Beyond the First Error: Process Reward Models for Reflective Mathematical Reasoning

Zhaohui Yang[1,2,3]*, Chenghua He[1]*, Xiaowen Shi[4], Shihong Deng[3]†, Linjing Li[1,2]†, Qiyue Yin[1]†, Daxin Jiang[3],
[1]Institute of Automation, Chinese Academy of Sciences
[2] School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]StepFun, China [4]Meituan
yangzhaohui2023@ia.ac.cn hechenghua23@mails.ucas.ac.cn
dshnightmare@gmail.com qyyin@nlpr.ia.ac.cn

Abstract

Many studies focus on data annotation techniques for training effective PRMs. However, current methods encounter a significant issue when applied to long CoT reasoning processes: they tend to focus solely on the first incorrect step and all preceding steps, assuming that all subsequent steps are incorrect. These methods overlook the unique self-correction and reflection mechanisms inherent in long CoT, where correct reasoning steps may still occur after initial reasoning mistakes. To address this issue, we propose a novel data annotation method for PRMs specifically designed to score the long CoT reasoning process. Given that under the reflection pattern, correct and incorrect steps often alternate, we introduce the concepts of **Error Propagation** and **Error Cessation**, enhancing PRMs' ability to identify both effective self-correction behaviors and reasoning based on erroneous steps. Leveraging an LLM-based judger for annotation, we collect 1.7 million data samples to train a 7B PRM and evaluate it at both solution and step levels. Experimental results demonstrate that compared to existing open-source PRMs and PRMs trained on open-source datasets, our PRM achieves superior performance across various metrics, including search guidance, BoN, and F1 scores. Compared to widely used MC-based annotation methods, our annotation approach not only achieves higher data efficiency but also delivers superior performance. Detailed analysis is also conducted to demonstrate the stability and generalizability of our method.

1 Introduction

Mathematical reasoning has become a crucial metric for evaluating the intelligence of LLMs (Achiam et al., 2023; Team et al., 2024; Touvron et al., 2023), garnering substantial attention from researchers in recent years. While numerous studies have focused on enhancing LLMs' mathematical reasoning capabilities (Wei et al., 2022; Liu et al., 2024), LLMs continue to exhibit limitations in practical applications, including calculation errors, flawed derivations, and logical errors. PRMs help address these challenges by providing fine-grained evaluation signals for the intermediate steps in the LLM reasoning process, indicating the correctness of each step.

In the short chain-of-thought (CoT) pattern, LLMs lack self-reflection abilities. This means that once a mistake is made, all following steps are likely to be wrong. The PRM only needed to identify correct-to-correct and correct-to-incorrect transitions. Therefore, conventional PRM data construction methods (Lightman et al., 2023; Luo et al., 2024) use all steps from correct solutions, but only use steps up to the first error step in incorrect solutions. However, as shown in Appendix D, long CoT LLMs often make mistakes during reasoning, but they can later correct themselves or come up with alternative solutions. Therefore, capturing this incorrect-to-correct pattern is crucial for PRMs to better evaluate the intermediate steps in long CoT reasoning.

Currently, the Monte Carlo (MC) method (Wang et al., 2024a) is widely used for automatically labeling intermediate steps. However, when dealing with long CoT patterns, the MC method has the following drawbacks: (1) Inaccurate labeling of intermediate reasoning steps. MC method rolls out multiple paths from an intermediate step. If one path reaches the correct answer, it assumes the step is correct. However, long CoT reasoning processes often involve self-correction or reflection, so the final answer doesn't always reflect the correctness of intermediate steps. (2) High computational cost. The MC method requires multiple rollouts for each intermediate step. In the long CoT pattern, numerous intermediate steps and lengthy rollouts can greatly increase the computational burden.

To address the shortcomings of current PRM de-

*Equal contribution.
†Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 4711–4728
November 4-9, 2025 ©2025 Association for Computational Linguistics

signs and MC methods in long CoT scenarios, we propose a new data annotation approach. First, we introduce two fundamental rules specifically for long CoT: **Error Propagation** and **Error Cessation**. These rules capture two key patterns: reasoning based on incorrect steps and self-correction after making mistakes. We then incorporate these rules into the LLM judge's prompt to guide the model in annotating intermediate steps of the reasoning process. Finally, we use the annotated data to train the PRM.

Experimental results demonstrate that compared to existing open-source PRMs and PRMs trained on open-source datasets, our PRM achieves the best performance at both the solution level and the step level. At the solution level, we use the traditional Best-of-N (BoN) method to see how well PRMs can select the best answer from several options. Since BoN doesn't make full use of the process reward signals that PRMs provide, we also propose a new metric that measures whether incorporating these signals into step-level search increases the probability of discovering the correct solution. At the step level, we establish a test set through the cross-validation of two distinct annotation methods (o1 model (OpenAI, 2024) and manual annotation), enabling rigorous assessment of PRMs in evaluating the correctness of individual solution steps. Across both solution-level and step-level evaluation metrics on the MATH500 and AIME24, our PRM consistently outperforms all baselines. Furthermore, we conduct a comprehensive comparison with the MC-based data annotation method, demonstrating that our method not only achieves greater data efficiency but also delivers superior performance. An in-depth analysis of our method is also conducted to further demonstrate its stability and generalizability.

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first to present a PRM data construction method for long CoT reasoning. We introduce context-aware **Error Propagation** and **Error Cessation** mechanisms to effectively capture both wrong-to-wrong and wrong-to-right patterns.
- We evaluate our method using both solution-level and step-level metrics, demonstrating its effectiveness in evaluating overall solutions as well as individual reasoning steps.
- Additional analysis demonstrates that our method has notable advantages in robustness, data efficiency, and generalization ability.

2 Related Work

Long Chain-of-Thought Reasoning Language Models LLMs have demonstrated remarkable reasoning capabilities for complex tasks. One pivotal method to improve the reasoning ability of LLMs is Chain-of-Thought (CoT) (Wei et al., 2022; Wang et al., 2025, 2024b), which significantly improves performance by guiding LLMs to generate intermediate reasoning steps. Initial research (Wei et al., 2022; Kojima et al., 2022; Zhang et al., 2022) on CoT mainly focuses on developing effective prompt engineering techniques. Openai o1 (OpenAI, 2024) is the first to introduce inference time scaling law, which employs reinforcement learning to encourage models to generate additional reasoning tokens, thereby overcoming more challenging tasks. In long CoT paradigm, LLMs can decompose problems, explore multiple pathways, and automatically correct reasoning errors. Several efforts (Team, 2024a; Guo et al., 2025; Team, 2025; Muennighoff et al., 2025) have successfully replicated this powerful reasoning ability. However, PRMs specifically tailored for this long CoT paradigm remain underexplored.

Application of Reward Models in Mathematical Reasoning Mathematical reasoning in LLMs has seen significant progress with the introduction of reward models. Two types of reward models are commonly used: Outcome Reward Model (ORM) and Process Reward Model (PRM). ORMs assess entire solutions by assigning scores to final answers, while PRMs assign scores to individual steps, offering granular feedback. Research (Lightman et al., 2023; Wang et al., 2024a) demonstrates that PRMs generally outperform ORMs, underscoring their greater potential to enhance reasoning accuracy through guided search (Park et al., 2024; Zhang et al.; Snell et al., 2024) and reinforcement learning (Gao et al., 2024; Setlur et al., 2024). However, the effectiveness of PRMs depends heavily on the availability of high-quality training data, which traditionally requires costly human annotation (Lightman et al., 2023; Uesato et al., 2022). To address this challenge, recent work (Wang et al., 2024a; Luo et al., 2024; Wang et al., 2024c; Chen et al., 2024; Zhang et al., 2025) explore automated data collection methods, improving efficiency with MC estimation-based techniques. Despite these

advances, the substantial computational demands of MC-based methods remain a significant barrier when dealing with long reasoning chains, hindering the accumulation of sufficient training data.

3 Disadvantages of Current PRMs

Previous works on training PRMs primarily focus on the first incorrect step and assume that all subsequent steps are reasoned based on this erroneous step, thus deeming them all incorrect. However, in a long CoT pattern, self-correction behaviors often occur, where correct reflective steps can still follow an incorrect step. This results in a distribution shift between the PRM training data and the data encountered during inference in long CoT scenarios, potentially affecting performance. Therefore, we aim to investigate: *Does the self-correction behavior of long CoT models impact the performance of current PRMs?*

To answer this question, we construct two test sets: an Error-Free Set (EF Set) and a Reflection-Based Set (RB Set). The final answers of solutions for both sets are correct. The key difference lies in the correctness of the intermediate reasoning steps. All reasoning steps in the Error-Free Set are correct, while solutions in the Reflection-Based Set contain erroneous intermediate steps and perform self-correction. We evaluate several open-source PRMs on these two test sets: Qwen2.5-Math-PRM-7B (Zhang et al., 2025), MathShepherd-PRM-7B (Wang et al., 2024a), and Skywork-PRM-7B (o1 Team, 2024), focusing on the accuracy of PRMs in determining whether the solutions are correct.

Our experimental results, as presented in Table 1, reveal significant performance disparities between the two test sets across all evaluated PRMs. The most effective model, Qwen2.5-PRM-7B, demonstrates a 10% performance gap between the two test sets, while other models show even larger disparities, even exceeding 50%. These findings suggest that PRMs exhibit reduced effectiveness in evaluating reasoning processes that incorporate intermediate reflection.

Model	EF Set	RB Set
Qwen2.5-MATH-PRM-7B	0.98	0.88
MathShepherd-PRM-7B	0.79	0.27
Skywork-PRM-7B	0.93	0.59

Table 1: The prediction accuracy of open-source PRMs.

4 Method

In this section, we introduce a framework for constructing PRM data tailored to reflective reasoning. First, we introduce how to train a long CoT model that can autonomously segment its output based on semantics. Then, we describe our detailed step-by-step annotation guidelines and the implementation of PRM training. The overall framework of our method is shown in Figure 1.

4.1 Dividing Reasoning Process into Steps

Currently, no open-source LLMs satisfy both requirements: (1) generating long CoT reasoning processes, and (2) including separators ensuring semantic integrity at each step. While using double line breaks as delimiters to segment reasoning chains is a common practice, it compromises semantic coherence. Moreover, in the long CoT paradigm, this approach results in numerous segmentation steps, increasing the annotation workload.

To address this challenge, we develop a specialized generator based on Supervised Fine-Tuning (SFT). For constructing the SFT dataset, inspired by (Zheng et al., 2024), we adopt a two-step approach to segment open-source long CoT data. We first replace all line breaks with spaces and then utilize LLMs to resegment the reasoning process. Our segmentation balances cognitive cohesion (merging conceptually related steps) and modular independence (ensuring each step represents a distinct reasoning unit), while controlling step count and token length. A Detailed solution segmentation case is shown in Appendix A.

4.2 Annotation Standards for Reflective Reasoning

In the long CoT reasoning process, it is common for LLMs to revisit previous steps for self-correction and reflection, which is important for improving reasoning ability. However, previous annotation methods typically focus only on identifying the first erroneous step and assessing the correctness of preceding steps. These approaches overlook effective reflections after the first error in reflective reasoning, which hampers comprehensive process supervision.

We classify steps following an error into two types: (1) Reasoning on faulty assumptions, which compounds mistakes and leads further from the solution; (2) Corrective steps, which identify and

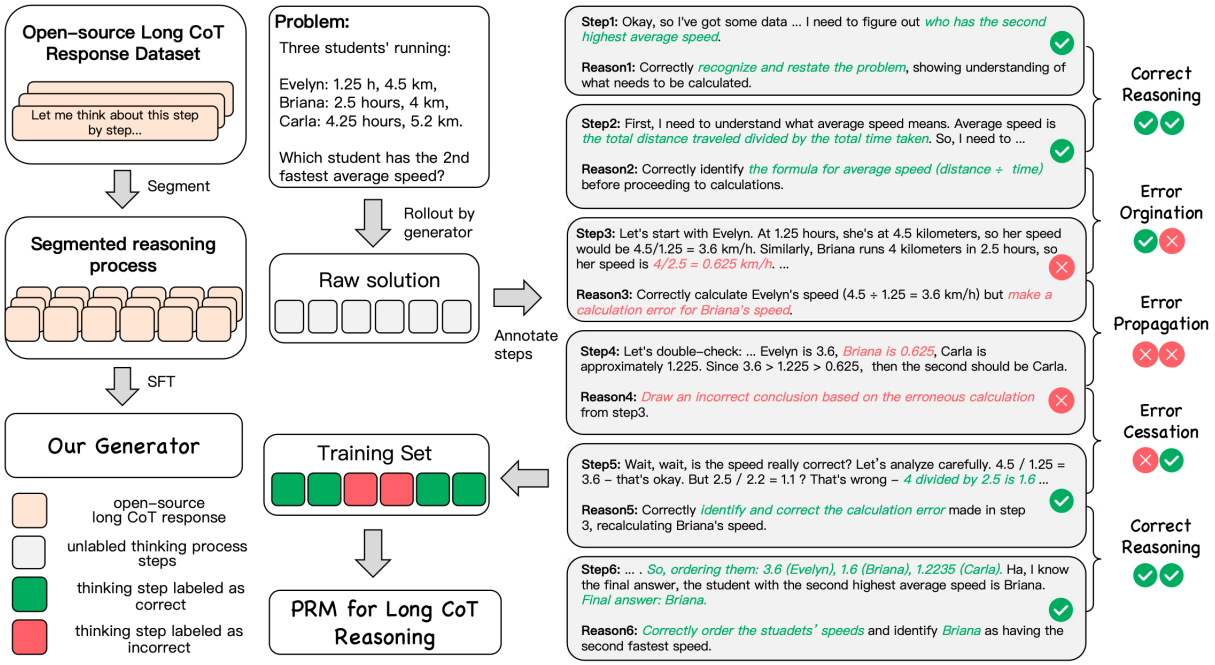

Figure 1: The overall framework of our method.

address errors, guiding the reasoning back on track. Based on our categorization, we introduce two new annotation rules designed to capture these post-error reasoning dynamics:

- **Error Propagation**: If the previous steps are incorrect and the current step neither introduces a new approach nor corrects the previous mistakes, but instead builds upon the erroneous steps, the current step is also considered incorrect.
- **Error Cessation**: If the previous steps are incorrect but the current step introduces a new, error-free approach or corrects the previous mistakes, the current step is considered correct.

Based on the annotation rules above, we assign appropriate labels to each step in the reasoning process. In addition, while our primary focus is on mathematical reasoning problems, these rules can also be applied to other domains such as coding, the 24-point game, and more.

4.3 LLM Judgement

Reflective models like the o1 (OpenAI, 2024) series have demonstrated outstanding performance in complex reasoning tasks. The results of the manual inspection in Appendix E indicate that reasoning LLMs are capable of effectively solving annotation tasks. Therefore, we incorporate the rules in Section 4.2 into the prompt (see Appendix B), allowing the reflective LLM to evaluate the correctness of each step.

4.4 Process Reward Model

Since the label for each step is a binary score, we use standard classification loss to train our PRM:

$$L_{PRM} = \sum_{i=0}^{K} \hat{y}_i \log y_i + (1 - \hat{y}_i) \log(1 - y_i) \quad (1)$$

where y_i represents the golden label of the i-th step s_i, $\hat{y}_i = PRM(\text{prompt}, s_{<=i})$ is the predicted score for s_i by PRM, and K is the total number of steps of the solution.

5 Experiments

In this section, we present the experimental results of our PRM in comparison with (1) open-source PRMs and PRMs trained on open-source datasets (Section5.2), and (2) PRMs trained on data generated through MC-based methods (Section5.3).

5.1 Experiment Settings

5.1.1 Dataset

Training: For training the PRMs and the generator, we construct the prompt set by combining the MATH training set with AIME problems from 1983 to 2023.

Model	MATH500		AIME2024		Step-Level Testset		
	PRM@64	PRM@8-step	PRM@64	PRM@8-step	Precision	Recall	F1
PRM-PRM800K	0.712	0.682	0.133	0.067	0.640	0.963	0.758
PRM-MS	0.758	0.706	0.233	0.100	0.613	**0.994**	0.758
Qwen2.5-PRM-7B	0.776	0.738	0.167	0.133	0.634	0.972	0.768
MathShepherd-7B	0.778	0.702	**0.267**	0.100	0.863	0.376	0.523
Skywork-PRM-7B	0.754	0.740	0.133	0.100	**0.936**	0.351	0.512
Ours	**0.816**	**0.750**	**0.267**	**0.167**	0.850	0.806	**0.828**

Table 2: The solution-level and step-level performance of Qwen2.5-7B-SFT* using our PRM and other baselines.

Evaluation: Our test set consists of MATH500 (Hendrycks et al., 2021) combined with AIME2024 (MAA, 2024). Additionally, we employ the generator to produce 800 solutions for prompts in the test set, forming our step-level test set, which is used to evaluate the accuracy of PRM in assessing each step of the reasoning process. The labels for this set are determined through a combination of cross-validation with the o1 model and manual annotation. For more information, please refer to Appendix E.

5.1.2 Generator

We segment about 2,000 samples in open-source datasets (Min et al., 2024) using method described in Section 4.1, and fine-tuned the Qwen2.5-7B-base (Team, 2024b) to obtain Qwen2.5-7B-SFT*.

5.1.3 Metrics

Our evaluation of PRMs includes solution-level and step-level analyses. At the solution level, we assess PRMs' ability to identify high-quality solutions using PRM@N and PRM@N-step. At the step level, we examine PRMs' accuracy to identify the correctness of individual steps with F1-related metrics.

Best of N. Consistent with previous work (Lightman et al., 2023; Wang et al., 2024a,c; Luo et al., 2024), we use Best-of-N for evaluation, which selects the highest-scoring response from N candidate responses. We define this evaluation metric as PRM@N. The score of each response is determined by the score at the final step.

Online Search. During each step, N candidates are sampled, and the step with the top score is selected to proceed with the generation. We define this metric as PRM@N-step.

Classification Metrics. To evaluate the classification performance at the step level, we use precision, recall, and F1 score as our metrics.

5.1.4 Baselines

Our baseline models are categorized as follows:

- **Fine-Tuned PRMs**: PRMs fine-tuned on open-source datasets, including: (1) PRM-PRM800K: Fine-tuned with the PRM800K dataset from OpenAI (Lightman et al., 2023). (2) PRM-MS: Fine-tuned with the Math Shepherd dataset from DeepSeek (Wang et al., 2024a).

- **Open-Source PRMs**: Existing open-source models, including: (1) Qwen2.5-MATH-PRM-7B (abbreviated as Qwen2.5-PRM-7B) (Zhang et al., 2025). (2) DeepSeek-MathShepherd-7B (abbreviated as MathShepherd-7B) (Wang et al., 2024a). (3) Skywork-PRM-7B (o1 Team, 2024).

5.2 Main Results

Solution Level According to the results shown in Table 2, we find that: (1) Our PRM consistently outperforms other PRMs in PRM@64. Specifically, on MATH500, it achieves an accuracy of 81.6%, which is 3.8% higher than the second-best PRM, demonstrating enhanced capability in identifying high-quality solutions. (2) Our PRM performs the best in PRM@8-step. This indicates that our PRM is capable of providing better intermediate signals, which guide the generator to produce higher-quality solutions.

Step Level The step-level experimental results in Table 2 indicate that: (1) Our PRM performs the best in step-level evaluation, achieving the highest F1 score. (2) Both open-source PRMs and PRMs trained on current open-source datasets exhibit an imbalance between precision and recall. PRM-PRM800K, PRM-MS, and Qwen2.5-PRM-7B tend to classify incorrect steps as correct, while MathShepherd-7B and Skywork-PRM-7B are more

prone to classifying correct steps as incorrect. In contrast, our PRM demonstrates the most balanced performance.

5.3 Comparison with MC-based Methods

Although MC-based method has been widely used, it contains significant noise. Their effectiveness relies on the ability of completion models. Correct steps may be misjudged as incorrect when completion models fail to produce correct solutions within limited rollouts. Reflective reasoning paradigm exacerbates the annotation noise. Incorrect steps may be misjudged as correct due to subsequent reflective and error-correction behaviors (Lanham et al., 2023).

5.3.1 Experiment Setup

For each prompt in the training set, we select 8 solutions generated by Qwen2.5-7B-SFT* for MC-based annotation, with the total number of steps approaching 1 million. We employ Qwen2.5-7B-SFT* to perform 8 completions for each step to assess their correctness. Following previous work (Wang et al., 2024a), we create hard labels based on the sampled completions and train PRM-MC-1M. The annotation process consumes approximately 23,040 A100 GPU hours. To ensure a fair comparison with PRM-MC-1M, we select the subset of our training data corresponding to solutions used in PRM-MC-1M training to train PRM-ours-1M. The PRM trained on the entire training set is referred to as PRM-ours-1.7M.

5.3.2 Results

Figure 2 presents the experimental results, which reveal that: (1) PRM-ours-1M achieves superior PRM@N scores compared to PRM-MC-1M consistently, demonstrating the improved quality of process supervision signals generated by our method. (2) PRM@N for PRM-ours-1.7M consistently outperforms PRM-ours-1M, indicating robust scalability of our approach. The PRMs trained using our method can effectively leverage more process supervision signals in larger training datasets.

Additionally, leveraging LLM for data annotation significantly improves efficiency compared to the MC-based method, enabling faster data annotation and more rapid training iterations.

Metric	PRM-MC-1M			PRM-ours-1M		
	Qwen	LLaMA	Gap	Qwen	LLaMA	Gap
PRM@8	0.752	0.386	0.366	0.786	0.466	0.320
PRM@16	0.758	0.370	0.388	0.776	0.464	0.312
PRM@32	0.762	0.400	0.362	0.786	0.478	0.308
PRM@64	0.760	0.436	0.324	0.786	0.518	0.268

Table 3: PRM@N of different models using PRM-MC and PRM-ours, along with the performance gaps.

6 Analysis

6.1 Robustness Compared to MC-based Method

This section aims to demonstrate that our method can provide more robust process supervision signals compared to commonly used MC-based methods. First, we show that there is a strong correlation between the signals obtained via the MC-based method and the completion model, and the correlation strengthens with increasing reasoning chain length. Second, we will compare the performance gap when the same PRM provides process reward signals to different models.

Correlation between Signals and Completion Model We first fine-tune LLaMA3.1-8B-Base (Dubey et al., 2024) using the SFT data described in Section 5.1.2 to get LLaMA3.1-8B-SFT*, both Qwen2.5-7B-SFT* and LLaMA3.1-8B-SFT* are employed as completion models for our analysis. Subsequently, we randomly sample 1,000 solutions generated by Qwen-2.5-7B-SFT* and use both completion models to annotate step-level hard labels following Math-Shepherd. Our analysis reveals that only 79% of the solution steps receive identical annotations from both models. Moreover, we observe a negative correlation between solution length and inter-model consistency. As illustrated in Figure 3, the consistency rate between the two completion models decreases as the number of solution steps increases. This finding suggests that MC-based annotation methods become progressively less reliable as reasoning chains extend.

Performance Gap when Providing Signals for Different Models We evaluate PRM-MC-1M and PRM-ours-1M by using them to provide reward signals during the inference processes of LLaMA3.1-8B-SFT* and Qwen2.5-7B-SFT* on MATH500. The results in Table 3 show that PRM-MC-1M introduces a significantly larger performance gap between the two models compared to PRM-ours-1M. This suggests that our method pro-

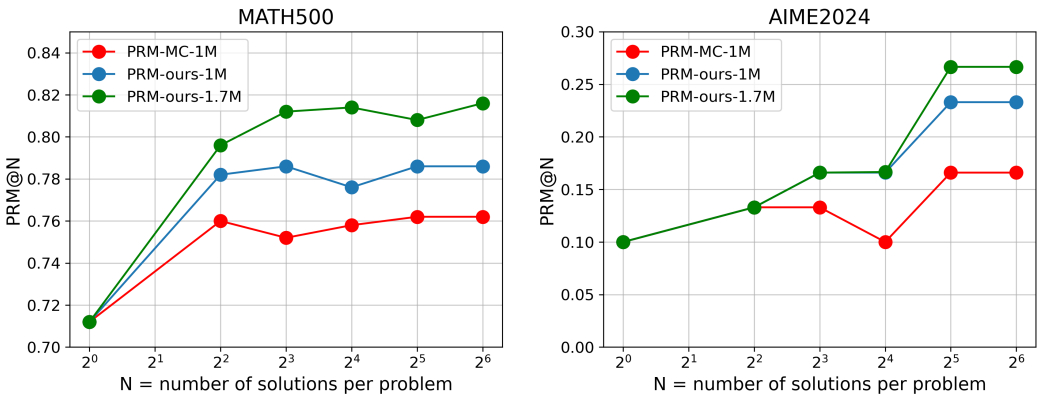

Figure 2: PRM@N of Qwen2.5-7B-SFT* using PRMs trained on data annotated by MC-based and our method.

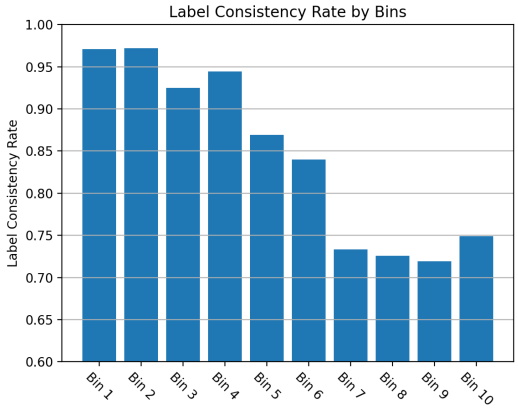

Figure 3: We categorize 1,000 solutions into 10 equal-sized bins based on their step counts, with Bin 1 containing solutions with the fewest steps and Bin 10 containing those with the most steps. Within each bin, we calculate the proportion of steps where both completion models assign identical hard labels.

Datasets	Metrics	FE	FES	Ours
MATH500	PRM@8	0.764	0.770	**0.786**
	PRM@16	0.774	**0.788**	0.776
	PRM@32	0.776	**0.792**	0.786
	PRM@64	0.780	**0.792**	**0.792**
AIME2024	PRM@8	0.133	0.133	**0.167**
	PRM@16	0.133	0.133	**0.167**
	PRM@32	0.167	0.167	**0.233**
	PRM@64	0.200	0.167	**0.233**

Table 4: PRM@N of Qwen2.5-7B-SFT* using PRMs trained under different experimental settings.

duces a more robust PRM, capable of providing consistent reward signals across different language models. Moreover, PRM-ours-1M demonstrates superior PRM@N compared to PRM-MC-1M across both Qwen2.5-7B-SFT* and LLaMA3.1-7B-SFT*. This consistent performance advantage across different models validates the adaptability of our PRM on different generators.

6.2 Data Distribution

This section examines the distributional differences between our training data and existing open-source datasets.

As illustrated in Figure 4, our dataset exhibits a significantly higher number of steps per solution and greater token consumption per step compared to open-source datasets. This distributional divergence primarily stems from our annotated data, which comprises long reasoning chains that incorporate reflective processes. These chains inherently demand more elaborate step-by-step reasoning and extended cognitive operations.

We believe that this distributional discrepancy causes PRMs trained on open-source data to generate less effective process reward signals for reflective reasoning chains. This limitation appears to be one of the key factors contributing to their inferior performance compared to our PRM at both the solution level and step level.

6.3 Enhanced Data Utilization Beyond First-Error Steps

This section aims to demonstrate the advantages of our method over the traditional paradigm, which typically only considers the first error step. Our approach extracts more training signals from each solution while maintaining the same sample size. We design a comparative study with three experimental settings: (1) Baseline: Our method; (2) First-error truncation (FE): Using identical training samples as (1) but only considering steps until the first error occurs; and (3) First-error truncation with supple-

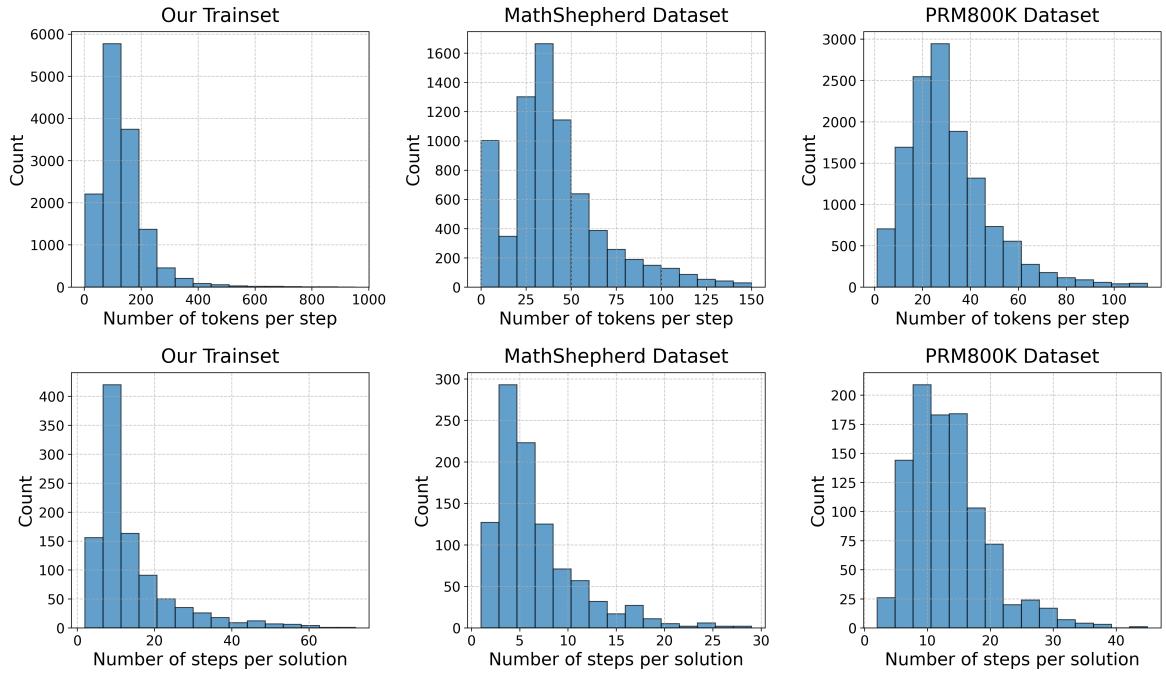

Figure 4: Distribution of the number of steps in each solution and the number of tokens contained in each step across different datasets. We randomly select 1,000 samples from each dataset for statistical analysis.

mentary data (FES): Extending (2) with additional training samples to align the total training signals in (1).

The experimental results in Table 4 reveal two key findings. First, given an equal number of solution samples, our method achieves superior performance on both MATH500 and AIME2024 by utilizing additional process supervision signals from each solution. Second, when aligning the number of signals, our method achieves comparable results to the traditional paradigm while using fewer solutions. Notably, on the more challenging AIME2024, our method even demonstrates a slight advantage. The enhanced effectiveness on AIME2024 can be attributed to its inherently challenging nature, which demands more sophisticated reflective reasoning processes.

6.4 Out-of-Distribution Performance

To demonstrate the generalizability of our method, we conduct evaluations on both elementary and advanced mathematical problems. Our evaluation select 500 questions from GSM8K (Cobbe et al., 2021) representing simple math problems, and 675 questions from Olympiad Bench (OBen) (He et al., 2024), which contains challenging, competition-level problems.

As can be seen from Table 5, most PRMs demonstrate comparable performance on GSM8K. While our PRM performs marginally below PRMs trained on Math-Shepherd and Skywork-PRM-7B, this slight difference might be attributed to the inclusion of GSM8K examples in their training set. Notably, when evaluated on the more challenging OBen dataset, our method exhibits substantially superior performance compared to all baseline approaches, highlighting its strength in handling sophisticated math problems. More information about the performance advantages of our PRM on complex datasets can be found in Appendix H.

Model	GSM8K	OBen
PRM-PRM800K	0.914	0.394
PRM-MS	**0.954**	0.441
Qwen2.5-PRM-7B	0.950	0.458
MathShepherd-7B	**0.954**	0.455
Skywork-PRM-7B	**0.954**	0.446
Ours	0.950	**0.510**

Table 5: PRM@64 on GSM8K and OBen of Qwen2.5-7B-SFT* using different PRMs.

7 Conclusion

In this paper, we introduce a PRM data annotation technique designed for reflective reasoning processes. We propose concepts of **Error Propagation** and **Error Cessation**, which enable precise

identification of steps based on flawed premises and highlight moments of meaningful reflection. Additionally, leveraging LLMs for annotation helps reduce the resource burden caused by extended reasoning chains. Experimental results demonstrate our PRM outperforms current open-source PRMs and PRMs trained on open-source datasets at the solution and step levels. Compared to commonly used MC-based methods, our method also exhibits comprehensive superiority.

8 Limitaions

Despite achieving superior performance in solution-level and step-level metrics compared to other baselines, there are several limitations to our approach. (1) Scalability: While our PRM provides more accurate signals in our diverse evaluation test, due to constraints in training data and computational resources, we are unable to validate the advantages of our method in broader experimental settings. Future work could explore how the accuracy and generalization of PRM scale as the number of prompts and generated solutions increases. (2) Dependency on LLM Capabilities: Our labeling method requires the LLM judges to have strong reasoning capabilities. Consequently, the accuracy of labeling is limited by the inherent abilities of the model itself. However, with ongoing improvements in open-source reflective models (Team, 2024a; Guo et al., 2025; Muennighoff et al., 2025), this issue could be mitigated. Future work could also investigate the performance of small reflection models used for annotation.

9 Acknowledgments

This work was supported in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA0480301 and the Key Research Project of Chinese Academy of Sciences (No. RCJJ-145-24-15).

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*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*.

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*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. 2024. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

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*.

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. 2024. Augmenting math word problems via iterative question composing. *arXiv preprint arXiv:2401.09003*.

Zichen Liu, Changyu Chen, Wenjun Li, Tianyu Pang, Chao Du, and Min Lin. 2025. There may not be aha moment in r1-zero-like training — a pilot study. https://oatllm.notion.site/oat-zero. Notion Blog.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

MAA. 2024. American invitational mathematics examination - aime.

Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, et al. 2024. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Skywork o1 Team. 2024. Skywork-o1 open series. https://huggingface.co/Skywork.

OpenAI. 2024. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/. Accessed: Month Day, Year.

Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. 2024. Ensembling large language models with process reward-guided tree search for better complex reasoning. *arXiv preprint arXiv:2412.15797*.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Kimi Team. 2025. Kimi k1.5: Scaling reinforcement learning with llms.

Qwen Team. 2024a. Qwen team. qwq: Reflect deeply on the boundaries of the unknown, 2024b.

Qwen Team. 2024b. Qwen2.5: A party of foundation models.

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*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024a. 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.

Yifei Wang, Yuheng Chen, Wanting Wen, Yu Sheng, Linjing Li, and Daniel Dajun Zeng. 2024b. Unveiling factual recall behaviors of large language models through knowledge neurons. *arXiv preprint arXiv:2408.03247*.

Yifei Wang, Feng Xiong, Yong Wang, Linjing Li, Xiangxiang Chu, and Daniel Dajun Zeng. 2025. Position bias mitigates position bias: Mitigate position bias through inter-position knowledge distillation. *arXiv preprint arXiv:2508.15709*.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. 2024c. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*.

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.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search, 2024a. *URL https://arxiv. org/abs/2406.03816*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

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

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

A Solution Reformatting

We use the method described in Section Section4.1 to segment the open-source sample. An example of the solution reformation is shown in Figure 5. After segmentation, sentences at the same step are shown in the same color.

B Prompt Template for LLM Annotation

The prompt template for LLM judger is shown in Figure 7.

C Training Details

C.1 SFT

The detailed training parameters for generator are provided in Table 6.

Hyperparameter	Value
learning rate	1e-5
epochs	3
batch size	24
max length	16384

Table 6: Hyperparameters of SFT

C.2 PRM Training

The detailed training parameters in PRM training are provided in Table 7.

Hyperparameter	Value
learning rate	1e-6
epochs	1
batch size	256
max length	10240

Table 7: Hyperparameters of PRM training

D Case Study: An Annotation Result of LLM judger

To better illustrate the PRM training data, an example case is presented in Figure 8. The model's reasoning process consists of 11 steps. The "content" field represents the model's reasoning at each step, "Score" indicates the evaluation by o1 for that step (1 for correct, 0 for incorrect), and "Reason" provides the rationale behind o1's scoring.

E Human Annotation Details

We employ individuals with bachelor's and graduate degrees to manually assess the annotation accuracy of different LLMs on the step-level test set. First, we provide training for the annotators to ensure they have a comprehensive understanding of the prompts used for LLM-based annotation in Appendix B. To facilitate a thorough comprehension of errors in the reasoning process, we classify mathematical reasoning errors into two categories: operation errors and conceptual errors. Operation errors encompass mistakes in formula calculations, derivations, and similar computational inaccuracies. In contrast, conceptual errors involve incorrect reasoning directions, such as misinterpreting the problem or improperly applying mathematical formulations.

For each entry in the step-level test set, annotators receive a file containing five fields: the question, the ground-truth answer, each step of the solution, the rationale provided by the LLM judge for each step, and the judgment result for each step. A step is considered correctly scored if both the rationale and the scoring outcome are deemed reasonable. Conversely, if either the rationale or the scoring outcome is found to be unreasonable, the scoring is marked as incorrect. The annotation accuracy of different LLMs is shown in Table 8.

Model	Annotation Accuracy
gpt-4o-2024-08-06	0.668
claude-3.5-sonnet	0.726
o1	**0.963**

Table 8: The annotation accuracy of different models.

F Generalization of our PRM

To further assess the generalization capability of our PRM, we conducted experiments on the AIME24 dataset under challenging conditions. Specifically, for each problem, we employ the DeepSeek-R1-Distill-Qwen-1.5B model to generate 32 candidate solutions. From these, we select 10 solutions per problem, ensuring that no more than 2 of the 10 are correct. This selection strategy increases the task difficulty for the PRM, as it must identify the correct solutions from a pool dominated by incorrect ones.

We evaluated the performance of various PRMs using two rule-based segmentation methods :

- **Segmentation based on Reflection Words (SRW)**: This method segments the response content using reflection words and merges short segments to form more coherent units.

- **Segmentation based on Double Newlines (SDN)**: Double newline characters serve as delimiters to segment the response.

Experimental results are shown in Table 9. Notably, even when the generation model is not our SFT model and segmentation is performed using simple rule-based approaches, our PRM consistently achieves superior performance in terms of PRM@10 on the AIME24 dataset. These results demonstrate the robustness and strong generalization ability of our PRM across different segmentation strategies and generation models.

Model	SRW	SDN
Qwen2.5-PRM-7B	0.367	0.333
MathShepherd-7B	0.267	0.300
Skywork-PRM-7B	0.300	0.300
Ours	**0.533**	**0.500**

Table 9: The PRM@10 of different PRMs under different segmentation methods.

G Performance on Larger Models

To strengthen our conclusions, we select Qwen2.5-14B-Base as the base model and apply our method to train a new PRM. As there are currently no open-source PRMs with comparable parameter sizes for direct comparison, we establish baselines by training Qwen2.5-14B-Base on the open-source PRM800K and MathShepherd datasets. The experimental results of various models on the PRM@64 metric are presented in Table 10. Our dataset achieves the best performance.

Dataset	MATH	OBen	AIME24
PRM800K	0.810	0.502	0.233
MathShepherd	0.770	0.455	0.200
Ours	0.964	**0.536**	**0.333**

Table 10: The performance of different PRM trainsets on Qwen2.5-14B-Base.

H Robust Performance Scaling with Dataset Complexity

While our PRM demonstrates modest improvements on simpler datasets, its true potential emerges when applied to more challenging problems that demand extensive reasoning and iterative refinement.

To quantify reasoning complexity across different datasets, we analyzed three key indicators: (1) average tokens, (2) average reasoning steps, and (3) average reflection tokens (RT). Reflection tokens encompass specific terms indicating reflection processes, such as "rethink" and "reconsider", with the vocabulary set derived from (Liu et al., 2025). To measure our method's effectiveness, we calculated our PRM's relative advantage, defined as the difference between our PRM's PRM@64 and the mean PRM@64 of all baseline methods.

As illustrated in Table 11, there is a clear correlation between dataset complexity and our method's performance advantage. Specifically, as problems require longer reasoning chains and more frequent reflection steps, the performance gap between our PRM and baseline methods widens significantly. This scaling behavior provides strong evidence that our approach is particularly effective for complex reasoning tasks that demand extended CoT reasoning.

	GSM8K	MATH500	OBen	AIME24
Difficulty Level	Grade-school	High-school	Competition	Competition
Average Tokens	2339	5376	8150	12908
Average Steps	6.9	13.1	17.8	25.8
Average RT	2.5	5.7	8.3	12.0
Advantage	0.4%	6.0%	7.1%	8.0%

Table 11: Statistical metrics of reasoning processes across different datasets, and the advantage of our PRM compared to other PRMs on different datasets.

Problem: The chord AB subtends an arc of the circle equal to 120°). Point C lies on this arc, and point D lies on the chord AB. Additionally, AD = 2, BD = 1, and DC = $\sqrt{2}$. Find the area of triangle ABC.
Solution after reformation: Alright, let's see here. I've got this geometry problem to solve. So, there's a circle with a chord AB that subtends an arc of 120 degrees. There's a point C on this arc, and a point D on chord AB such that AD is 2, BD is 1, and DC is sqrt(2). I need to find the area of triangle ABC. First, I should probably draw a diagram to visualize this. So,

I've got a circle, chord AB with points A and B on the circle, and the arc from A to B that's 120 degrees. Point C is somewhere on that arc, and point D is on AB such that AD is 2 and BD is 1, making AB a total of 3 units. DC is sqrt(2), which is interesting.

Since AB subtends an arc of 120 degrees, that means the central angle corresponding to arc AB is 120 degrees. In a circle, the measure of an inscribed angle is half the measure of the central angle that subtends the same arc. So, angle ACB, which is inscribed and subtended by arc AB, should be 60 degrees.

Wait, no. Actually, angle ACB is not necessarily 60 degrees. Let me think again. The central angle is 120 degrees, so the inscribed angle subtended by the same arc would be half of that, which is 60 degrees. But angle ACB is an inscribed angle subtended by arc AB, so yes, it should be 60 degrees.

But hold on, point C is on the arc AB, so angle ACB is indeed 60 degrees. Now, I need to find the area of triangle ABC. To find the area, I can use the formula (1/2)*base*height, or (1/2)*ab*sin(C), where a and b are sides and C is the included angle.

But I don't know the lengths of sides AC and BC yet. Maybe I can find those using the Law of Sines or Cosines.

First, I need to find the lengths of AC and BC. Since I have a circle with chord AB of length 3, and central angle 120 degrees, I can find the radius of the circle.

The formula for the length of a chord is 2*r*sin(theta/2), where theta is the central angle in radians. Wait, theta is in degrees here, so I need to be careful with units.

Let me convert 120 degrees to radians. 120 degrees is (120/180)*pi = (2/3)pi radians.

So, chord length AB = 2*r*sin((120)/2) = 2*r*sin(60 degrees). Sin 60 degrees is sqrt(3)/2, so AB = 2*r*(sqrt(3)/2) = r*sqrt(3).

But AB is given as AD + DB = 2 + 1 = 3 units. So, r*sqrt(3) = 3, which means r = 3/sqrt(3) = sqrt(3). Okay, so the radius of the circle is sqrt(3).

Figure 5: An Example of solution reformation (part 1).

Now, I need to find AC and BC. Since C is on the arc AB that's 120 degrees, the angle at the center is 120 degrees, so the arc AC and arc CB must add up to 120 degrees. But I don't know their individual measures yet. Wait, point C is a specific point on the arc AB, but I

don't have more information about its position. Maybe I need to use point D and the given lengths to find more information.

Point D is on AB such that AD = 2 and DB = 1, and DC = sqrt(2). So, triangle DBC has sides DB = 1, DC = sqrt(2), and angle at D I don't know yet.

Coordinates of A: (sqrt(3), theta)

Coordinates of B: (-sqrt(3)/2, 3/2)

Coordinates of C: (sqrt(2),1)

Using shoelace formula again:

Area = (1/2)|x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)|

Plug in:

= (1/2)|sqrt(3)*(3/2 - 1) + (-sqrt(3)/2)*(1 - 0) + sqrt(2)*(0 - 3/2)|

= (1/2)|sqrt(3)*(1/2) - sqrt(3)/2 - 3*sqrt(2)/2|

= (1/2)| (sqrt(3)/2 - sqrt(3)/2) - 3*sqrt(2)/2 |

= (1/2)| 0 - 3*sqrt(2)/2 |

= (1/2)*(3*sqrt(2)/2)

= (3*sqrt(2))/4

Okay, so maybe it's correct. The area of triangle ABC is (3*sqrt(2))/4.

Figure 6: An Example of solution reformation (part 2).

You are a mathematical expert. The user will provide a math problem, a step-by-step solution process, and the GT answer. First, you need to extract the short final answer from the GT Answer. Then, carefully check the user's step-by-step solution process and assign a score of either 0 or 1 for each step.
You need to carefully examine the correctness of each step and provide a brief explanation. If there is an error in the current solution step, such as a calculation error, a derivation error, or a logical error, the score should be 0. If the current solution step is error-free, follow these rules to assign a score:

1. **Error Propagation:** If there is an error in the preceding steps and the current step doesn't provide a new problem-solving idea or perform a proper correction, the score should be 0. For example, if STEP K contains an error, and STEP K+1 continues analyzing based on STEP K's error, then STEP K+1 is also considered incorrect, and the score should be 0.

2. **Error Cessation:** If there is an error in the preceding steps, but the current step introduces a new, correct solution approach, the score should be 1. If the current step corrects the error from the previous steps, the score should also be 1. For example,

   - STEP K contains an error.
   - STEP K+1 continues the analysis based on the error.
   - STEP K+2 corrects the previous error or introduces a problem-solving idea.

   In this case, STEP K and STEP K+1 should be scored as 0, and STEP K+2 should be scored as 1.

3. Final Step: For the final step in the solution, if the answer obtained does not match the short answer, the score should be 0. If it matches the short answer, the score should be 1.

Your response format should be in json format:

```
[
    {
        "STEP 0": 1(int),
        "Reason": xxxx(str)
    },
    {
        "STEP 1": 1(int),
        "Reason": xxxx(str)
    }
    ...
]
```

Note: When analyzing the solution process, it is important to remain objective and rational, and not be misled by the description of the solution steps.

Figure 7: The prompt template for LLM judger.

Problem: A cross-country team completed a training run, and each student's performance was measured. Evelyn ran 4.5 distance units in 1.25 time units. Briana ran 4 distance units in 2.5 time units. Carla ran 5.2 distance units in 4.25 time units. Debra ran 2.8 distance units in 5.6 time units. Angela ran 1.4 distance units in 6.8 time units. Based on these results, which student had the second highest average speed?
Solution:

- Step 1
  - **Content:** Okay, so I've got this cross-country team's training run data. There are five students: Evelyn, Briana, Carla, Debra, and Angela. Each has recorded their distance and time. I need to figure out who has the second highest average speed.
  - **Score:** 1.
  - **Reason:** This step correctly introduces the problem, identifying the five students (Evelyn, Briana, Carla, Debra, and Angela) and the need to determine who has the second highest average speed.
- Step 2
  - **Content:** First, I need to understand what average speed means. Average speed is the total distance traveled divided by the total time taken. So, I need to look at each student's data and see how far they went and how long it took them. The data gives me the distance traveled and time taken for each student.
  - **Score:** 1.
  - **Reason:** This step correctly explains that average speed is calculated by dividing the total distance traveled by the total time taken.
- Step 3
  - **Content:** Let's start with Evelyn. She traveled 4.5 distance units in 1.25 time units. Briana traveled 4 distance units in 2.5 time units. Carla traveled 5.2 distance units in 4.25 time units. Debra traveled 2.8 distance units in 5.6 time units. Angela traveled 1.4 distance units in 6.8 time.
  - **Score:** 1.
  - **Reason:** Correctly lists all the data points given in the problem: Evelyn (4.5 distance units in 1.25 time units), Briana (4 distance units in 2.5 time units), Carla (5.2 distance units in 4.25 time units), Debra (2.8 distance units in 5.6 time units), and Angela (1.4 distance units in 6.8 time units).
- Step 4
  - **Content:** To find average speed, I need to calculate the total distance each ran and the total time it took, then divide distance by time. Based on the given data, it seems like each student maintained a constant speed since we only have their final distance and time measurements.
  - **Score:** 1.
  - **Reason:** This step correctly explains the approach to calculate average speed by dividing distance by time for each student.

Figure 8: An example of an annotation result from the LLM judger (part 1).

- Step 5
  - **Content:** So, for Evelyn, from time 0 to 1.25 units, she went from 0 to 4.5 units, so her speed would be 4.5 divided by 1.25. Let me calculate that: 4.5 / 1.25 = 3.6 units per time unit. Similarly, Briana went from 0 to 4 units in 2.5 time units, so her speed is 4 / 2.5 = 0.625 units per time unit.
  - **Score:** 0.
  - **Reason:** This step incorrectly calculates Briana's speed, 4/2.5=1.6.
- Step 6
  - **Content:** Carla went from 0 to 5.2 units in 4.25 time units, so her speed is 5.2 / 4.25 = approximately 1.225 units per time unit. Debra went from 0 to 2.8 units in 5.6 time units, so her speed is 2.8 / 5.6 = 0.5 units per time unit. Angela went from 0 to 1.4 units in 6.8 time units, so her speed is 1.4 / 6.8 $\approx$ 0.205 units per time unit. So, comparing these speeds, Evelyn has the highest average speed at 3.6 units per time unit, second is Carla at 1.225, then Briana at approximately 0.625, Debra at 0.5, and finally Angela at approximately 0.205.
  - **Score:** 0.
  - **Reason:** Based on the incorrect calculation results in Step 5, the ranking is performed. This step is also considered incorrect.
- Step 7
  - **Content:** Wait a minute, but I should make sure I'm interpreting the data correctly. Maybe these numbers represent different aspects of the run. If these numbers represent the entire run, then the distance would be the total distance traveled, and the time would be the total time taken. Let's double-check: Evelyn is 3.6, Briana is 0.625, Carla is approximately 1.225, Debra is 0.5, Angela is 0.205. Since 3.6 > 1.225 > 0.625 > 0.5 > 0.205, then the second should be Carla.
  - **Score:** 0.
  - **Reason:** Based on the incorrect calculation results from Step 5, the ranking is performed incorrectly.
- Step 8
  - **Content:** Wait, let's recheck to prevent mistakes. For Evelyn: 4.5 / 1.25, or 4.5 / (5/4) = 4.5 * (4/5) = (4.5 * 4)/5 = 18/5 = 3.6, yes. For Briana: 2.5 / 4 = 25/40 = 0.625, yes. For Carla: 5.2 / 4.25 $\approx$ 1.2235, yes. For Debra: 2.8 / 5.6 = 0.5, yes. Angela: 1.4 / 6.8 $\approx$ 0.205, yes. The calculations are correct, and the order seems correct.
  - **Score:** 0.
  - **Reason:** When calculating Briana's average speed, the numerator and denominator are reversed.

Figure 9: An example of an annotation result from the LLM judger (part 2).

- Step 9
  - **Content:** Wait, wait, is the calculation really correct? Let's analyze carefully. 4.5 / 1.25 = 3.6 — that's okay. But 4 / 2.5 = 0.625? That's wrong — 2.5 should not be greater than 4. 4 / 2.5 = 1.6 units per time unit, not 0.625 like I incorrectly calculated before! So the correct ordering of speeds is: 3.6 (Evelyn) > 1.6 (Briana) > 1.224 (Carla) > 0.5 (Debra) > 0.206 (Angela).
  - **Score:** 1.
  - **Reason:** Correctly recalculates and verifies all average speeds: Evelyn (3.6), Briana (1.6), Carla (1.225), Debra (0.5), and Angela (0.205). Properly rank them in descending order and correctly identify that Briana has the second highest average speed.
- Step 10
  - **Content:** To confirm, let's list the average speeds again: Evelyn: 4.5 / 1.25 = 3.6, Briana: 4 / 2.5 = 1.6, Carla: 5.2 / 4.25 $\approx$ 1.225, Debra: 2.8 / 5.6 = 0.5, Angela: 1.4 / 6.8 $\approx$ 0.205, So, ordering them: 3.6 (Evelyn), 1.6 (Briana), 1.225 (Carla), 0.5 (Debra), 0.2059 (Angela). Therefore, the second highest is Briana with approximately 1.6.
  - **Score:** 1.
  - **Reason:** Provides a final confirmation of the calculations and ranks, correctly stating that the second highest average speed belongs to Briana.
- Step 11
  - **Content:** Ha, I know the final answer, the student with the second highest average speed is Briana. Final answer: Briana.
  - **Score:** 1.
  - **Reason:** Correctly states the final answer as Briana.

Figure 10: An example of an annotation result from the LLM judger (part 3).