From Mathematical Reasoning to Code: Generalization of Process Reward Models in Test-Time Scaling

Zhengyu Chen[1]*, Yudong Wang [2]*, Teng Xiao[3], Ruochen Zhou [1], Xuesheng Yang [4], Wei Wang[1], Zhifang Sui [2], Jingang Wang[1]

[1]Meituan Inc.
[2] National Key Laboratory for Multimedia Information Processing, Peking University
[3] Allen Institute for Artificial Intelligence
[4] Peking University
{chenzhengyu04,wangjingang02}@meituan.com

Abstract

Recent advancements in improving the reasoning capabilities of Large Language Models have underscored the efficacy of Process Reward Models (PRMs) in addressing intermediate errors through structured feedback mechanisms. This study analyzes PRMs from multiple perspectives, including training methodologies, scalability, and generalization capabilities. We investigate the interplay between pre-training and reward model training FLOPs to assess their influence on PRM efficiency and accuracy in complex reasoning tasks. Our analysis reveals a pattern of diminishing returns in performance with increasing PRM scale, highlighting the importance of balancing model size and computational cost. Furthermore, the diversity of training datasets significantly impacts PRM performance, emphasizing the importance of diverse data to enhance both accuracy and efficiency. We further examine test-time scaling strategies, identifying Monte Carlo Tree Search as the most effective method when computational resources are abundant, while Best-of-N Sampling serves as a practical alternative under resource-limited conditions. Notably, our findings indicate that PRMs trained on mathematical datasets exhibit performance comparable to those tailored for code generation, suggesting robust cross-domain generalization. Employing a gradient-based metric, we observe that PRMs exhibit a preference for selecting responses with similar underlying patterns, further informing their optimization.

Introduction

In recent years, the advancement of Process Reward Models (PRMs) has garnered significant attention due to their potential to enhance Large Language Models' mathematical reasoning capabilities (Zhang et al. 2025; Lightman et al. 2023). These models aim to systematically identify and reduce intermediate errors in reasoning processes (Zhang et al. 2024b; Wang et al. 2024; Chen, Xiao, and Kuang 2022). This targeted approach has yielded promising results in mathematics, leading researchers to explore whether such capabilities extend to other fields, such as code generation. This study extends the exploration of PRMs to code generation tasks, examining their generalization ability and performance under various scaling conditions to optimize PRM training.

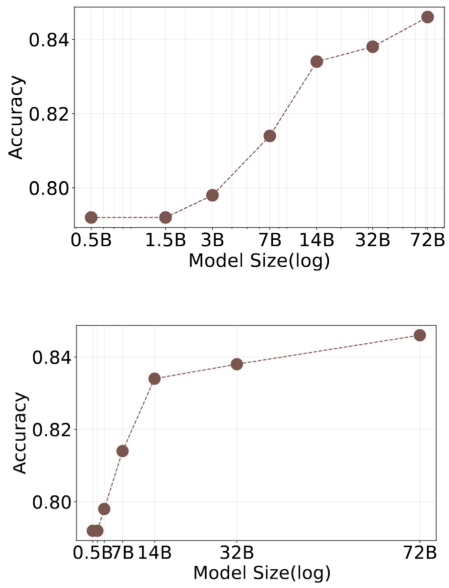

Figure 1: Comparison of the performance of PRMs (init from Qwen2.5) with varying model sizes during inference using the same language model. As model size increases, accuracy improves rapidly, indicating a better capability to capture the complexity necessary for solving mathematical problems.

Our research investigates the scalability of PRMs by analyzing the interplay between pre-training and reward model training FLOPs. We focus on how these computational resources impact the efficiency and accuracy of PRMs in executing complex reasoning tasks. By evaluating PRMs primarily trained on mathematical data, we aim to assess their robustness in handling novel tasks within the domain of code generation.

Scaling, data diversity, and search strategy are systematically analyzed as factors influencing a single question: how to train and deploy PRMs efficiently under compute con-

*Equal Contribution.

straints. Key findings from our study reveal several insights into these factors. Notably, we observe diminishing returns in model performance as PRM size increases, emphasizing the need for a balanced approach to model size and computational cost. Moreover, the diversity of training datasets significantly influences PRM performance, highlighting the importance of incorporating varied data to enhance accuracy and efficiency. In addition to training considerations, our research explores test-time scaling strategies (Setlur et al. 2024; Chen et al. 2024a) to optimize PRM deployment. We evaluate various search strategies, including Best-of-N Sampling, Beam Search, Monte Carlo Tree Search (MCTS), and Majority Voting, to enhance reasoning accuracy in test scenarios. Our results indicate that MCTS is the most effective strategy when ample computational resources are available, while Best-of-N Sampling offers a practical solution under resource constraints due to its simplicity and speed.

Our investigation explores the generalization capabilities of PRMs across domains, particularly from mathematical reasoning to code generation. The results demonstrate that PRMs trained on mathematical corpora perform comparably to those optimized on code-centric datasets, suggesting promising cross-domain adaptability. Here are the contributions of this paper:

1. *Analysis of Training Compute on PRM Performance*: We analyze the effects of pretraining and reward training compute on PRM efficacy. By experimenting with models of varying sizes and distinct compute allocations, we identify the balance between model size and compute resources, enhancing generalization across tasks.

2. *Test-Time Scaling Performance With Strategies*: We assess PRM performance during test-time scaling, focusing on the optimization of performance with search strategies. This involves the importance of selecting search strategies based on computational resources and time constraints.

3. *PRM Generalization Across Domains*: We explore PRM generalization in mathematics and coding tasks. With various search methods and diverse training sets, we assess cross-domain performance, revealing the generalization from math to code. Through quantitative analysis of the patterns inherent in the model's output, our findings indicate that the PRM exhibits a preference for selecting content exhibiting similar underlying reasoning patterns.

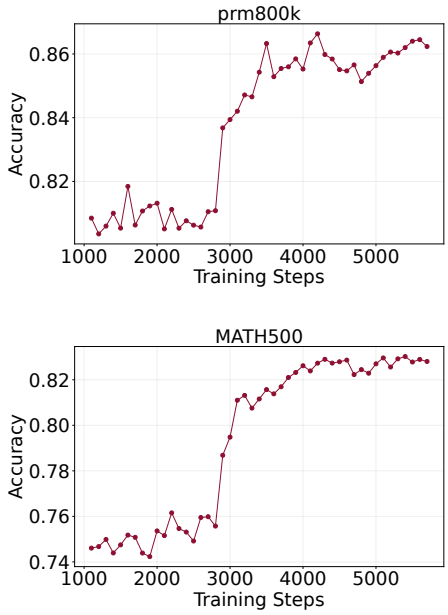

Figure 2: Impact of PRM Training Steps on Accuracy. The x-axis represents the number of training steps, while the y-axis indicates the accuracy.

Process Reward Modeling

Process Reward Models are designed to improve the performance of LLMs by providing granular feedback on intermediate steps during the problem-solving process. Unlike traditional models that only evaluate the final output, PRMs assess each step in a reasoning sequence, identifying and addressing errors as they occur. This step-wise evaluation allows for more accurate and reliable solutions, particularly in complex tasks such as mathematical reasoning and code generation.

Automatic Step-Level Annotation and Filtering

In the training process of Process Reward Models, automatic step-level annotation and filtering are crucial steps to enhance model performance. This process involves the following aspects:

1. Collection of Reasoning Tasks: To construct a high-quality training dataset, we collect a diverse set of reasoning tasks. These tasks cover a wide range of subjects and difficulty levels, ensuring that the PRM can generalize across different types of reasoning processes. Each task is broken down into a sequence of intermediate steps, with each step representing an action or decision that a model might take when generating a solution.

2. Annotation of Steps with LLM: Each step in the reasoning process is annotated based on its correctness. To facilitate the annotation process, we utilize pre-trained Large Language Models to generate multiple candidate solutions for each problem. By simulating rollouts from each solution, we identify the intermediate steps where errors occur. Heuristic methods, such as Monte Carlo estimation and binary search, are employed to label each step as correct or incorrect. This automated pipeline generates extensive and diverse supervision data, reducing the dependency on costly human annotations and facilitating scalable training of PRMs.

3. Data Filtering: To improve the quality of the training data, we introduced an ensemble-based filtering mechanism. This mechanism cross-verifies the correctness of reasoning steps using different LLMs. Only those steps where all LLMs agree on the error location are retained for training. This approach mitigates the noise and inaccuracies inherent in using a single model, thereby enhancing the reliability of the training data.

Through these steps—including cross-task data fusion, dual-model consistency verification, and multi-model fil-

tering—our Automatic Step-Level Annotation and Filtering (ASLAF) achieves a more significant improvement in dataset effectiveness compared to previous data resources (Wang et al. 2024; Lightman et al. 2023). This not only substantially enhances the efficiency and accuracy of PRM training but also lays a solid foundation for the model's performance in complex reasoning tasks. We validate the effectiveness of Automatic Step-Level Annotation and Filtering (ASLAF) in Experiments.

Objective and Framework

The primary objective of training the PRM is to transform the evaluation of reasoning processes into a supervised learning task. The PRM is designed to predict the correctness of each step in a sequence of reasoning, thereby providing continuous feedback to guide the LLM towards generating accurate solutions. For generation, Qwen2.5-Coder-7B, 32B-Instruct and QwQ-32B-Preview is utilized. The PRMs are initialized from the Qwen2.5 series (0.5B–72B) models, with the original language modeling head replaced by a scalar-value head designed for evaluating reasoning steps.

In this framework, given a problem Q and a sequence of reasoning steps $x_1, x_2, \ldots, x_T$, the PRM outputs a probability $p_t = \text{PRM}(Q, x_1, x_2, \ldots, x_t)$, which represents the likelihood that the step x_t is correct. The goal of the training process is to minimize the discrepancy between these predicted probabilities and the true binary labels y_t assigned to each step, where y_t indicates whether x_t is correct or not.

The PRM is trained using supervised learning, with the training data comprising problems paired with annotated reasoning steps, each labeled as correct or incorrect. This transforms the problem into a binary classification task. Specifically, the task is to predict the binary label y_t for each step:

$$y_t = \text{PRM}(Q, x_1, x_2, \ldots, x_t) \tag{1}$$

To measure the prediction error for each step, the binary cross-entropy loss is utilized, defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{t=1}^{N}[y_t \log(p_t) + (1 - y_t)\log(1 - p_t)] \tag{2}$$

where y_t is the true label for the t-th step, and p_t is the predicted probability that the step is correct. This loss function helps optimize the PRM to align its predictions with the true correctness of steps.

Once trained, PRMs are integrated with LLMs to provide real-time feedback during the reasoning process. At each step, the LLM generates potential actions (next reasoning steps), which are evaluated by the PRM. The feedback is used to adjust the policy, guiding the model towards more accurate reasoning pathways.

Experimental Setup

In this section, we outline the experimental setup employed to evaluate the performance of PRMs under different data construction methodologies. Our experiments aim to assess the comparative effectiveness of different strategies in enhancing the detection of erroneous reasoning steps and improving downstream task performance.

Datasets

Post Training We construct the three mathematical training sets and code training sets for PRMs. The mathematical training corpus is derived from PRM 800K (Lightman et al. 2023) and Math-shepherd (Wang et al. 2024), which include reasoning processes and evaluation labels. Moreover, we use ASLAF to combine and filter PRM 800K and Math-shepherd datasets. The code corpus is obtained from TACO (Li et al. 2023), APPS (Hendrycks et al. 2021), and CodeContent (Li et al. 2022). We use pre-trained LLM to generate reasoning processes and resulting code, and subsequently employ LLM to annotate them.

The ASLAF dataset contains approximately 1.2 million samples, while PRM800k and Math-Shepherd contain 800k and 445k samples respectively. To ensure fair comparison in our experiments, we randomly sampled 400k examples from each dataset for training. This controlled sampling approach allows us to isolate the effects of data quality and diversity from those of data quantity.

Evaluation To comprehensively evaluate the model's capability in mathematical reasoning problems, we utilize the MATH-500, and PRM 800K (Lightman et al. 2023). In terms of code generation, we employ HUMANEVAL+, MBPP+, LiveCodeBench, and BigCodeBench. EvalPlus (Liu et al. 2023) introduces HUMANEVAL+, which adds 80 specially designed problems to the original HUMANMEVAL (Chen et al. 2021), along with corrections to the answers in the original dataset. Similarly, MBPP+ provides an additional 35 test cases for MBPP (Austin et al. 2021).

Evaluation Methodology

To evaluate the efficacy of our trained PRMs, we focus on two primary aspects: 1) Downstream Task Performance: The models' ability to enhance overall problem-solving effectiveness. 2) Erroneous Step Identification: Precision in detecting specific reasoning errors.

Consistent with prior work (Lightman et al. 2023; Wang et al. 2024; Luo et al. 2024; Cobbe et al. 2021; Yang et al. 2024), we employ the BON sampling strategy; the highest-scored response from N candidates is selected according to the PRM, with evaluations specifically using "prm@N".

Additionally, the results of majority voting among the eight samples (maj@N) serve as a baseline, while pass@N (the proportion of test instances where any of the eight samples achieved a correct final answer) is referenced as an upper bound.

Through this setup, we aim to glean insights into the comparative strengths and limitations of different annotation methods and their impact on the generalization capabilities of PRMs.

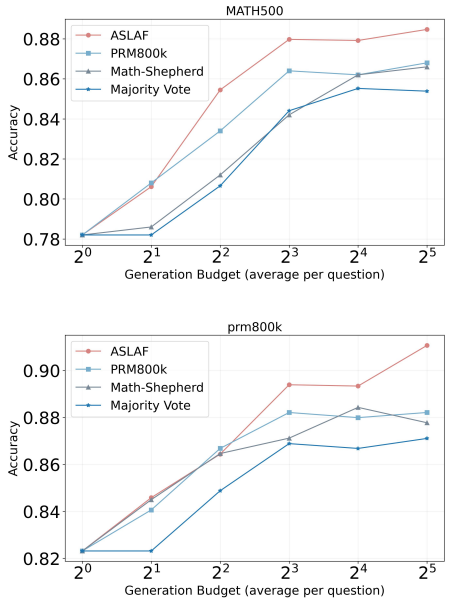

Figure 3: Performance of PRM Training on Different Datasets.

Generalization of Process Reward Model

Impact of Pre-Training and Reward Training FLOPs

Understanding the relationship between computational resources and model performance is crucial for optimizing the training of PRMs. We explore the scaling performance for training the PRM, focusing on both pre-training FLOPs, which involve different model sizes, and reward model training FLOPs, which pertain to the specific training needs of the PRM.

The training FLOPs for the PRM consist of two main components: pre-training FLOPs and reward model training FLOPs. Pre-training FLOPs involves selecting different model sizes for the PRM, which affects the initial computational requirements. Larger models generally require more FLOPs during the pre-training phase, but they also have the potential to capture more complex patterns in the data. Reward model training FLOPs per-tains to the specific computational needs during the training of the reward model itself.

> **Observation 1.** ***Diminishing Returns of PRM Scaling.*** *Larger models initially provide substantial accuracy improvements, effectively capturing the complexity required for solving mathematical problems. However, as the model size continues to increase, the rate of accuracy improvement slows, indicating diminishing returns. This trend suggests that beyond a certain point, further increasing model size results in less significant performance gains relative to the additional computational cost.*

Pre-training FLOPs are primarily determined by the size of the model selected for initial training. We use Qwen2.5s to study this problem, including pre-trained models of 7 sizes, including 0.5B, 1.5B, 3B, 7B, 14B, 32B, and 72B. Figure 1 illustrates the relationship, with the x-axis representing different model sizes of the PRM and the y-axis showing the accuracy on Math tasks, including PRM 800k and MATH-500 datasets. Key observations include: As the model size increases, accuracy initially improves rapidly, suggesting that larger models better capture the complexity required for solving mathematical problems. This quick improvement underscores the benefits of enlarging model size as initialization for PRMs. However, as the model size continues to expand, the rate of accuracy improvement decelerates, reflecting diminishing returns. This pattern implies that beyond a certain threshold, further enlarging the model yields less substantial performance improvements compared to the additional computational expense.

Reward model training FLOPs refer to the computational resources required during the PRM-specific training phase. Understanding these resources is crucial for optimizing the efficiency and effectiveness of the PRM. Figure 2 and 3 provide insights into this aspect.

Figure 2 shows the relationship between training steps and accuracy on Math tasks. As the number of training steps increases, accuracy initially remains unchanged during the early training phase. This plateau is followed by a sudden leap in accuracy, known as the ”emergence” phenomenon, after which accuracy continues to increase gradually. This pattern suggests that a critical number of training steps is required before the PRM begins to effectively leverage the information in the data, resulting in a rapid improvement in performance. The subsequent gradual increase indicates that further training continues to refine the PRM's capabilities, albeit at a slower pace.

> **Observation 2.** ***The choice and diversity of training datasets significantly impact the performance of Process Reward Models.*** *It highlights the importance of incorporating a wide range of data during reward model training to maximize accuracy and efficiency, ultimately leading to improved reasoning performance of LLMs with optimized computational resources.*

Figure 3 shows the impact of different PRM training on various datasets during inference. The x-axis represents the BON strategy with varying N, and the y-axis indicates the accuracy on Math tasks. The training datasets include the PRM800k dataset, the math-shepherd dataset, and the ASLAF dataset based on both. It highlights that PRM trained on our proposed ASLAF outperforms PRMs trained on the PRM800k and Math-Shepherd datasets, as well as the majority vote method.

The key observations are: *Dataset Influence.* The choice of training dataset significantly affects model performance. The ASLAF dataset generally provides higher accuracy, suggesting that high-quality and diverse datasets can enhance training efficiency and effectiveness. *BON Strategy Impact.* As N increases in the BON strategy, accuracy improves, demonstrating the utility of evaluating multiple candidate solutions to identify the most accurate reasoning path.

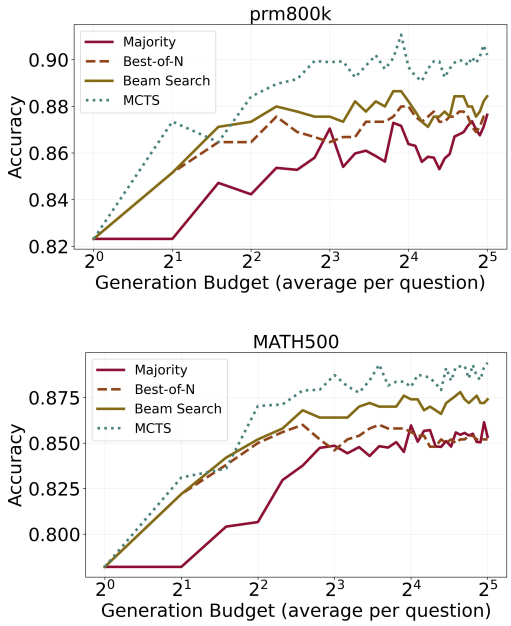

Figure 4: Impact of Different Search Strategies on Accuracy with Generation Budget. This figure examines the effect of various search strategies on accuracy, with the x-axis representing the generation budget (average per question) and the y-axis showing accuracy on Math tasks, including PRM800k and Math500. The search strategies analyzed include best-of-N, beam search, MCTS, and majority vote.

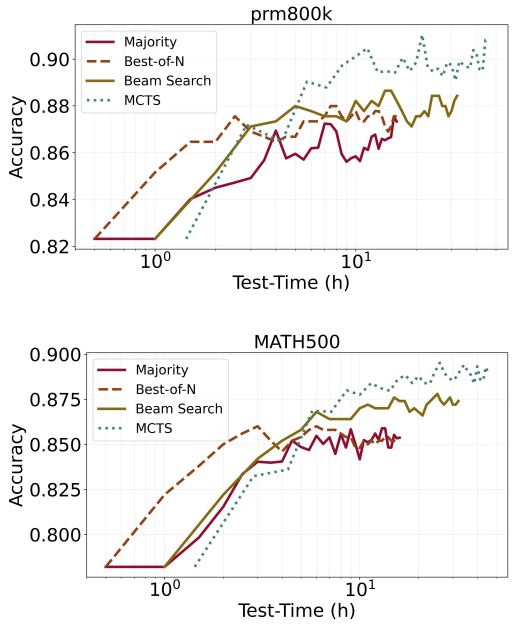

Figure 5: Impact of Different Search Strategies on Accuracy with Test-Time Constraints. This figure explores the performance of different search strategies under fixed test-time constraints, with the x-axis representing Test-Time (hours) and the y-axis showing accuracy on Math tasks, including PRM800k and Math500. The strategies include best-of-N, beam search, MCTS, and majority vote.

Trade-offs in Training and Testing FLOPs. While increasing N in the BON strategy can enhance accuracy, it also requires more computational resources, underscoring the need for a balanced approach to optimize training FLOPs.

The figure reveals that the choice of training dataset significantly impacts model performance. Using the ASLAF method to annotate step labels of PRM800k and math-shepherd results in the highest accuracy across different N values, indicating that a diverse and comprehensive training dataset enhances the model's generalization ability and performance on complex tasks. This underscores the importance of incorporating a wide range of data during reward model training to maximize accuracy.

Test-Time Scaling Performance

Test-time scaling involves optimizing the deployment of PRMs to maximize their performance across various test scenarios. This includes strategies for efficient candidate generation, scoring, and selection, as well as methods for leveraging multiple PRMs to achieve consensus and improve accuracy. To enhance the reasoning process at test time, we employ search strategies that utilize the PRM's feedback to navigate the solution space effectively. The key strategies include:

Best-of-N Sampling: Generate N candidates and select the highest-scoring solution according to a preference reward model: Best Solution $= \arg\max_{i \in \{1,2,\ldots,N\}} R_i$ This approach improves solution quality by exploring diverse reasoning paths.

Beam Search: Maintain K highest-scoring partial solutions at each step. For each path, calculate cumulative scores: Top Paths $=$ Top-K $\left\{\sum_{t=1}^{T} R(x_t)\right\}$ This efficiently balances exploration and computational resource allocation.

Monte Carlo Tree Search (MCTS): Represent the reasoning process as a tree where nodes are states and edges are actions. Use a policy, such as Upper Confidence Bound for Trees (UCT), to select the next node: $\text{UCT}(s,a) = Q(s,a) + c \cdot \sqrt{\frac{\ln N(s)}{N(s,a)}}$ where $Q(s,a)$ is the estimated value of action a from state s, $N(s)$ is the visit count of state s, and $N(s,a)$ is the visit count of action a from state s. Perform rollouts to simulate the outcome of following a particular path. Update the value estimates of nodes based on the results of the rollouts. MCTS effectively balances exploration and exploitation, allowing the model to explore vast solution spaces strategically.

Majority Voting: Generate multiple candidate solutions. Aggregate the final answers, selecting the most frequently occurring answer: Final Answer $=$ Mode$(\{A_1, A_2, \ldots, A_N\})$ Majority voting leverages the collective insights from multiple solutions, enhancing the robustness of the final answer.

We evaluate the effectiveness of test-time scaling strate-

Reward Model	HE+	MBPP+	LCB	BCB
Generate Model: Qwen2.5-Coder-7B-Instruct				
-	81.1	72.2	14.2	49.0
Majority Vote	81.5	73.7	15.6	50.7
PRM-Math	86.0	77.3	22.6	54.2
PRM-Code	84.2	78.6	18.7	52.3
Generate Model: Qwen2.5-Coder-32B-Instruct				
-	82.9	77.0	25.8	57.5
Majority Vote	85.8	77.6	28.2	59.4
PRM-Math	91.5	78.8	30.3	60.1
PRM-Code	89.0	78.4	29.7	60.1
Generate Model: QwQ-32B-Preview				
-	84.8	70.9	36.1	54.3
Majority Vote	86.8	76.0	40.1	54.6
PRM-Math	89.6	76.2	41.9	54.2
PRM-Code	86.0	79.6	41.9	53.1

Table 1: The performance of PRM models trained on different datasets for code generation tasks varies across different generative models. All the models are trained from Qwen2.5-72B. Campared to PRM trained with Math, the PRM model trained on code datasets does not exhibit a significant advantage in the task.

gies using a suite of mathematical benchmarks. These experiments demonstrate significant improvements in reasoning accuracy, underscoring the practical utility of test-time scaling strategies.

Observation 3. ***Select search strategies to optimize the performance of Process Reward Models based on the specific computational context.*** *MCTS is identified as the most effective search strategy when there is ample computational resource availability, outperforming beam search, best-of-N, and majority vote. Best-of-N initially outperforms other methods under limited test-time conditions due to its simplicity and speed.*

Figure 4 examines the effect of various search strategies on accuracy with the generation budget. As the generation budget increases, MCTS consistently outperforms other strategies, followed by beam search, best-of-N, and majority vote in descending order of accuracy. This indicates that MCTS is the most effective strategy when ample computational resources are available, as it balances exploration and exploitation efficiently. Beam search also performs well, offering a good trade-off between resource usage and accuracy. Best-of-N and majority vote, while simpler, are less effective in maximizing accuracy under higher generation budgets. Majority Voting enhances robustness, particularly in scenarios with high variability in candidate solutions. Search strategies with PRM such as MCTS, beam search, best-of-N outperform majority vote, achieving higher accuracy and better error localization capabilities. Integrating the PRM into the decoding process leads to substantial improvements in reasoning accuracy, as the model receives continuous feedback and refines its reasoning iteratively. These findings underscore the importance of test-time scaling in enhancing the practical utility of PRMs.

Moreover, Figure 5 explores performance under fixed test-time constraints. Unlike the generation budget scenario, best-of-N initially outperforms other methods under limited test-time conditions due to its simplicity and speed. However, as more test-time is allocated, MCTS surpasses other strategies, demonstrating its superior ability to explore the solution space effectively given sufficient time. Beam search also shows competitive performance but requires more time to achieve results comparable to MCTS. Majority vote remains the least effective strategy under time constraints, highlighting its limitations in leveraging extensive search.

These findings underscores the importance of selecting search strategies based on available computational resources and time constraints.

Generalization Across Domains

To investigate the generalization capabilities of PRMs, we independently train PRMs on domain-specific corpora (math and code datasets) and rigorously assess their performance on cross-domain code generation benchmarks.

Observation 4. ***Generalization from Math to Code.*** *PRMs trained on math corpora demonstrate comparable performance to those optimized on code-centric datasets in code generation tasks.*

Generalization from Math to Code Domain As evidenced by the experimental results in Table 1, PRMs consistently enhance model performance across most evaluation tasks. Specifically, LLMs augmented with PRMs trained with math domain, achieve a 4% average performance improvement in code generation benchmarks. Notably, PRMs trained on ASLAF datasets exhibit superior generalization capabilities compared to those trained solely on code-centric datasets.

Pattern Similarity To assess whether PRM captures general reasoning patterns across mathematical and coding tasks, we employ a gradient-based similarity metric to evaluate the internal patterns in responses (Wang et al. 2025). Specifically, for a reference model $\mathscr{F}$ and an input sentence s, we define the activation of the i-th weight θ_i^r as:

$$A(\theta^r, s)_i = |\theta_i^r \cdot \frac{\partial \mathscr{F}(\theta_i^r, s)}{\partial \theta_i^r}|.$$

By concatenating all A_i, we obtain a vector A that represents the activation patterns of the reference model for the given input. Consequently, we can evaluate the similarity between two reasoning outputs sets S_1, S_2, based on their pattern similarity using the following formula:

$$\mathscr{S} = \sum_{s_1 \in S_1, s_2 \in S_2} \frac{\nabla A(\theta^r, s_1) \cdot \nabla A(\theta^r, s_2)}{||A(\theta^r, s_1)||_2, ||A(\theta^r, s_2)||_2}$$

As the result in Table 2, the pattern similarity between Math_{PRM} and Code_{PRM} exceeds that of other pairs, sug-

S_1	S_2	$\mathscr{S}$
Math_{PRM}	Code_{PRM}	30.95
Math_{PRM}	Code	29.07
Math	Code_{PRM}	29.42
Math	Code	26.75

Table 2: Similarity of responses from Qwen2.5-Coder-32B-Instruct across code and math domains. Here, "Math" denotes responses to math tasks, while "Math_{PRM}" are those selected by the PRM-Math; "Code" and "Code_{PRM}" follow the same convention.

gesting that the PRM model identifies cross-domain responses with similar patterns. For instance, in Figure 6, responses with rethinking patterns receive higher scores from PRM.

Related Work

Process Reward Models PRMs enhance reasoning by providing intermediate feedback. Recent studies (Zhang et al. 2025; Lightman et al. 2023; Chen et al. 2025a) address annotation and evaluation challenges using methods like consensus filtering. Algorithmically, Ma, Cao, and Chee (2024) proposed a heuristic greedy search, while Zhang et al. (2024b) introduced an entropy-regularized PRM. Math-Shepherd (Wang et al. 2024) minimizes manual effort by automating annotation. Finally, Setlur et al. (2024) developed Process Advantage Verifiers to optimize efficiency in reinforcement learning.

Test-time Scaling Efficient allocation of test-time compute, as illustrated by Setlur et al. (2024); Meituan (2025); Chen et al. (2024b), can significantly outperform larger models. Chen et al. (2024a) proposed a two-stage algorithm that reduces failure rates exponentially with increased compute, while Zhang (Zhang et al. 2024a) found that model scaling offers greater benefits than data scaling. Yue et al. (2024) highlighted performance gains through optimal inference scaling. Wu et al. (2024); Beeching, Tunstall, and Rush (2024) have discussed test-time scaling, but they are all limited to the mathematical domain.

Scaling of RL Research into scaling laws in reinforcement learning and reward modeling has identified consistent power-law relationships (Chen et al. 2025b). Gao (Gao, Schulman, and Hilton 2023) explored distinctions between optimization methods, and Neumann (Neumann and Gros 2024) noted scaling effects in multi-agent RL. Rafailov (Rafailov et al. 2024) and Hilton (Hilton, Tang, and Schulman 2023) examined overoptimization and intrinsic performance, respectively, illustrating the importance of understanding scaling laws for AI alignment and model design.

Limitations

This study is subject to several limitations that warrant consideration when interpreting and generalizing the presented findings.

Question:

You are given a string s and an integer k. Encrypt the string using the following algorithm: For each character c in s, replace c with the k^{th} character after c in the string (in a cyclic manner). Return the encrypted string.
...

Reason:

Step 1:

```
...
# Find the k-th character after 'char'
    in a cyclic manner
new_char = s[(s.index(char) + k) %
    length]
...
```

0.5371

Step 2:
However, the above solution uses 's.index(char)' which will not work correctly if there are duplicate characters in the string 's'. Instead, we use current index of the character in the loop. Here is the corrected solution:

0.7563

Step 3:

```
...
# Find the k-th character after the
current character in a cyclic manner
new_char = s[(i + k) % length]
...
```

0.875

Figure 6: PRM score for a case with a correct answer. The full context is provided in Figure 2 in the Appendix.

1. **Computational Resources:** The scope of our experimentation was inherently limited by computational resource constraints. This restricted our capacity for a more comprehensive exploration of larger model architectures and complex scaling methodologies.
2. **Task Focus:** The investigation was confined primarily to mathematical reasoning and code generation. Consequently, a complete understanding of PRM across diverse domains requires further empirical validation.

Conclusion

This paper demonstrates the potential of Process Reward Models to significantly enhance the reasoning capabilities of Large Language Models across diverse domains, particularly in transitioning from mathematical reasoning to code generation tasks. Our comprehensive analysis provides key insights into the scalability, efficiency, and adaptability of PRMs, contributing to the broader understanding of their application and development. Our findings indicate that as PRM size increases, performance improvements follow a pattern of diminishing returns, emphasizing the importance of balancing model size with computational cost. Additionally, the diversity of training sets significantly impacts PRM performance, highlighting the importance of incorporating varied data in reward model training to improve accuracy and efficiency.

References

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*.

Beeching, E.; Tunstall, L.; and Rush, S. 2024. Scaling test-time compute with open models.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Chen, Y.; Pan, X.; Li, Y.; Ding, B.; and Zhou, J. 2024a. A simple and provable scaling law for the test-time compute of large language models. *arXiv preprint arXiv:2411.19477*.

Chen, Z.; Wang, S.; Xiao, T.; Wang, Y.; Chen, S.; Cai, X.; He, J.; and Wang, J. 2025a. Revisiting scaling laws for language models: The role of data quality and training strategies. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 23881–23899.

Chen, Z.; Xiao, T.; and Kuang, K. 2022. BA-GNN: On Learning Bias-Aware Graph Neural Network. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 3012–3024. IEEE.

Chen, Z.; Xiao, T.; Kuang, K.; Lv, Z.; Zhang, M.; Yang, J.; Lu, C.; Yang, H.; and Wu, F. 2024b. Learning to Reweight for Generalizable Graph Neural Network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 8320–8328.

Chen, Z.; Yang, J.; Xiao, T.; Zhou, R.; Zhang, L.; Xi, X.; Shi, X.; Wang, W.; and Wang, J. 2025b. Can Tool-Integrated Reinforcement Learning Generalize Across Diverse Domains? *arXiv preprint arXiv:2510.11184*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*.

Gao, L.; Schulman, J.; and Hilton, J. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, 10835–10866. PMLR.

Hendrycks, D.; Basart, S.; Kadavath, S.; Mazeika, M.; Arora, A.; Guo, E.; Burns, C.; Puranik, S.; He, H.; Song, D.; and Steinhardt, J. 2021. Measuring Coding Challenge Competence With APPS. *NeurIPS*.

Hilton, J.; Tang, J.; and Schulman, J. 2023. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*.

Li, R.; Fu, J.; Zhang, B.-W.; Huang, T.; Sun, Z.; Lyu, C.; Liu, G.; Jin, Z.; and Li, G. 2023. TACO: Topics in Algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*.

Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Dal Lago, A.; Hubert, T.; Choy, P.; de Masson d'Autume, C.; Babuschkin, I.; Chen, X.; Huang, P.-S.; Welbl, J.; Gowal, S.; Cherepanov, A.; Molloy, J.; Mankowitz, D.; Sutherland Robson, E.; Kohli, P.; de Freitas, N.; Kavukcuoglu, K.; and Vinyals, O. 2022. Competition-Level Code Generation with AlphaCode. *arXiv preprint arXiv:2203.07814*.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2023. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050*.

Liu, J.; Xia, C. S.; Wang, Y.; and Zhang, L. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36: 21558–21572.

Luo, L.; Liu, Y.; Liu, R.; Phatale, S.; Lara, H.; Li, Y.; Shu, L.; Zhu, Y.; Meng, L.; Sun, J.; et al. 2024. Improve Mathematical Reasoning in Language Models by Automated Process Supervision. *arXiv preprint arXiv:2406.06592*.

Ma, Y.; Cao, Z.; and Chee, Y. M. 2024. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems*, 36.

Meituan. 2025. LongCat-Flash-Thinking Technical Report. arXiv:2509.18883.

Neumann, O.; and Gros, C. 2024. AlphaZero Neural Scaling and Zipf's Law: a Tale of Board Games and Power Laws. *arXiv preprint arXiv:2412.11979*.

Rafailov, R.; Chittepu, Y.; Park, R.; Sikchi, H.; Hejna, J.; Knox, B.; Finn, C.; and Niekum, S. 2024. Scaling laws for reward model overoptimization in direct alignment algorithms. *arXiv preprint arXiv:2406.02900*.

Setlur, A.; Nagpal, C.; Fisch, A.; Geng, X.; Eisenstein, J.; Agarwal, R.; Agarwal, A.; Berant, J.; and Kumar, A. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*.

Wang, P.; Li, L.; Shao, Z.; Xu, R.; Dai, D.; Li, Y.; Chen, D.; Wu, Y.; and Sui, Z. 2024. 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)*, 9426–9439.

Wang, Y.; Dai, D.; Yang, Z.; Ma, J.; and Sui, Z. 2025. Exploring activation patterns of parameters in language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 25416–25424.

Wu, Y.; Sun, Z.; Li, S.; Welleck, S.; and Yang, Y. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Yang, A.; Zhang, B.; Hui, B.; Gao, B.; Yu, B.; Li, C.; Liu, D.; Tu, J.; Zhou, J.; Lin, J.; et al. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Yue, Z.; Zhuang, H.; Bai, A.; Hui, K.; Jagerman, R.; Zeng, H.; Qin, Z.; Wang, D.; Wang, X.; and Bendersky, M. 2024. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*.

Zhang, B.; Liu, Z.; Cherry, C.; and Firat, O. 2024a. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv preprint arXiv:2402.17193*.

Zhang, H.; Wang, P.; Diao, S.; Lin, Y.; Pan, R.; Dong, H.; Zhang, D.; Molchanov, P.; and Zhang, T. 2024b. Entropy-Regularized Process Reward Model. *arXiv preprint arXiv:2412.11006*.

Zhang, Z.; Zheng, C.; Wu, Y.; Zhang, B.; Lin, R.; Yu, B.; Liu, D.; Zhou, J.; and Lin, J. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.