GenPRM: Scaling Test-Time Compute of Process Reward Models via Generative Reasoning

Jian Zhao[1,3*], **Runze Liu**[1,2*†], **Kaiyan Zhang**[1], **Zhimu Zhou**[3], **Junqi Gao**[4], **Dong Li**[4], **Jiafei Lyu**[1], **Zhouyi Qian**[4], **Biqing Qi**[2‡], **Xiu Li**[1‡], **Bowen Zhou**[1,2‡]

[1]Tsinghua University
[2]Shanghai AI Laboratory
[3]Beijing University of Posts and Telecommunications
[4]Harbin Institute of Technology

Abstract

Recent advancements in Large Language Models (LLMs) have shown that it is promising to utilize Process Reward Models (PRMs) as verifiers to enhance the performance of LLMs. However, current PRMs face three key challenges: (1) limited process supervision and generalization capabilities, (2) dependence on scalar value prediction without leveraging the generative abilities of LLMs, and (3) inability to scale the test-time compute of PRMs. In this work, we introduce GenPRM, a generative process reward model that performs explicit Chain-of-Thought (CoT) reasoning with code verification before providing judgment for each reasoning step. To obtain high-quality process supervision labels and rationale data, we propose Relative Progress Estimation (RPE) and a rationale synthesis framework that incorporates code verification. Experimental results on ProcessBench and several mathematical reasoning tasks show that GenPRM significantly outperforms prior PRMs with only 23K training data from MATH dataset. Through test-time scaling, a 1.5B GenPRM outperforms GPT-4o, and a 7B GenPRM surpasses Qwen2.5-Math-PRM-72B on ProcessBench. Additionally, GenPRM demonstrates strong abilities to serve as a critic model for policy model refinement. This work establishes a new paradigm for process supervision that bridges the gap between PRMs and critic models in LLMs.

Code — github.com/RyanLiu112/GenPRM

Datasets — huggingface.co/datasets/GenPRM/GenPRM-MATH-Data

Extended version — arxiv.org/pdf/2504.00891

1 Introduction

Large Language Models (LLMs) have shown significant advances in recent years (OpenAI 2023; Anthropic 2023; OpenAI 2024a,b; DeepSeek-AI et al. 2025; Zhang et al. 2025c; Sun et al. 2025). As OpenAI o1 demonstrates the great effectiveness of scaling test-time compute (OpenAI 2024a), an increasing number of researches focus on Test-Time Scaling (TTS) methods to improve the reasoning performance of LLMs (Snell et al. 2025; Liu et al. 2025a; Zhang et al. 2025a; Wang et al. 2025b; Zeng et al. 2025a; Liu et al. 2025b; Wang et al. 2025a).

Effective TTS requires high-quality verifiers, such as Process Reward Models (PRMs) (Liu et al. 2025a). However, existing PRMs face several limitations. They exhibit limited process supervision capabilities and struggle to generalize across different models and tasks (Zheng et al. 2024; Zhang et al. 2025e; Liu et al. 2025a). Furthermore, most current approaches train PRMs as classifiers that output scalar values, neglecting the pre-trained natural language generation abilities of LLMs. This classifier-based modeling inherently prevents PRMs from leveraging test-time scaling methods to enhance process supervision capabilities. These limitations lead us to the following research question: ***How can generative modeling enhance the process supervision capabilities of PRMs while enabling test-time scaling?***

In this work, we address these challenges through a generative process reward model, named GenPRM. Specifically, GenPRM differs from classification-based PRMs in that GenPRM redefines process supervision as a generative task rather than a discriminative scoring task by integrating Chain-of-Thought (CoT) (Wei et al. 2022) reasoning and code verification processes before providing final judgment. To improve conventional hard label estimation, we propose Relative Progress Estimation (RPE), which leverages a relative criterion for label estimation. Additionally, we introduce a rationale synthesis framework with code verification to obtain high-quality process supervision reasoning data. A comparison of our method with previous classification-based methods is presented in Figure 1.

Our contributions can be summarized as follows:

1. We propose a generative process reward model that performs explicit CoT reasoning with code verification and utilizes Relative Progress Estimation to obtain accurate PRM labels.
2. Empirical results on ProcessBench and common mathe-

*These authors contributed equally.
†Project lead & Work done during an internship at Shanghai AI Laboratory.
‡Corresponding author.

matical reasoning tasks demonstrate that GenPRM outperforms prior classification-based PRMs. Additionally, smaller GenPRM models can surpass larger PRMs via TTS.

3. We provide a new perspective on PRMs in this work, fully leveraging their TTS capabilities, reshaping their applications, and opening new directions for future research in process supervision.

2 Related Work

Process Reward Models. Process Reward Models (PRMs) have proven superior to Outcome Reward Models (ORMs) for step-wise scoring in mathematical reasoning (Uesato et al. 2022; Lightman et al. 2024). Due to the significant human cost of annotating process supervision datasets like PRM800K, prior works have developed automated label generation methods, such as Monte Carlo estimation (Wang et al. 2024b) and binary search (Luo et al. 2024). Subsequent research improves PRMs through methods such as advantage modeling (Setlur et al. 2025), Q-value rankings (Li and Li 2025), implicit entropy regularization (Zhang et al. 2024a), retrieval-augmented generation (Zhu et al. 2025), and fast-slow verification (Zhong et al. 2025). Furthermore, the community has developed high-quality open-source PRMs, including the RLHFlow series (Xiong et al. 2024), Math-psa (Wang et al. 2024a), Skywork series (Skywork o1 Team 2024), and Qwen2.5-Math series (Zheng et al. 2024; Zhang et al. 2025e). Recently, some works focus on extending PRMs to other tasks, including coding (Zhang et al. 2024b), medical tasks (Jiang et al. 2025), agentic tasks (Choudhury 2025), general domain tasks (Zhang et al. 2025b; Zeng et al. 2025b), and multimodal tasks (Wang et al. 2025c). Current studies also focus on benchmarking PRMs (Zheng et al. 2024; Song et al. 2025) to systematically evaluate their performance.

Large Language Model Test-Time Scaling. Scaling test-time computation is an effective method for improving performance during the inference phase (OpenAI 2024a; DeepSeek-AI et al. 2025). TTS is commonly implemented with external verifiers (e.g., ORMs and PRMs) or strategies (e.g., beam search and MCTS) (Wu et al. 2025; Snell et al. 2025; Beeching, Tunstall, and Rush 2024; Liu et al. 2025a). In this work, we scale the test-time computation of a generative PRM with an explicit reasoning process and GenPRM can also serve as a verifier or a critic model in external TTS.

Enhancing the Generative Abilities of Reward Models. Previous research has investigated methods to enhance the generative capabilities of reward models using CoT reasoning (Ankner et al. 2024; Zhang et al. 2025d; Mahan et al. 2024). For instance, CLoud reward models (Ankner et al. 2024) are trained to generate critiques for responses and predict rewards using an additional reward head. GenRM-CoT (Zhang et al. 2025d) and GenRM (Mahan et al. 2024) train generative reward models that perform CoT reasoning before making final predictions via SFT and preference learning, respectively. CTRL (Xie et al. 2025) demonstrates that critic models exhibit strong discriminative abilities when utilized as generative reward models. Prior to these works, GRM (Yang et al. 2024c) regularizes the hidden states of reward models with a text generation loss.

3 Preliminaries

3.1 Test-Time Scaling

In this work, we consider three test-time scaling methods.

Majority Voting. Majority voting (Wang et al. 2023) selects the answer that appears the most frequently among all solutions.

Best-of-N. Best-of-N (BoN) (Brown et al. 2024; Snell et al. 2025) selects the best answer from N candidate solutions.

Critique and Refinement. In this process, a critic model finds errors in a solution and provides feedback. The original model then uses this critique to generate a new, improved answer recursively (Madaan et al. 2023).

4 Method

In this section, we first describe how to develop GenPRM and integrate the reasoning process with code verification. Then we introduce how to scale test-time compute of policy models using GenPRM and apply TTS for GenPRM and present the improved label estimation method and data generation and filtering framework of GenPRM.

4.1 GenPRM and Test-Time Scaling

From Discriminative PRM to Generative PRM

Discriminative PRM. Assume we have a PRM dataset $\mathcal{D}_{\text{Disc}} = \{(s_t, a_t), r_t\}$, where $r_t \in \{0, 1\}$ for PRM labels with hard estimation. The discriminative PRM r_ψ is trained via cross-entropy loss (Skywork o1 Team 2024; Zhang et al. 2025e):

$$\begin{aligned}\mathcal{L}_{\text{CE}}(\psi) = -\,\mathbb{E}_{(s_t,a_t,r_t)\sim\mathcal{D}_{\text{Disc}}}[r_t \log r_\psi(s_t, a_t) + \\ (1 - r_t)\log(1 - r_\psi(s_t, a_t))].\end{aligned} \tag{1}$$

Direct Generative PRM. With a dataset $\mathcal{D}_{\text{Direct-Gen}} = \{(s_t, a_t), r_t\}$, where r_t is `Yes` for a correct step and `No` otherwise, the direct generative PRM (Xiong et al. 2024) is trained through SFT to predict `Yes` or `No` for each step. For step t, we use the probability of the `Yes` token as the predicted process reward $\hat{r}_t$:

$$\hat{r}_t = r_\psi(\texttt{Yes} \mid s_t, a_t). \tag{2}$$

Generative PRM. By equipping the direct generative PRM with an explicit reasoning process like CoT (Wei et al. 2022), we obtain a generative PRM. Let $v_{1:t-1}$ denote the rationale from step 1 to $t-1$ and v_t denote the rationale for step t. Assume we have a dataset $\mathcal{D}_{\text{Gen}} = \{(s_t, a_t, v_{1:t-1}), (v_t, r_t)\}$. GenPRM learns to reason and verify each step via SFT on this dataset. The generative process reward $\hat{r}_t$ can be obtained via the following equation:

$$\hat{r}_t = r_\psi(\texttt{Yes} \mid s_t, a_t, v_{1:t-1}, v_t), \tag{3}$$

where $v_t \sim r_\psi(\cdot \mid s_t, a_t, v_{1:t-1})$.

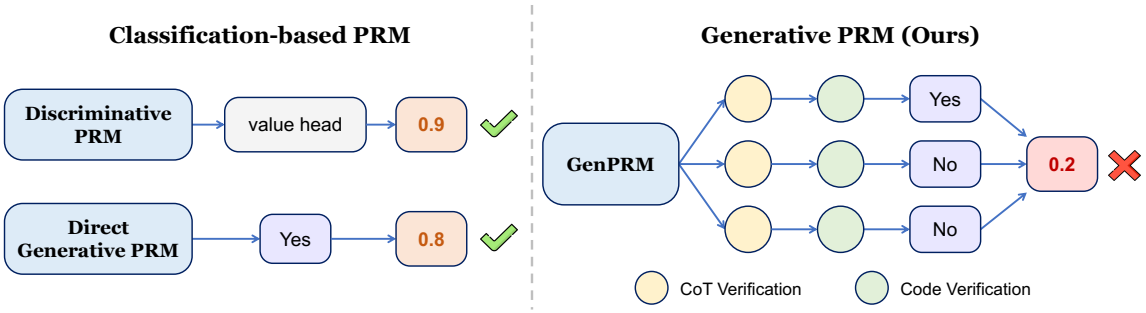


Figure 1: A comparison between previous classification-based PRMs (left) and GenPRM (right). In this hypothetical illustration, the classification-based PRMs erroneously judge an example trace as good, while GenPRM is able to leverage test-time scaling abilities (with CoT and code verification) to correctly identify errors, as shown in Table 8 in Appendix B (Zhao et al. 2025).

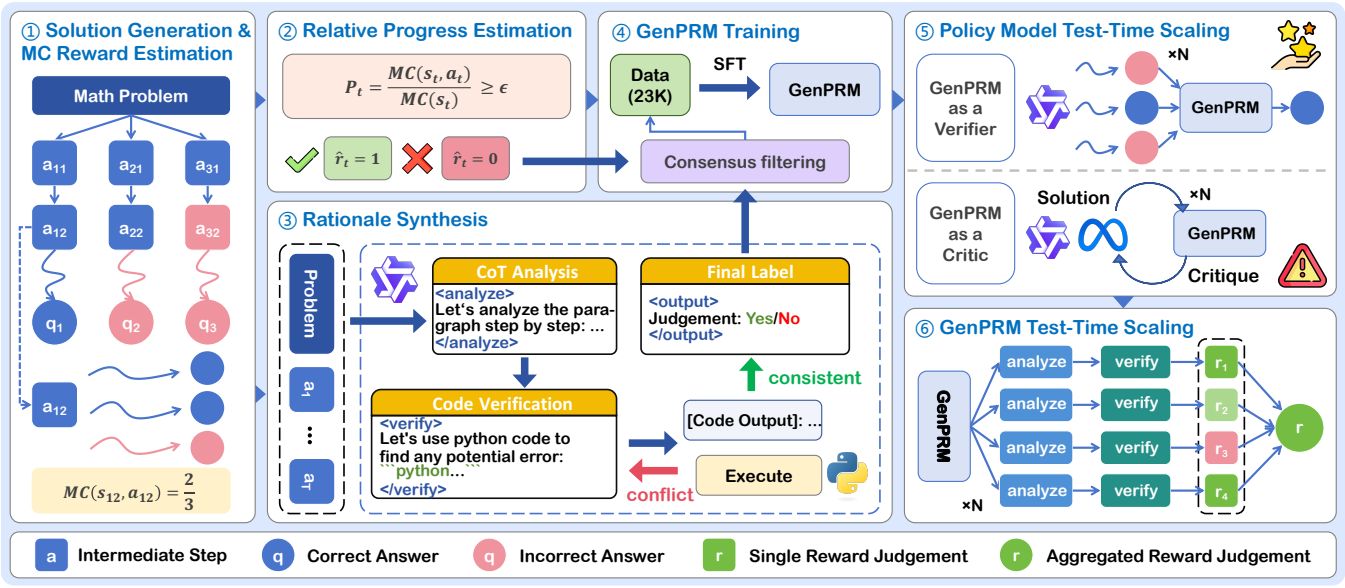


Figure 2: Our framework consists of six key parts: (1) The policy model generates solution steps, with MC scores estimated from rollout trajectories (§4.2). (2) Our proposed RPE derives accurate PRM labels (§4.2). (3) High-quality process supervision data is synthesized through CoT reasoning augmented with code verification (§4.2). (4) We apply consensus filtering followed by SFT to train GenPRM. (5) The trained GenPRM functions as a verifier or critic, enabling enhanced test-time scaling for policy models (§4.1). (6) The performance of GenPRM further improves through test-time scaling (§4.1).

Generative PRM with Code Verification. If we only verify the reasoning step with CoT based on natural language, the process may lack robustness in certain complex scenarios (Zhu et al. 2024; Gou et al. 2024). The difference between the generative PRM and the generative PRM with code verification is that the latter generates code to verify the reasoning step by executing it and provides the judgment based on the execution results. At step t, after generating the rationale v_t containing CoT and code, we execute the code and obtain feedback f_t. Given the current state s_t, action a_t, previous rationales $v_{1:t-1}$, and previous corresponding execution feedback $f_{1:t-1}$, the PRM first generates the rationale v_t. After execution and obtaining the feedback f_t, we compute the final generative process reward as follows:

$$\hat{r}_t = r_\psi(\texttt{Yes} \mid s_t, a_t, v_{1:t-1}, f_{1:t-1}, v_t, f_t), \quad (4)$$

where $v_t \sim r_\psi(\cdot \mid s_t, a_t, v_{1:t-1}, f_{1:t-1})$. In the following sections, we refer to GenPRM as this generative PRM type with code verification. The effectiveness of CoT and code verification can be found in Section 5.4.

Test-Time Scaling

Policy Model Parallel TTS: GenPRM as a Verifier. To scale the test-time compute of policy models, we can sample multiple responses from policy models and then use GenPRM as a verifier to select the final answer (Snell et al. 2025).

Policy Model Sequential TTS: GenPRM as a Critic. By equipping the PRM with generative process supervision abilities, GenPRM can be naturally used as a critic model to refine the outputs of policy models and we can scale the refinement process with multiple turns.

GenPRM TTS. In addition to scaling the policy model, we can also scale the test-time compute of GenPRM. When evaluating each solution step, we first sample N reasoning verification paths and then use majority voting to obtain the final prediction by averaging the rewards. For GenPRM without code verification, the rewards are computed as follows:

$$\hat{r}_t = \frac{1}{N}\sum_{i=1}^{N} r_\psi(\texttt{Yes} \mid s_t, a_t, v^i_{1:t-1}, v^i_t). \quad (5)$$

And we can further incorporate code verification and execution feedback into this reasoning process:

$$\hat{r}_t = \frac{1}{N}\sum_{i=1}^{N} r_\psi(\texttt{Yes} \mid s_t, a_t, v^i_{1:t-1}, f^i_{1:t-1}, v^i_t, f^i_t). \quad (6)$$

Then the rewards can be used for ranking the responses of policy models. The aggregation of step-level rewards is discussed in Appendix A (Zhao et al. 2025). Also it could be converted into binary labels through a threshold 0.5 for judging the correctness of the step.

4.2 Synthesizing Data of GenPRM

Solution Generation and Monte Carlo Estimation In this part, we introduce how to generate solution data and obtain initial process reward labels via Monte Carlo estimation.

Solution Generation with Step Forcing. We utilize the 7.5K problems from the training set of the MATH dataset (Hendrycks et al. 2021) as the problem set. For each problem, we use Qwen2.5-7B-Instruct (Yang et al. 2024a) as the policy model to collect multiple solutions. Since using "`\n\n`" for step division does not consider the semantics of each step and may result in overly fine-grained division, we apply a step forcing approach to generate solutions. Specifically, we add "`Step 1:`" as the prefix for the policy model to complete the response. For a response with T reasoning steps, the format is as follows:

The response format with step forcing

Step 1: {step content} ... **Step T:** {step content}

The proportion of correct versus incorrect paths varies significantly depending on problem difficulty. To obtain a sufficient number of both, we sample up to 2048 paths per problem. If either correct or incorrect paths are missing after sampling 2048 responses, the corresponding problems are discarded.

Balancing the Precision and Efficiency of MC Estimation. Following Math-Shepherd (Wang et al. 2024b), we estimate the probability of correctness for each step using completion-based sampling. For each reasoning step s_t, we generate K completion trajectories using a completion model, specifically Qwen2.5-Math-7B-Instruct (Yang et al. 2024b), and use MC estimation to calculate the probability that the current step a_t is correct (Wang et al. 2024b; Zhang et al. 2025e):

$$MC(s_t, a_t) = MC(s_{t+1}) = \frac{1}{K}\sum_{j=1}^{K} \mathbb{1}(q_j = q^*), \quad (7)$$

where q_j is the answer of the j-th response, q^* is the ground-truth answer, and $\mathbb{1}$ is the indicator function. However, it is difficult for the completion model to reach the correct answer for hard problems even when the original step is correct, leading to incorrect results for MC estimation. To address this and balance the computation cost, we use a dynamic K based on the estimated Pass@1 $MC(s_1)$:

$$K = \begin{cases} 128 & \text{if } 0 \le MC(s_1) < 0.1, \\ 64 & \text{if } 0.1 \le MC(s_1) < 0.9, \\ 32 & \text{if } 0.9 \le MC(s_1) < 1. \end{cases} \quad (8)$$

Relative Progress Estimation Previous work has shown that hard label estimation is better than soft label estimation for PRMs (Zhang et al. 2025e). However, after MC estimation, we observe that although the MC score of many steps is greater than 0, the steps are incorrect, as also noted by Zhang et al. (2025e). For this issue, we propose Relative Progress Estimation (RPE), which assumes that a reasoning step should be considered as a beneficial one if it is easier to reach the correct answer by adding this step as the generation prefix.

Specifically, the MC score is an empirical estimation of the current state s_t. To evaluate the quality of the current action a_t, it is natural to compare the MC score of the next state s_{t+1} with that of the current state s_t, since $s_{t+1} = [s_t, a_t]$. For each response, if the first erroneous step is step t' (i.e., $MC(s_{t'}) = 0$), we set the MC score of the following steps to 0. Our RPE P_t for step t is defined as follows:

$$P_t = \frac{MC(s_t, a_t)}{MC(s_t)}, \quad (9)$$

where $MC(s_1)$ is the estimated Pass@1 computed in the solution generation phase. However, we empirically find that using a strict criterion where progress is always greater than 1 leads to unsatisfactory performance, as shown in Table 2. To address this, we estimate the final reward label $\hat{r}_t$ by introducing a threshold ϵ:

$$\hat{r}_t = \begin{cases} 1 & \text{if } P_t \ge \epsilon, \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

We also discuss another form of relative progress $P_t = MC(s_t, a_t) - MC(s_t)$ in Table 2 in Section 5.4.

Rationale Generation and Filtering To generate our high-quality training data, we employ a three-stage automated pipeline using QwQ-32B (Qwen Team 2025) as the rationale generator.

Step 1: Code-Based Rationale Generation. For each reasoning step a_t in a solution, the model generates both a natural language CoT analysis and a Python script to programmatically verify its correctness, a method shown to improve verification (Zhu et al. 2024).

Step 2: Code Execution and Verification. The script is executed, and its output is fed back to the model. We observe that this allows for a form of self-correction if the code's result contradicts the initial CoT analysis.

Step 3: Label Judgment and Consensus Filtering. Finally, we apply a strict **consensus filtering** to ensure data quality. We compare the labels derived from our RPE method with labels generated by an LLM-as-a-judge (Zheng et al. 2023). Entire solutions with any label inconsistency are discarded. This rigorous process filters out approximately 51% of the candidates, yielding a final, high-fidelity dataset of 23K examples for training GenPRM.

5 Experiments

In this section, we aim to answer the following questions:

- **Q1:** How does GenPRM perform compared with previous PRMs? (§5.2, §5.3)
- **Q2:** How does the performance of GenPRM scale with more test-time compute? (§5.2, §5.3)
- **Q3:** How does GenPRM benefit policy model test-time scaling? (§5.3)
- **Q4:** How do the components and hyperparameters influence GenPRM? (§5.4)

5.1 Setup

Benchmarks. We evaluate GenPRM and baseline methods on ProcessBench (Zheng et al. 2024), a benchmark designed to assess process supervision capabilities in mathematical reasoning tasks.[1] Additionally, we conduct BoN and critic refinement experiments using MATH (Hendrycks et al. 2021), AMC23 (AI-MO 2024b), AIME24 (AI-MO 2024a), and Minerva Math (Lewkowycz et al. 2022). For BoN response generation, we employ Qwen2.5-Math-7B-Instruct (Yang et al. 2024b) and Gemma-3-12b-it (Gemma Team and Google DeepMind 2025) as policy models. For policy model TTS with GenPRM as the critic, we use Gemma-3-12b-it (Gemma Team and Google DeepMind 2025) and Qwen2.5-7B-Instruct (Yang et al. 2024a) as generators.

Baselines. For ProcessBench and BoN experiments, we compare GenPRM with the following methods:

- **Math-Shepherd-PRM-7B** (Wang et al. 2024b): This method trains a PRM using hard labels computed based on MC estimation.
- **RLHFlow series** (Xiong et al. 2024): Includes RLHFlow-PRM-Mistral-8B and RLHFlow-PRM-Deepseek-8B.
- **Skywork-PRM series** (Skywork o1 Team 2024): Comprises Skywork-PRM-1.5B and Skywork-PRM-7B.
- **EurusPRM** (Cui et al. 2025): EurusPRM-Stage1 and EurusPRM-Stage2 are trained as implicit PRMs (Yuan et al. 2024).
- **Qwen2.5-Math series** (Zheng et al. 2024; Zhang et al. 2025e): Qwen2.5-Math-7B-Math-Shepherd and Qwen2.5-Math-7B-PRM800K are trained with Math-Shepherd (Wang et al. 2024b) and PRM800K (Lightman et al. 2024), respectively. For Qwen2.5-Math-PRM-7B and Qwen2.5-Math-PRM-72B, the training data is applied consensus filtering using LLM-as-a-judge (Zheng et al. 2023).
- **RetrievalPRM-7B** (Zhu et al. 2025): The method enhances PRM with retrieved questions and corresponding steps.
- **Universal-PRM-7B** (Tan et al. 2025): The method proposes an automated framework using ensemble prompting and reverse verification.
- **Dyve-14B** (Zhong et al. 2025): This method dynamically applies fast or slow verification for each reasoning step.
- **Direct Generative PRM-7B**: The method trains a direct generative PRM with the original language head via SFT using the same data as GenPRM, but without CoT and code verification.

For critic experiments, we use the following methods for comparison:

- **Self-Refine** (Madaan et al. 2023): This method uses the generator to self-critique and refine the solution.
- **DeepSeek-R1-Distill-Qwen-7B** (DeepSeek-AI et al. 2025): This model is fine-tuned based on Qwen2.5-Math-7B (Yang et al. 2024a) using high-quality reasoning data generated by DeepSeek-R1 (DeepSeek-AI et al. 2025).

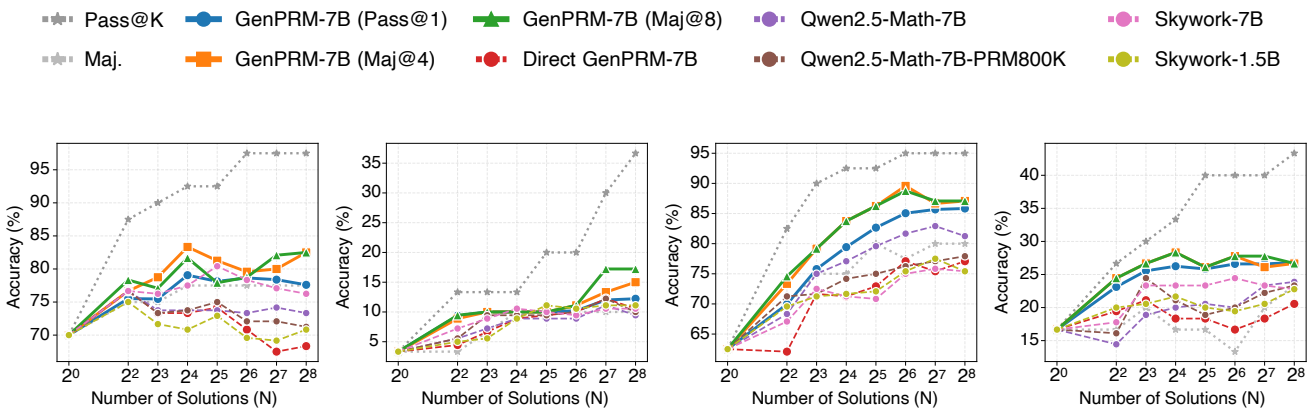
Implementation Details. For RPE, we set $\epsilon = 0.8$ across all experiments, with ablation studies presented in Section 5.4. Rationale data is generated using QwQ-32B (Qwen Team 2025) with the prompt template shown in Table 5 in Appendix A (Zhao et al. 2025). Our base models are from the DeepSeek-R1-Distill series (DeepSeek-AI et al. 2025), specifically the 1.5B, 7B, and 32B parameter variants. The training configuration for our method uses a batch size of 64 and a learning rate of 2.0×10^{-6}. During evaluation, we employ a temperature of 0.6. For critique refinement experiments, we extract content within the `<analyze></analyze>` tags, focusing exclusively on steps predicted as negative by the policy model. The baseline methods utilize standardized prompt templates (detailed in Table 7 in Appendix A (Zhao et al. 2025)) to ensure consistent critique generation formats.

5.2 ProcessBench Results

GenPRM outperforms classification-based PRMs on ProcessBench. As shown in Table 1, GenPRM-7B significantly outperforms direct generative PRM and surpasses **all** previous PRMs with parameters less than 72B on ProcessBench. Also, GenPRM-1.5B outperforms Skywork-PRM-1.5B by a large margin. It is noteworthy that GenPRM is trained with merely **23K** data from **MATH** (Hendrycks et al. 2021) only. By comparing the detailed results in Table 10 in Appendix B (Zhao et al. 2025), we find the performance gain of GenPRM mainly comes from the stronger abilities of finding **erroneous** steps and we provide concrete cases in Appendix C (Zhao et al. 2025). These results demonstrating the superiority of generative modeling of PRM.

GenPRM enables smaller PRMs surpass $10\times$ larger PRMs and GPT-4o via TTS. We also compare the TTS results of GenPRM in Table 1 and find that GenPRM-1.5B surpasses GPT-4 and GenPRM-7B exceeds Qwen2.5-Math-PRM-72B on ProcessBench via simply majority voting,

[1] Our evaluation code is adapted from https://github.com/QwenLM/ProcessBench.



(a) AMC23 (Qwen) (b) AIME24 (Qwen) (c) AMC23 (Gemma) (d) AIME24 (Gemma)

Figure 3: BoN results on AMC23 and AIME24. Full results are shown in Appendix B (Zhao et al. 2025).

Model	Training Data	F1 Score
LLMs		
GPT-4o-0806	unk	61.9
o1-mini	unk	87.9
QwQ-32B	unk	83.7
PRMs (1.5B)		
Skywork-PRM-1.5B	unk	36.4
GenPRM-1.5B (Pass@1)	**23K**	**57.3**
GenPRM-1.5B (Maj@8)	**23K**	**63.4**
PRMs (7-8B)		
Math-Shepherd-PRM-7B	445K	31.5
RLHFlow-PRM-Mistral-8B	273K	28.4
RLHFlow-PRM-Deepseek-8B	253K	26.6
Skywork-PRM-7B	unk	42.1
EurusPRM-Stage1	463K	31.2
EurusPRM-Stage2	30K	31.3
Qwen2.5-Math-7B-Math-Shepherd	445K	28.9
Qwen2.5-Math-7B-PRM800K	264K	56.5
Qwen2.5-Math-PRM-7B	∼344K	73.5
RetrievalPRM-7B	404K	65.8
Universal-PRM-7B	unk	74.3
Direct Generative PRM-7B	23K	60.0
GenPRM-7B (Pass@1)	**23K**	**75.2**
GenPRM-7B (Maj@8)	**23K**	**80.5**
PRMs (14-72B)		
Dyve-14B	117K	55.8
Qwen2.5-Math-PRM-72B	∼344K	78.3
GenPRM-32B (Pass@1)	**23K**	77.6
GenPRM-32B (Maj@8)	**23K**	**82.6**

Table 1: ProcessBench results reported with F1 scores. For 1.5B PRMs, **bold** indicates the best Pass@1 or scores superior to GPT-4o. For 7-8B and 14-72B PRMs, **bold** denotes the best Pass@1 or scores superior to Qwen2.5-Math-PRM-72B.

showing that scaling test-time compute is highly effective for GenPRM. We also find that the performance improvement of scaling the test-time compute on **harder** problems is larger than that of easier questions.

5.3 Policy Model Test-Time Scaling Results

GenPRM as a Verifier. The results in Figure 3 (a)-(b) show that GenPRM outperforms the baselines on AMC23 and AIME24 with Qwen2.5-Math-7B-Instruct (Yang et al. 2024b) as the policy model. The advantage of GenPRM becomes larger by scaling the test-time compute of GenPRM and the policy model. Figure 3 (c)-(d) demonstrates that GenPRM generalizes well to responses with Gemma-3-12b-it (Gemma Team and Google DeepMind 2025) as the policy model.

GenPRM as a Critic. We also conduct experiments by using GenPRM as a critic to refine the outputs of the policy model. The results in Figure 4 show that GenPRM exhibits strong critique abilities than the baselines, significantly improving the performance of the policy model and the performance continues to increase with more refinement based on the critic feedback.

Trade-off between performance and token consumption. We conduct experiments analyzing the trade-off between performance and token budgets. The results in Figure 5 show that under small budgets, scaling the policy with majority voting is more effective. However, under large budgets, increasing GenPRM's test-time compute yields better performance gains. This indicates a compute-performance trade-off between PRM and policy. The results of adaptive pruning of GenPRM are shown in Appendix B (Zhao et al. 2025).

5.4 Analysis

Label Estimation Method and Criterion. To explore how different label estimation affects GenPRM, we conduct experiments with the following methods: (1) hard label (Wang et al. 2024b; Zhang et al. 2025e); (2) RPE in (9); and (3) a RPE variant ($P_t = MC(s_t, a_t) - MC(s_t)$). For the RPE

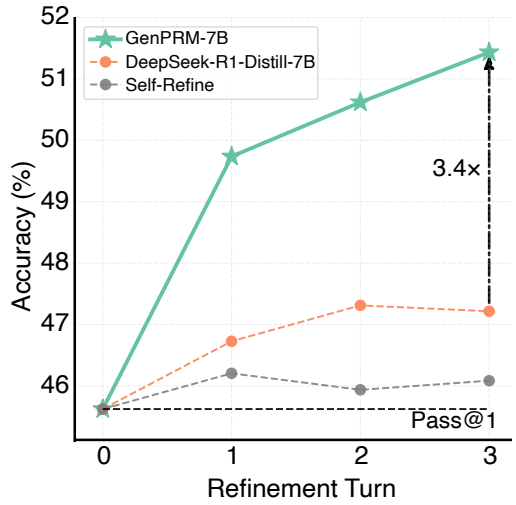


Figure 4: Results of critique refinement experiments. The reported performance is an average calculated over 4 benchmarks: AMC23, AIME24, MATH, and Minerva Math.

Estimation Method	Positive Criterion	F1 Score
$P_t = MC(s_t, a_t)$ (hard label)	$P_t > 0$	73.2
$P_t = MC(s_t, a_t) - MC(s_t)$	$P_t \geq -0.1$	74.1
	$P_t \geq -0.3$	74.1
	$P_t \geq -0.5$	74.3
$P_t = \frac{MC(s_t, a_t)}{MC(s_t)}$	$P_t \geq 0.1$	73.5
	$P_t \geq 0.5$	73.5
	$P_t \geq 0.8$	**75.2**
	$P_t \geq 1.0$	72.3

Table 2: Results of GenPRM with different label estimation method and threshold on ProcessBench, reported with Pass@1. The best results are shown in **bold**.

and its variant, we use different thresholds ϵ and set the labels as correct by checking whether $P_t \geq \epsilon$. The results in Table 2 show that RPE and its variant outperforms hard label estimation and RPE with $\epsilon = 0.8$ achieves the best result. Full results are shown in Appendix B (Zhao et al. 2025).

Reasoning Components. To understand how each reasoning component influence GenPRM, we conduct experiments by training GenPRM with: (1) CoT data only, (2) code verification data only, and (3) full data. During inference phase, we could explicitly control different inference modules by special tag. For example, GenPRM trained with full data can be used to verify each step with CoT only by stopping generation at `</analyze>` token. The results in Table 3 show that: (1) the improvement of GenPRM mainly comes from CoT reasoning; (2) generating code and reasoning with code execution result improves the performance as well.

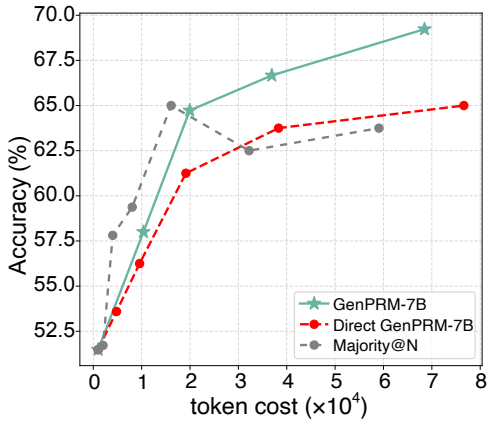


Figure 5: Accuracy as a function of token cost on the AMC23 dataset. The token cost is modulated by varying the number of samples (N) in Best-of-N.

Training		Inference			F1 Score
CoT	Code	CoT	Code	Code Exec.	
-	-	-	-	-	60.0
-	✓	-	✓	-	64.2
-	✓	-	✓	✓	69.6
✓	-	✓	-	-	78.8
		-	✓	-	61.5
		-	✓	✓	66.5
✓	✓	✓	-	-	79.3
		✓	✓	-	79.9
		✓	✓	✓	**80.5**

Table 3: Results on ProcessBench of `GenPRM` with different reasoning components, reported with Maj@8. The best results are shown in **bold**.

6 Conclusion

In this work, we propose GenPRM, a generative process reward model that performs explicit reasoning and code verification for process supervision. Experimental results on ProcessBench and several mathematical datasets show GenPRM outperforms prior PRMs. We also demonstrate that the performance of GenPRM increases via test-time scaling and GenPRM is effective as a critic model. We believe that this work provides perspectives on PRMs by demonstrating the strong TTS abilities of PRMs and extending the applications.

7 Limitations

Although GenPRM focuses mainly on mathematical reasoning tasks, it is worth to explore how to apply generative reasoning on coding and general reasoning tasks in the future (Zhang et al. 2025b). Additionally, it would be interesting to leverage RL to incentivize the generative reasoning abilities of GenPRM.

Acknowledgements

This work was supported by the STI 2030-Major Projects under Grant 2021ZD0201404 and the Shanghai Municipal Science and Technology Major Project.

References

AI-MO. 2024a. AIME 2024. https://huggingface.co/datasets/AI-MO/aimo-validation-aime. Accessed: 2025-12-22.

AI-MO. 2024b. AMC 2023. https://huggingface.co/datasets/AI-MO/aimo-validation-amc. Accessed: 2025-12-22.

Ankner, Z.; Paul, M.; Cui, B.; Chang, J. D.; and Ammanabrolu, P. 2024. Critique-out-Loud Reward Models. *arXiv preprint arXiv:2408.11791*.

Anthropic. 2023. Introducing Claude. https://www.anthropic.com/index/introducing-claude/. Accessed: 2025-12-22.

Beeching, E.; Tunstall, L.; and Rush, S. 2024. Scaling Test-Time Compute with Open Models.

Brown, B.; Juravsky, J.; Ehrlich, R.; Clark, R.; Le, Q. V.; Ré, C.; and Mirhoseini, A. 2024. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. *arXiv preprint arXiv:2407.21787*.

Choudhury, S. 2025. Process Reward Models for LLM Agents: Practical Framework and Directions. *arXiv preprint arXiv:2502.10325*.

Cui, G.; Yuan, L.; Wang, Z.; Wang, H.; Li, W.; He, B.; Fan, Y.; Yu, T.; Xu, Q.; Chen, W.; et al. 2025. Process Reinforcement through Implicit Rewards. *arXiv preprint arXiv:2502.01456*.

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Gemma Team; and Google DeepMind. 2025. Introducing Gemma 3: The most capable model you can run on a single GPU or TPU. https://blog.google/technology/developers/gemma-3. Accessed: 2025-12-22.

Gou, Z.; Shao, Z.; Gong, Y.; yelong shen; Yang, Y.; Duan, N.; and Chen, W. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *International Conference on Learning Representations (ICLR)*.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Jiang, S.; Liao, Y.; Chen, Z.; Zhang, Y.; Wang, Y.; and Wang, Y. 2025. MedS^3: Towards Medical Small Language Models with Self-Evolved Slow Thinking. *arXiv preprint arXiv:2501.12051*.

Lewkowycz, A.; Andreassen, A.; Dohan, D.; Dyer, E.; Michalewski, H.; Ramasesh, V.; Slone, A.; Anil, C.; Schlag, I.; Gutman-Solo, T.; Wu, Y.; Neyshabur, B.; Gur-Ari, G.; and Misra, V. 2022. Solving Quantitative Reasoning Problems with Language Models. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 3843–3857. Curran Associates, Inc.

Li, W.; and Li, Y. 2025. Process Reward Model with Q-value Rankings. In *International Conference on Learning Representations (ICLR)*.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2024. Let's Verify Step by Step. In *International Conference on Learning Representations (ICLR)*.

Liu, R.; Gao, J.; Zhao, J.; Zhang, K.; Li, X.; Qi, B.; Ouyang, W.; and Zhou, B. 2025a. Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling. *arXiv preprint arXiv:2502.06703*.

Liu, R.; Wang, J.; Shi, Y.; Xie, Z.; An, C.; Zhang, K.; Zhao, J.; Gu, X.; Lin, L.; Hu, W.; Li, X.; Zhang, F.; Zhou, G.; and Gai, K. 2025b. Attention as a Compass: Efficient Exploration for Process-Supervised RL in Reasoning Models. *arXiv preprint arXiv:2509.26628*.

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*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 46534–46594. Curran Associates, Inc.

Mahan, D.; Van Phung, D.; Rafailov, R.; Blagden, C.; Lile, N.; Castricato, L.; Fränken, J.-P.; Finn, C.; and Albalak, A. 2024. Generative Reward Models. *arXiv preprint arXiv:2410.12832*.

OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2024a. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms. Accessed: 2025-12-22.

OpenAI. 2024b. OpenAI o3-mini. https://openai.com/index/openai-o3-mini. Accessed: 2025-12-22.

Qwen Team. 2025. QwQ-32B: Embracing the Power of Reinforcement Learning. https://qwenlm.github.io/blog/qwq-32b. Accessed: 2025-12-22.

Setlur, A.; Nagpal, C.; Fisch, A.; Geng, X.; Eisenstein, J.; Agarwal, R.; Agarwal, A.; Berant, J.; and Kumar, A. 2025. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In *International Conference on Learning Representations (ICLR)*.

Skywork o1 Team. 2024. Skywork-o1 Open Series. https://huggingface.co/Skywork. Accessed: 2025-12-22.

Snell, C. V.; Lee, J.; Xu, K.; and Kumar, A. 2025. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations (ICLR)*.

Song, M.; Su, Z.; Qu, X.; Zhou, J.; and Cheng, Y. 2025. PRMBench: A Fine-grained and Challenging Benchmark for Process-Level Reward Models. *arXiv preprint arXiv:2501.03124*.

Sun, S.; Liu, R.; Lyu, J.; Yang, J.-W.; Zhang, L.; and Li, X. 2025. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *Knowledge-Based Systems*, 326: 114065.

Tan, X.; Yao, T.; Qu, C.; Li, B.; Yang, M.; Lu, D.; Wang, H.; Qiu, X.; Chu, W.; Xu, Y.; et al. 2025. AURORA: Automated Training Framework of Universal Process Reward Models via Ensemble Prompting and Reverse Verification. *arXiv preprint arXiv:2502.11520*.

Uesato, J.; Kushman, N.; Kumar, R.; Song, F.; Siegel, N.; Wang, L.; Creswell, A.; Irving, G.; and Higgins, I. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Wang, J.; Fang, M.; Wan, Z.; Wen, M.; Zhu, J.; Liu, A.; Gong, Z.; Song, Y.; Chen, L.; Ni, L. M.; et al. 2024a. OpenR: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*.

Wang, J.; Liu, R.; Lin, L.; Hu, W.; Li, X.; Zhang, F.; Zhou, G.; and Gai, K. 2025a. ASPO: Asymmetric Importance Sampling Policy Optimization. *arXiv preprint arXiv:2510.06062*.

Wang, J.; Liu, R.; Zhang, F.; Li, X.; and Zhou, G. 2025b. Stabilizing Knowledge, Promoting Reasoning: Dual-Token Constraints for RLVR. *arXiv preprint arXiv:2507.15778*.

Wang, P.; Li, L.; Shao, Z.; Xu, R.; Dai, D.; Li, Y.; Chen, D.; Wu, Y.; and Sui, Z. 2024b. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9426–9439.

Wang, W.; Gao, Z.; Chen, L.; Zhe, C.; Zhu, J.; Zhao, X.; Liu, Y.; Cao, Y.; Ye, S.; Zhu, X.; Lu, L.; Duan, H.; Qiao, Y.; Dai, J.; and Wang, W. 2025c. VisualPRM: An Effective Process Reward Model for Multimodal Reasoning. *arXiv preprint arXiv:2503.10291*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations (ICLR)*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems (NeurIPS)*, volume 35, 24824–24837.

Wu, Y.; Sun, Z.; Li, S.; Welleck, S.; and Yang, Y. 2025. Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for LLM Problem-Solving. In *International Conference on Learning Representations (ICLR)*.

Xie, Z.; Chen, L.; Mao, W.; Xu, J.; Kong, L.; et al. 2025. Teaching Language Models to Critique via Reinforcement Learning. *arXiv preprint arXiv:2502.03492*.

Xiong, W.; Zhang, H.; Jiang, N.; and Zhang, T. 2024. An Implementation of Generative PRM. https://github.com/RLHFlow/RLHF-Reward-Modeling.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024a. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Yang, A.; Zhang, B.; Hui, B.; Gao, B.; Yu, B.; Li, C.; Liu, D.; Tu, J.; Zhou, J.; Lin, J.; Lu, K.; Xue, M.; Lin, R.; Liu, T.; Ren, X.; and Zhang, Z. 2024b. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv preprint arXiv:2409.12122*.

Yang, R.; Ding, R.; Lin, Y.; Zhang, H.; and Zhang, T. 2024c. Regularizing Hidden States Enables Learning Generalizable Reward Model for LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yuan, L.; Li, W.; Chen, H.; Cui, G.; Ding, N.; Zhang, K.; Zhou, B.; Liu, Z.; and Peng, H. 2024. Free Process Rewards without Process Labels. *arXiv preprint arXiv:2412.01981*.

Zeng, S.; Tian, K.; Zhang, K.; Gao, J.; Liu, R.; Yang, S.; Li, J.; Long, X.; Ma, J.; Qi, B.; and Zhou, B. 2025a. ReviewRL: Towards Automated Scientific Review with RL. *arXiv preprint arXiv:2508.10308*.

Zeng, T.; Zhang, S.; Wu, S.; Classen, C.; Chae, D.; Ewer, E.; Lee, M.; Kim, H.; Kang, W.; Kunde, J.; et al. 2025b. VersaPRM: Multi-Domain Process Reward Model via Synthetic Reasoning Data. *arXiv preprint arXiv:2502.06737*.

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

Zhang, K.; Liu, R.; Zhu, X.; Tian, K.; Zeng, S.; Jia, G.; Fan, Y.; Lv, X.; Zuo, Y.; Jiang, C.; Liu, Z.; Wang, J.; Wang, Y.; Zhao, R.; Hua, E.; Wang, Y.; Wang, S.; Gao, J.; Long, X.; Sun, Y.; Ma, Z.; Cui, G.; Bai, L.; Ding, N.; Qi, B.; and Zhou, B. 2025a. MARTI: A Framework for Multi-Agent LLM Systems Reinforced Training and Inference.

Zhang, K.; Zhang, J.; Li, H.; Zhu, X.; Hua, E.; Lv, X.; Ding, N.; Qi, B.; and Zhou, B. 2025b. OpenPRM: Building Open-domain Process-based Reward Models with Preference Trees. In *International Conference on Learning Representations (ICLR)*.

Zhang, K.; Zuo, Y.; He, B.; Sun, Y.; Liu, R.; Jiang, C.; Fan, Y.; Tian, K.; Jia, G.; Li, P.; et al. 2025c. A Survey of Reinforcement Learning for Large Reasoning Models. *arXiv preprint arXiv:2509.08827*.

Zhang, L.; Hosseini, A.; Bansal, H.; Kazemi, M.; Kumar, A.; and Agarwal, R. 2025d. Generative Verifiers: Reward Modeling as Next-Token Prediction. In *International Conference on Learning Representations (ICLR)*.

Zhang, Y.; Wu, S.; Yang, Y.; Shu, J.; Xiao, J.; Kong, C.; and Sang, J. 2024b. o1-Coder: an o1 Replication for Coding. *arXiv preprint arXiv:2412.00154*.

Zhang, Z.; Zheng, C.; Wu, Y.; Zhang, B.; Lin, R.; Yu, B.; Liu, D.; Zhou, J.; and Lin, J. 2025e. The Lessons of Developing Process Reward Models in Mathematical Reasoning. *arXiv preprint arXiv:2501.07301*.

Zhao, J.; Liu, R.; Zhang, K.; Zhou, Z.; Gao, J.; Li, D.; Lyu, J.; Qian, Z.; Qi, B.; Li, X.; et al. 2025. GenPRM: Scaling Test-Time Compute of Process Reward Models via Generative Reasoning. *arXiv preprint arXiv:2504.00891*.

Zheng, C.; Zhang, Z.; Zhang, B.; Lin, R.; Lu, K.; Yu, B.; Liu, D.; Zhou, J.; and Lin, J. 2024. ProcessBench: Identifying Process Errors in Mathematical Reasoning. *arXiv preprint arXiv:2412.06559*.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 46595–46623. Curran Associates, Inc.

Zhong, J.; Li, Z.; Xu, Z.; Wen, X.; and Xu, Q. 2025. Dyve: Thinking Fast and Slow for Dynamic Process Verification. *arXiv preprint arXiv:2502.11157*.

Zhu, J.; Zheng, C.; Lin, J.; Du, K.; Wen, Y.; Yu, Y.; Wang, J.; and Zhang, W. 2025. Retrieval-Augmented Process Reward Model for Generalizable Mathematical Reasoning. *arXiv preprint arXiv:2502.14361*.

Zhu, X.; Qi, B.; Zhang, K.; Long, X.; Lin, Z.; and Zhou, B. 2024. PaD: Program-aided Distillation Can Teach Small Models Reasoning Better than Chain-of-thought Fine-tuning. In Duh, K.; Gomez, H.; and Bethard, S., eds., *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2571–2597. Mexico City, Mexico: Association for Computational Linguistics.