	37.6	10.0	45.0	51.9
M_2	87.6	69.4	63.9	37.2	3.3	40.0	50.2
M_3	88.5	70.0	61.3	36.7	10.0	40.0	51.1
M_4	88.3	70.2	63.9	37.6	13.3	45.0	53.1
PSRLM - M_1	88.5	69.5	61.8	36.0	6.7	45.0	51.3
M_2	88.8	69.7	63.9	36.3	**16.7**	47.5	53.8
M_3	88.5	72.2	64.7	**39.9**	10.0	50.0	54.2
M_4	**88.8**	**73.3**	**65.2**	38.7	13.3	**55.0**	**55.7**
72B Base Model							
M_0	87.5	69.7	55.3	28.9	10.0	40.0	48.6
SRLM - M_1	92.9	76.4	67.3	41.8	16.7	47.5	57.1
M_2	92.1	76.1	66.8	42.1	20.0	55.0	58.7
M_3	92.5	75.8	67.5	42.5	20.0	52.5	58.5
M_4	92.8	76.1	66.2	44.0	13.3	42.5	55.8
PSRLM - M_1	92.6	75.6	67.3	41.8	13.3	45.0	55.9
M_2	92.6	76.4	67.8	41.8	20.0	57.5	59.4
M_3	93.7	76.4	67.3	42.7	23.3	52.5	59.3
M_4	**93.7**	**76.6**	**68.1**	**44.1**	**23.3**	**57.5**	**60.6**

Table 1: Accuracy of Process-based Self-Rewarding based on 7B and 72B base models. SRLM is the self-rewarding language model algorithm as the baseline. We bold the best results for each parameter size in each benchmark.

calculation on mathematical benchmarks, we follow the implementation of Yang et al. (2024b).

Implementations For initial PRM training, we fine-tune full parameters on 128 NVIDIA A100 GPUs for 1 epoch with learning_rate=$1e-5$ and batch_size=128. For preliminary preference pairs selection, we set simulation_depth=3, num_iterations=100, Temperature=0.7, and top_p=0.95. simulation_depth indicates the maximum simulation depth. num_iterations indicates the maximum number of iterations. When training M_0 to M_1, we utilize 28,889 IFT and 4,179 EFT samples. We fine-tune LLMs' full parameters on 32 NVIDIA H100 GPUs for 3 epochs with learning_rate=$1e-6$ and batch_size=32. During the reasoning and preference data generation stage, we utilize temperature sampling which trade-off generation quality and diversity (Zhang et al., 2024). We set Temperature=0.5, top_p=0.95. The search width for each step is set to 6, and the max iteration number is set to 20. Finally, in the step-wise preference optimization, we train LLM's full parameters on 32 NVIDIA H100 GPUs for 1 epoch with learning_rate=$5e-7$ and batch_size=32. To get models from M_2 to M_4, we use 400, 800, and $1,200$ math questions for preference pairs generation respectively, which are all sampled from the train subset of NuminaMath. For all solution-scoring judgment strategy experiments, we use the same prompt template of Yuan et al. (2024). We use greedy search in evaluations.

5 Results

In this section, we report our main results on different mathematical benchmarks and conduct some discussions and analyses based on the results.

5.1 Main Results

We report the performance of M_0 to M_4 based on Qwen2.5-Math-7B and Qwen2.5-Math-72B respectively in Table 1. Our findings are as follows:

As the number of iterations increases, the overall performance of the model improves. Traditionally, external reward signals and training data are utilized for improving LLMs' performance. Our results indicate that models' overall performance on mathematical tasks significantly improves from M_1 to M_4 solely through Process-based Self-Rewarding and step-wise preference optimization without any additional guidance. This leverages the potential of LLMs for both mathematical reasoning and as evaluators.

Our fine-grained algorithm outperforms the tranditional method. After three iterations, our approach achieves superior performance compared

7B	GSM8k	MATH	Gaokao2023En	OlympiadBench	AIME2024	AMC2023
SRLM	+0.1	+1.2	+2.3	0.0	+3.3	0.0
Process-based (Ours)	+0.3	+3.8	+3.4	+2.7	+6.6	+10.0
72B	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
SRLM	-0.1	-0.3	-1.1	+2.2	-3.4	-5.0
Process-based (Ours)	+1.1	+1.0	+0.8	+2.3	+10.0	+12.5

Table 2: The results of LLMs' mathematical performance changes after all iterations from M_1 to M_4.

to method that applies rewards and conducts training on the entire response. Given that the initialization with different EFT data lead to different M1 fiducial performance in the two methods, we also report the performance changes from M1 to M4 after multiple iterations in Table 2, which reflects the algorithm's effectiveness and stability in improving the model's mathematical capabilities. Our method achieves more stable and effective improvements across all benchmarks. On one hand, using step-wise preference data enables the model to focus on more fine-grained information; on the other hand, conducting LLM-as-a-Judge on individual steps helps the model more easily detect subtle differences and errors.

The models show noticeable improvements on some complex tasks. For some complex and highly challenging benchmarks, such as MATH, AIME2024, and AMC2023, LLMs' performance show significant improvement. Complex problems require multi-step, long-thought reasoning. Our method effectively leverages the model's existing knowledge to optimize the individual intermediate reasoning steps, achieving favorable results.

Our method remains effective across models of different parameter sizes. We validate our method on both 7B and 72B LLMs to strengthen our conclusions. We find performance improvements across models of different parameter sizes on multiple mathematical tasks through Process-based Self-Rewarding. We also find that the 72B model gains more stable improvements compared to the 7B model, whose mathematical reasoning and LLM-as-a-Judge capabilities are stronger.

Overall, we can find that the models iterating based on the Process-based Self-Rewarding paradigm achieve significant improvements across multiple mathematical tasks, outperforming the traditional self-rewarding method.

Model	7B	72B
M_0 (3-shot)	57.2	73.4
M_1	92.8 (↑)	95.6 (↑)
M_2	91.6 (↓)	95.8 (↑)
M_3	92.0 (↑)	95.2 (↓)
M_4	92.2 (↑)	95.6 (↑)

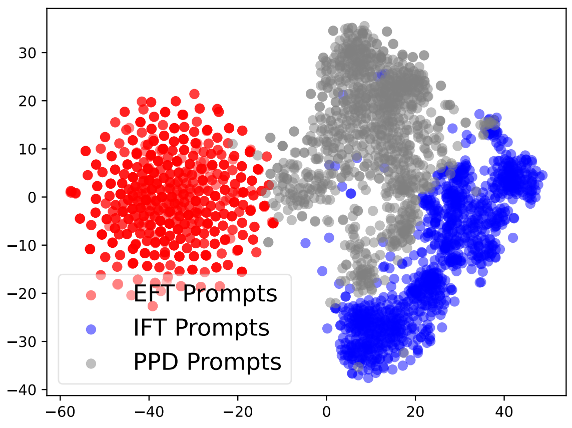
Table 3: Judgment accuracy in step-wise LLM-as-a-Judge. We report the results of models with different parameter sizes. Additionally, we use arrows to indicate the changes in accuracy during the iterations.

5.2 Further Analysis

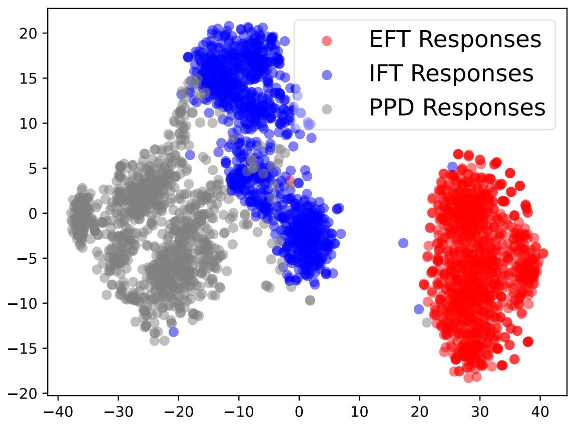
Based on the above results, we conduct more analysis and observations of the pipeline.

Step-wise LLM-as-a-Judge Capability. We evaluate the LLMs' ability to accurately assess reasoning steps as a reward model during the iterative process. We test the model on the test set including 500 samples (§3.2). We report the results in Table 3. As shown in the table, LLMs achieve strong reward model performance after initialization with a small amount of EFT data, which indicates the immense potential of LLMs for step-wise LLM-as-a-Judge with CoT reasoning. Additionally, we can observe that, under the same conditions, the larger model exhibits stronger capabilities as a reward model than the smaller one.

Additionally, although we mix EFT data and IFT data for initialization and introduce no additional LLM-as-a-Judge data during subsequent iterations, the LLMs' capabilities to perform LLM-as-a-Judge as a reward model are still good. Furthermore, a consistent pattern is observed across different models where evaluation accuracy initially increases, then decreases, and finally rises again. Based on the analysis above, initially, LLMs gain strong evaluation capabilities through training on EFT data. And there is a temporary decline (but very slight) due to training on mathematical data. Ultimately, as the model's mathematical abilities improve, its ability to evaluate mathematical reasoning steps also increases.



(a) Prompt Distributions



(b) Response Distributions

Figure 2: The data distribution of prompts and responses in EFT (red), IFT (blue) and PPD (grey) data.

Iterations	Step Num		Step Length	
	GSM8k	MATH	GSM8k	MATH
M_1	5.89	8.41	47.79	61.00
M_2	5.55	7.64	51.19	67.17
M_3	5.10	6.30	57.75	80.46
M_4	4.87	5.54	62.86	96.63

Table 4: Statistics of step number and step length on GSM8k and MATH benchmarks based on 72B models. The full results are reported in Appendix A.

Data Distribution Analysis. Following Yuan et al. (2024), we also analyze the distribution of different data. We utilize Bert (Devlin, 2018) for embedding and t-SNE (Van der Maaten and Hinton, 2008) based on the implementation of Poličar et al. (2024) for visualization. We present the results in Figure 2. For prompts, the distributions of EFT data and IFT data do not overlap, allowing the model to distinctly learn two different task patterns. For models' responses, we can find the similar phenomenon that the distribution of PPD and IFT responses is distinct from EFT's, which reduces the mutual interference between LLMs' two capabilities during iteration. This allows the model's ability to perform LLM-as-a-Judge to improve alongside its mathematical ability finally, without being overly influenced by the training data itself.

Step Number and Length of Responses. Step-by-step reasoning is important for LLMs to solve complex reasoning tasks. Therefore, we conduct statistical analysis on the reasoning steps during iterations. As shown in Table 4, for the same model, more difficult problems require more reasoning steps and longer step lengths. As the iterations progress, the step number across different tasks decreases, while the length of each step increases. This indicates that performing Process-based Self-Rewarding encourages the model to generate longer and higher-quality single reasoning steps, which helps to reach final answers with fewer steps. Additionally, this behavior is also related to LLMs' preferences when performing LLM-as-a-Judge evaluations. More results are in Appendix A.

Strategy	Greedy Search	Test-time Scaling
M_1	55.9	58.2
M_4	60.6	62.4

Table 5: The average results of 72B model on all benchmarks using greedy search or test-time scaling. The full results are reported in Table 9.

Test-time Scaling with Process-based Self-Rewarding Language Models. In the test-time scaling, LLMs conduct step search and select based on the rewards from PRM. Although we don't primarily focus on the test-time scaling performance in our work, LLMs in the Process-based Self-Rewarding paradigm naturally have the ability to perform test-time scaling based on self-rewarding. We perform 6 generations for each step with the temperature of 0.5 and select the best one. The results we report in Table 5 indicate that the model achieves better performance through test-time scaling compared to generating directly. Additionally, the model's performance with test-time scaling improves after iterations from M_1 to M_4, which corresponds to the uptrend of the model's mathematical abilities and LLM-as-a-Judge capabilities.

6 Conclusion

We propose a novel paradigm, Process-based Self-Rewarding Language Models, that enables LLMs to perform step-by-step long-thought mathematical reasoning and step-wise LLM-as-a-Judge simultaneously. Given the characteristics of complex math reasoning tasks, we introduce the step-by-step reasoning, step-wise LLM-as-a-Judge and step-wise preference optimization technique into the framework. Our results indicate that Process-based Self-Rewarding algorithm outperforms the original Self-Rewarding on a variety of complex mathematical reasoning tasks, showing potential of stronger reasoning ability better than human in the future.

7 Limitations

We aim to draw more attention to the study of adapting the self-rewarding paradigm to the complex mathematical reasoning tasks, which allows for the possibility of continual improvement beyond the human preferences. Although our new Process-based Self-Rewarding algorithm has shown effective improvements across different mathematical reasoning tasks, there are still some limitations waiting for further research. Although we successfully enable the model to perform effective step-wise LLM-as-a-Judge with a small amount of EFT data, the basic capabilities of initialized M_1 model directly influence the effectiveness of subsequent process-based self-rewarding. Utilizing more high-quality data to initialize LLMs more adequately may lead to stronger performance.

Additionally, due to the limited resources, we only conduct the process-based self-rewarding experiments from M_1 to M_4. Building on this, conducting experiments with more iterations to explore the impact of iteration count on LLMs' performance can help us better understand and utilize the process-based self-rewarding method.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Step-level value preference optimization for mathematical reasoning. *arXiv preprint arXiv:2406.10858*.

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*.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

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*.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11170–11189.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*.

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*.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Hai Zhao, and Pengfei Liu. 2023. Generative judge for evaluating alignment. *arXiv preprint arXiv:2310.05470*.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. Mario: Math reasoning with code interpreter output–a reproducible pipeline. *arXiv preprint arXiv:2401.08190*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Yinhong Liu, Han Zhou, Zhijiang Guo, Ehsan Shareghi, Ivan Vulić, Anna Korhonen, and Nigel Collier. 2024. Aligning with human judgement: The role of pairwise preference in large language model evaluators. *arXiv preprint arXiv:2403.16950*.

Adian Liusie, Potsawee Manakul, and Mark JF Gales. 2023. Zero-shot nlg evaluation through pairware comparisons with llms. *arXiv preprint arXiv:2307.07889*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *arXiv preprint arXiv:2405.14734*.

Pavlin G. Poličar, Martin Stražar, and Blaž Zupan. 2024. opentsne: A modular python library for t-sne dimensionality reduction and embedding. *Journal of Statistical Software*, 109(3):1–30.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

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

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, et al. 2024a. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024b. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. 2024. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. *arXiv preprint arXiv:2407.19594*.

Tianyi Xiong, Xiyao Wang, Dong Guo, Qinghao Ye, Haoqi Fan, Quanquan Gu, Heng Huang, and Chunyuan Li. 2024. Llava-critic: Learning to evaluate multimodal models. *arXiv preprint arXiv:2410.02712*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024b. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought. *arXiv preprint arXiv:2304.13007*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. 2023. Rrhf: Rank responses to align language models with human feedback without tears. *arXiv preprint arXiv:2304.05302*.

Shimao Zhang, Yu Bao, and Shujian Huang. 2024. Edt: Improving large language models' generation by entropy-based dynamic temperature sampling. *arXiv preprint arXiv:2403.14541*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Step Number and Step Length Statistics

We report the full results of step number and step length across all benchmarks on the 7B and 72B models here. The 7B results are reported in Table 7. And the 72B results are reported in Table 8.

B Solution Scoring v.s. Step-wise Pairwise Comparison

We evaluate the GPT-4o's (Hurst et al., 2024) consistency and agreement with humans on two different LLM-as-a-Judge strategies for complex mathematical reasoning tasks, including assigning scores to the answers and performing pairwise comparison between two individual reasoning steps. We report the results in Table 6. Our results indicate that for the complex mathematical reasoning task, step-wise pairwise comparison has better consistency and agreement with humans than solution scoring. It is highly challenging for LLMs to assign a proper and steady score to a complex long-thought multi-step solution.

C Prompt Templates

We list the prompt templates we used in our work here. The prompt we use for constructing step-by-step formatted reasoning is shown in Figure 3. And the prompts we used for step-by-step long-thought mathematical reasoning and step-wise LLM-as-a-Judge are shown in Figure 4 and Figure 5 respectively.

Judge Strategy	**Consistency**	**Agreement**
Step-wise Pairwise Comparison	0.84	0.88
Solution Scoring	0.72	0.32

Table 6: The consistency and agreement with human evaluation of step-wise pairwise comparison and solution scoring.

Step Num	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
M_1	5.91	9.35	8.68	11.75	7.97	11.18
M_2	5.24	8.03	7.43	9.54	7.03	9.85
M_3	4.50	6.43	5.84	7.36	7.13	6.9
M_4	4.09	5.21	5.11	6.14	6.4	5.53
Step Length	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
M_1	48.59	61.61	69.74	103.95	100.43	76.13
M_2	54.02	70.04	85.26	108.26	114.27	115.29
M_3	63.36	89.68	99.59	127.97	118.67	109.45
M_4	73.64	113.14	118.02	142.69	138.18	127.18

Table 7: Statistics of step number and step length on different methematical benchmarks based on 7B models.

Step Num	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
M_1	5.89	8.41	8.34	10.21	8.23	9.95
M_2	5.55	7.64	7.34	9.05	7.37	9.75
M_3	5.10	6.30	5.99	6.54	7.07	6.55
M_4	4.87	5.54	5.36	5.75	6.33	6.1
Step Length	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
M_1	47.79	61.00	69.72	95.38	104.97	79.36
M_2	51.19	67.17	78.00	101.93	118.08	86.88
M_3	57.75	80.46	91.21	122.53	118.61	108.95
M_4	62.86	96.63	106.28	134.62	133.66	113.60

Table 8: Statistics of step number and step length on different methematical benchmarks based on 72B models.

Setting	**GSM8k**	**MATH**	**Gaokao2023En**	**OlympiadBench**	**AIME2024**	**AMC2023**
M_1 Greedy Search	92.6	76.0	66.2	41.8	13.3	45.0
M_4 Greedy Search	93.7	76.6	68.1	44.1	23.3	57.5
M_1 Test-time Scaling	94.5	79.1	64.9	41.6	16.7	52.5
M_4 Test-time Scaling	94.5	79.3	68.3	43.7	23.3	65.0

Table 9: The full results of greedy search and test-time scaling on 72B model.

```
There is a math problem and its corresponding solution. Please divide the
given solution into individual steps logically. Use "Step n: " before
each step to distinguish between different steps, where n is a positive
integer starting from 1, representing the current step number. Only
divide the steps without altering any information in the original
solution. Please output only the divided solution steps in the format
mentioned above, and do not include any additional information. Do not
omit the final answer that is placed in boxed.

[The Start of Question Provided]
{question}
[The End of Question Provided]

[The Start of Solution Provided]
{solution}
[The End of Solution Provided]
```

Figure 3: The prompt for converting the the given solution into step-by-step format logically without altering any information in the original solution.

```
Let's think step by step and solve the following math problem. Use "Step
n: " before each step to distinguish between different steps, where n is
a positive integer starting from 1, representing the current step number.
Put your final answer in boxed.

Problem: {problem}
```

Figure 4: The prompt for LLMs conducting step-by-step long-thought reasoning.

```
Please act as an impartial judge and evaluate the quality of two next
reasoning steps provided by two AI assistants to the question and
partial reasoning steps displayed below. Your evaluation should
consider correctness and helpfulness. You will be given assistant A’s
answer, and assistant B’s answer. Your job is to evaluate which
assistant’s answer is better. You should compare the two responses and
provide a detailed explanation. Avoid any position biases and ensure
that the order in which the responses were presented does not influence
your decision. Do not allow the length of the responses to influence
your evaluation. Do not favor certain names of the assistants. Be as
objective as possible. After providing your explanation, output your
final verdict by strictly following this format: "[[A]]" if assistant A
is better, and "[[B]]" if assistant B is better.

[Question and Intermediate Reasoning Steps Provided]
{Question and Partial Reasoning Steps}

[The Start of Assistant A’s Next Reasoning Step]
{Step A}
[The End of Assistant A’s Next Reasoning Step]

[The Start of Assistant B’s Next Reasoning Step]
{Step B}
[The End of Assistant B’s Next Reasoning Step]
```

Figure 5: The prompt for LLMs conducting step-wise LLM-as-a-Judge. We create this prompt template following the basic pattern of Zheng et al. (2023).