Formal-PRM: A Process Reward Model Based on Formalized Verification

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated formidable capabilities in solving mathematical problems, yet they may still make logical reasoning and computational errors during the problem-solving process. Using Process Reward Models (PRMs) to evaluate the correctness of each step in solutions generated by large models is an important approach. However, existing PRMs still suffer from some problems, such as a lack of generalization. Therefore, this paper proposes a novel framework, Formal-PRM, which includes a Formalizer and a Checker, to formally verify the correctness of solutions generated by large language models. We empirically investigate the effectiveness of Formal-PRM in two scenarios: 1) Verification: Formal-PRM is used to determine whether a solution to a given problem is correct. 2) Test-time scaling: When Formal-PRM identifies errors in a solution generated by an LLM-based solution generator, it provides corrective suggestions from the Checker to the generator to re-generate the solution. We evaluate our framework on the widely used PRM benchmark: ProcessBench, demonstrating the superiority of our approach over existing methods. Moreover, our method outperforms existing approaches in test-time scaling.

1 Introduction

Utilizing LLMs for mathematical reasoning is a research area of significant importance, and recent efforts have achieved remarkable progress (Guo et al., 2025; Chervonyi et al., 2025). However, even the most advanced LLMs are still prone to making errors when solving mathematical problems (Mirzadeh et al., 2024; Zhou et al., 2024; Sun et al., 2025), particularly when the process involves complex logical reasoning and calculations. An effective remedy is test-time scaling, which uses a verifier to verify the correctness of solutions generated by LLMs. Furthermore, such a verifier may also provide feedback on incorrect answers to help LLM-based solution generators produce correct responses.

Process Reward Models (PRMs) (Lightman et al., 2023; Khalifa et al., 2023; Wang et al., 2024a; Zhao et al., 2025; She et al., 2025), which can provide a reward for each individual reasoning step, become a research focus in test-time scaling. Compared with other reward models (e.g., Outcome Reward Models), PRMs exhibit better overall performance, stronger generalization, and a specific capability to identify and address procedural errors during reasoning (Wang et al., 2024a) .

In general, existing PRMs in this context can be divided into two main categories: Discriminative PRMs (Lightman et al., 2023; Wang et al., 2024a; Khalifa et al., 2023) and Generative PRMs (Zhao et al., 2025; She et al., 2025).

• **Discriminative PRMs.** Given a mathematical problem-solving process, A Discriminative PRM assigns a score (usually within the interval [0,1]) to each step of the solution. For a given step, the higher the PRM score is, the more likely the model considers this step correct. The advantage of Discriminative PRMs is that they can determine whether a step is correct using a single token, thus achieving a fast scoring speed.

• **Generative PRMs.** A Generative PRM provides an evaluation of the correctness of each step in the reasoning process in textual form. By introducing a Chain of Thought (CoT), the judgment of the correctness of each step by a Generative PRM is usually more accurate than that by a Discriminative PRM.

However, neither existing Discriminative PRMs nor existing Generative PRMs can well avoid the weakness of large language models in handling complex mathematical reasoning (Zheng et al., 2025). That is, their assessment of the correctness of each step in a solution turns out to be unreliable frequently. For example, recent stud-

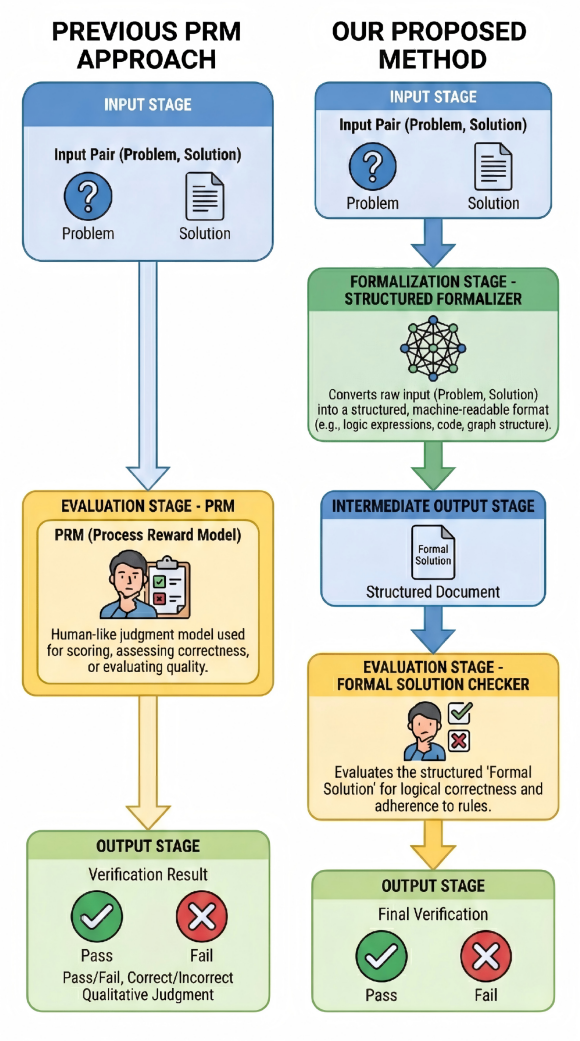

Figure 1: On the left side of the picture is the verification process of the previous PRM, and on the right side is our method.

ies (Song et al., 2025) have shown that PRMs often struggle to detect fine-grained errors in reasoning processes, and their performance is only slightly better than random guessing. In addition, existing PRMs generally suffer from a lack of robustness. For example, a model that performs well on simple problems in lower grades may perform poorly on difficult problems in higher grades. A potentially viable alternative to PRM is integrating large language models with interactive theorem provers(Ranta, 2004; Schaefer and Kohlhase, 2020; Pathak, 2024): the large language model generates a formal solution, and then an ITP is used to verify the correctness of the formal solution. However, this approach currently faces significant difficulties(Chen et al., 2025).Interactive theorem provers impose extremely stringent requirements on the formal correctness of proofs—even a non-canonical expression can cause verification to fail (despite the correctness of its expressed meaning). Additionally, completing a formal proof often demands a deep familiarity with the relevant knowledge. This sometimes forces general large models to generate thousands of formal proofs before one finally passes verification, making the process highly inefficient.

By analyzing the limitations of existing methods, we propose a novel approach: having a large language model convert natural-language solutions into formal-language proofs, and then utilizing the large model—instead of an interactive theorem prover (ITP)—to determine the correctness of the formalized answers.

As illustrated in Figure 2 : our approach offers the following advantages: 1) It is challenging to explicitly define fine-grained steps in natural language reasoning, whereas this is straightforward in formal language responses. 2) Due to the prevalence of leaps in training data, there are more or less gaps between premises and conclusions in certain steps of LLM-generated solutions. During formalization, the large language model fills in these reasoning gaps, thereby facilitating subsequent verification of the correctness of each step. 3) Employing an LLM instead of an ITP as a checker can fully exploit the model's comprehension capability, allowing verification to succeed even when the formal solution contains minor yet correctness-preserving non-canonical expressions.

In summary, we propose a novel framework, Formal-PRM, to verify solutions generated by LLMs. This method decomposes the verification of solution correctness into two stages. First, using LLM to converts a solution expressed by natural language into a solution expressed by formal language, limiting the rules for each derivation step to a finite set of patterns, thus obtaining standardized, fine-grained solutions. Then, useing another language model as a checker to verify the solution in the formal language. Our experiments show that our approach (i.e., Formal-PRM) clearly outperforms directly verifying natural language solutions with large language models.

Our main contributions:

1) We propose a formal language (named SimpleMath) based on the predicate logic. The significance of proposing this language is that the context constructed by SimpleMath closely resembles the extensive natural language-based mathematical texts that LLMs have learned during their

pre-training phase, thus making the formalization easier.

2) We propose a novel PRM via verifying solutions expressed in SimpleMath with the help of LLMs.

3) We empirically evaluate our approach on a widely used PRM benchmark, i.e., ProcessBench (Song et al., 2025), and our empirical results demonstrate the superiority of our approach over existing approaches.

4) We empirically investigate the application of our approach in Test-Time Scaling and our empirical results demonstrate that our approach outperforms existing approaches.

2 Methodology

As shown in Figure 3, we input the problem and an LLM-generated (partial) solution , in which there may be leaps between the premises and the conclusions, into the Solution Formalizer to obtain an equivalent formal (partial) solution. Then, we use the Checker to determine whether each conclusion is true under its premises.

2.1 Solution Formalizer

The task of the Solution Formalizer is to transform a solution expressed in natural language into one expressed in a formal language. Therefore, constructing such a Formalizer involves the following steps:

- Define a formal language for expressing solutions;
- Construct a dataset containing mathematical problems, natural language solutions, and corresponding formal language solutions for training a formalizer ;
- Fine-tune a large language model using this dataset.

SimpleMath Language.

Existing mainstream formal languages for proving mathematical theorems, such as Lean and Coq, are based on type theory. Such formal languages significantly differ from what is used in the mathematical community when expressing mathematical concepts. To make the formalization process easier, we first define a formal language called SimpleMath, which is closer to a natural mathematical language. We provide an example in Appendix A to illustrate the convenience of SimpleMath in expressing mathematical concepts.

[th!]

Our SimpleMath language is an extension of the language of predicate logic, achieved by introducing additional constants and syntactic sugars. For example :

$$
\begin{aligned}
&definition(f): \mathbb{N} \to \mathbb{N} \\
&f(n) := f(n-1) + f(n-2), if\ n \geq 3\ ; \\
&\quad | \ 1, if\ n = 2\ ; \\
&\quad | \ 1, if\ n = 1\ ;
\end{aligned}
$$

This definition in SimpleMath has roughly the same effect as the following formular in first order language:

$$
\begin{aligned}
&forall\ n, n \in \mathbb{N} \to (P_1(n) \wedge P_2(n)) \\
&where, \\
&P_1(n) : (n = 1 \vee n = 2) \to f(n) = 1 \\
&P_2(n) : (n \geq 3 \to f(n) = f(n-1) + f(n-2))
\end{aligned}
$$

Context. As illustrated in Figure 2, our context is of Fitch Style (Genesereth and Kao, 2022). In the context, statements can be categorized into these types:

• **Goals.** A goal represents a mathematical problem to be solved.

• **Facts.** A **Fact** refers to a known condition or piece of information within a problem that is accepted as true without requiring proof.

• **Assumptions.** An **Assumption** represents a mathematical proposition assumed to be true. In SimpleMath, the implication elimination rule is applied to assumptions to derive new propositions.

• **Theorems.** A **Theorem** refers to a statement that has been proven to be true based on previously established definitions, facts, and other theorems.

• **Definitions.** A **Definitions** refers to a precise statement that clearly explains the meaning of a mathematical concept.

• **Conclusions.** A **Conclusion** refers to a statement derived from known facts, definitions, theorems and previously established conclusions within a problem-solving process. In SimpleMath, every conclusion must explicitly state its premises and the logical inference rules used.

In the context, each statement contains global information: an assumption_list, which records under which assumptions the statement was derived. The information in the assumption_list is crucial,

PROBLEM STATEMENT

prove : $\forall x, y$ in $\mathbb{R}$, $x^2 + y^2 = 1$ implies $x + y <=$ sqrt(2) ;

NATURAL LANGUAGE PROOF

Given $x^2 + y^2 = 1$, we utilize trigonometric substitution. Let $x = \sin(t)$ and $y = \cos(t)$ for some real number t.
Then, the expression becomes $x + y = \sin(t) + \cos(t)$.
By the auxiliary angle formula, this is equal to $\sqrt{2} \cdot \sin(t + \pi/4)$.
Since the maximum value of the sine function is 1, the maximum value of the entire expression is $\sqrt{2} * 1 = \sqrt{2}$.
Therefore, it is proven that $x + y \leq$ sqrt(2).

FORMAL SOLUTION BASED ON SIMPLEMATH

[0]	definition $D = \{z \mid \exists x, y, x^{*2} + y^{*2} = 1$ and $z = x + y\}$	
[1]	goal $\forall z \in D, z <=$ sqrt(2)	
[2]	assumption $a \in D$	from(universal_introduction_setup)
[3]	$\exists x, y, x^{*2} + y^2 = 1$ and $a = x + y$	from(definition_expansion, [0], [2])
[4]	$x^{*2} + y^2 = 1$ and $a = x + y$	from(existential_elimination, [3])
[5]	$\exists t, x = \sin(t)$ and $y = \cos(t)$	from(trig_parametrization, [4])
[6]	$x = \sin(t)$ and $y = \cos(t)$	from(existential_elimination, [5])
[7]	$a = \sin(t) + \cos(t)$	from(substitution, [4], [6])
[8]	$\sin(t) + \cos(t) <=$ sqrt(2)	from(trig_inequality_theorem)
[9]	$a <=$ sqrt(2)	from(transitivity, [7], [8])
[10]	$\forall z \in D, z <=$ sqrt(2)	from(universal_introduction_conclusion, [2], [9])
[11]	Q.E.D.	from(goal_achievement, [1], [10])

Figure 2: Examle of different languages. The first solution is in natural language, while the second solution is a formal solution based on SimpleMath.

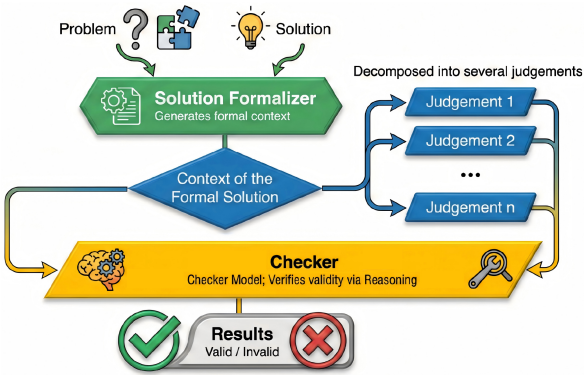

Figure 3: The overview framework of Formal-PRM including a Solution Formalizer and a Checker . First, the problem and solution are input into the solution formalizer, resulting in a context of the formal solution, which can be decomposed into several judgements. And then, we obtain the results through Checker which determines the validity of judgments by reasoning.

as it determines which logical inference rules are applicable at each step of the reasoning process.

Judgement. The format of a judgement is $P_1, P_2, \ldots, P_n \vdash Q$, where each P_i (for $i = 1, \ldots, n$) and Q are statements. It denotes that the proposition Q can be derived under the premises $P_1, \ldots, P_n$.

Solution Graphs

We use a Solution Graph to represent the relationships between different statements within a context, and each context can be transformed into its corresponding Solution Graph.

Given a context, its corresponding Solution Graph is defined as a tuple (V, E), where:

• V is the node set, where each element $v_i, i = 0, 1...$ corresponds to the the i-th statement within the context.

• E is the edge set, where $e_{0,n}, e_{1,n}, ..., e_{n-1,n} \in E$ if and only if $v_0, ..., v_n \in V$ and v_n is a direct conclusion of $v_0, v_1, ..., v_{n-1}$.

One of the most significant observations is that Solution Graphs are often sparse (as illustrated in Figure 4), implying that when a Checker model is verifying the correctness of a particular conclusion, it only needs to input the few statements that are relevant to that conclusion.

Decompose the formal solution

By utilizing the information in the Solution Graph, we can obtain a series of judgments of the following form:

$$P_{i_1}, \ldots, P_{i_k} \vdash Conclusion.$$

We use the example in Figure 2 to illustrate the process of obtaining the judgement from the formal solution. Statements [3] to [10] are conclusions. Therefore, we can derive 8 judgements from the formal solution. The first judgement is

$$Definition_0, Assumption_2 \vdash Conclusion_3 ,$$

where

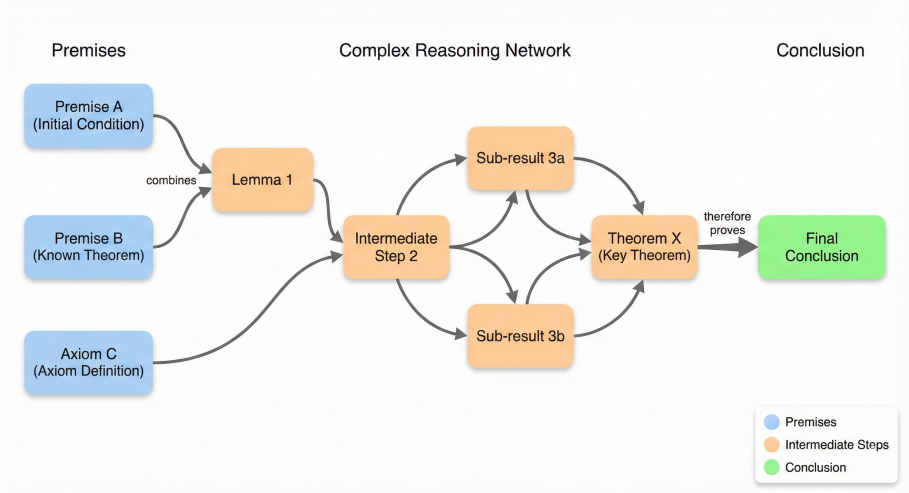

Figure 4: The figure is a SolutionGraph of formalsolution , where each statement is direct conclusion of some previous step's statements.

$$Definition_0:$$

$$D = \{x | \exists x, y \in \mathbf{R}, x^2 + y^2 = 1 \wedge z = x + y\};$$

$$Assumption_2:$$

$$a \in D;$$

$$Conclusion_3:$$

$$\exists x, y \in \mathbf{R}, x^2 + y^2 = 1 \wedge a = x + y.$$

This judgement can be abbreviated as $[0], [2] \vdash [3]$. Using the same method, we can derive 8 such judgements. Since the Solution Graph can represent dependency relationships between different statements, we can simplify judgments.

For example, In the example above, If we do not consider the information in the SolutionGraph that expressing the dependencies between different statements represented by the SolutionGraph, the judgement should be :

$$Definition_0, Goal_1, \\ Assumption_2 \vdash Conclusion_3.$$

rather than :

$$Definition_0, Assumption_2, \vdash Conclusion_3.$$

Leveraging the information from the SolutionGraph can often reduce the number of premises in the Judgement. The reason lies in the fact that, given a formal context P of length L (i.e., containing L statements) and a conclusion c, the number n of premises in P that are directly logically related to c is often much smaller than L. The reason lies in the fact that the number of premises in logical inference rules is typically less than 4.

Data Generation & Training

The data we use to train the Solution Formalizer is generated through the following pipeline:

1. **Initial Dataset Construction**: We collect 1,000 mathematical problems and engage human experts to write corresponding solutions in our formal language (i.e., SimpleMath). This yields an initial dataset

$$\mathcal{D}_{\text{initial}} = \{(\, problem_i \,, formal\_solution_i \,)\}.$$

2. **Initial Model Training**: Using dataset $\mathcal{D}_{initial}$, we perform supervised fine-tuning on `Qwen3-32B` to obtain a large language model $\phi_{initial}$ capable of generating formalized solutions.

3. **Formalization Data Expansion**: We select 12k mathematical problems drawn from mathematical competitions and standard textbooks. For each mathematical problem, we first use the large language model $\phi_{initial}$ to generate a formalized solution in SimpleMath. We then filter out those solutions that are incorrect and, through prompt engineering, instruct $\phi_{initial}$ to convert the formalized solutions into their natural-language counterparts, thereby constructing an expanded dataset $\mathcal{D}_{formalize}$:

$$\{(problem_i,\ solution_i,\ formal\_solution_i)\}.$$

The dataset $\mathcal{D}_{\text{formalize}}$ will be used for supervised fine-tuning of the large language model to obtain the formalizer: ϕ_{formal}.

2.2 Checker

After the formulator parses the problem P and the solution S into formal representations $\widehat{P}$ and $\widehat{S}$,we obtained a series of judgements: $\mathbb{T}_1 \vdash Q_1, ..., \mathbb{T}_i \vdash Q_i$. Here $\mathbb{T}_i$ is the context that includes all useful primises, and Q_i is the conclusion to verify. Our Checker model is prompted to evaluate the correctness of these judgments and, when a judgment is erroneous, to provide the reasons for the error. The formal solution derived by the Formalizer exhibits a finer granularity of reasoning compared to the solution, which significantly reduces the difficulty for the Checker in assessing the correctness of the judgement. The prompt for the Checker is available at the anonymous website https://anonymous.4open.science/r/Formal-PRM-7381.

We summarize the procedure for using F-PRM to evaluate the correctness of a solution as follows:

- Step 1: Use the formalizer to formalize the given solution, obtaining a formal solution. After obtaining the formal solution, derive the SolutionGraph from it.
- Step 2: Decompose the formal solution into multiple judgement of the form $T \vdash P$, and reduce T to a set of premises relevant to P, according to the solution graph.
- Step 3: Use the checker to verify the correctness of all judgments. If all judgments are correct, the solution is deemed correct.

3 Experiments

We conducted a series of experiments to compare Formal-PRM with existing PRMs on the following tasks: 1) Evaluate the correctness of each step in a solution. 2) Test-time scaling.

For complex problems, generating the correct reasoning may become a challenge for large language models (LLMs), and it often requires multiple attempts. There are two common scenarios for enhancing the capabilities of large models during the test-time:

- Best-of-N (Ichihara et al., 2025) : select the solution with the highest score among N candidate solutions using a PRM as the evaluator.
- Solution-Refine (Madaan et al., 2024): assess the correctness of each step in the generated solution with a PRM; if a step is judged to be incorrect, this information is passed to the generator LLM to produce a new solution. the solution.

3.1 Experimental Setup

Benchmarks:

- To assess the efficacy of the PRMs for process-level rewarding, we test it on a demanding benchmark: ProcessBench (Song et al., 2025).
- We evaluate the performance of PRM for test-time scaling on four datasets (i.e., MATH, AMC23, AIME24, and Minerva Math).

Baselines:

- Math-Shepherd (Wang et al., 2024a) : Automatically assigning step labels that represent the probability of reaching the correct solution, based on Monte Carlo Tree Search.
- RLHFlow-DeepSeek (Wang et al., 2024b): Similar to Math-Shepherd, but trained with iterative DPO.
- Qwen2.5-Math-PRMs (Zhang et al., 2025) : Employing LLM-as-Judge for consistency filtering yields a 1.8 M-sample training dataset that gives rise to two discriminative PRMs : Qwen2.5-Math-PRM-7B and Qwen2.5-Math-PRM-72B.
- Gen-PRM (Zhao et al., 2025):A generative process reward model that performs explicit Chain-of-Thought (CoT) reasoning before providing judgment for each reasoning step.

• For task two, we compare Formal-PRM to Self-Consistency (Wang et al.) , GenPRM(Zhao et al., 2025) and RLHFlow-DeepSeek-8B.
• For task three, we compare Formal-PRM with the GenPRM (Zhao et al., 2025) .

Implement details: Our base models are from DeepSeek-R1-Distill-Qwen-7B and the Qwen3 series (including Qwen3-8B, Qwen3-14B, and Qwen3-32B). We fine-tune the large language model using the LoRA algorithm with a learning rate of 3×10^{-4} and a batch size of 24. More information about the experiment is available on the anonymous website https://anonymous.4open.science/r/Formal-PRM-7381.

3.2 Main Results

Task one: Determining the correctness of a solution

We report the results of Formal-PRM and the baselines in Table 1. We have the following observations:

1) With the same parameter count, a generative PRM performs better than adiscriminative PRM, and both GenPRM and Formal-PRM significantly outperform the Qwen2.5-Math-PRM series.

2) Formal-PRM exhibits better robustness. The questions in ProcessBench are sourced from GSM8K, Math, OlympiadBench, and Omni-MATH. Among these, GSM8K and Math are relatively simple, while OlympiadBench and Omni-MATH are more difficult. As shown in table 1, The correctness of our Formal-PRM is more stable and does not show significant decline as the difficulty of the questions increases. We believe the reason that our Formal-PRM is more stable is that, during the formalization process, we break down the answers into finer-grained derivations. Therefore, the difficulty of judging each step in the problem-solving process does not increase with the overall difficulty of process rewarding.

Task two and three: Applying PRM to guide the test-time scaling of the Best-of-N(Ichihara et al., 2025) and Solution Refine(Madaan et al., 2024) Table 2 and Table 3 show that Formal-PRM outperforms the baselines on MATH, MC23,AIME24, and MinervaMath with Qwen2.5-Math-7B-Instruct and Qwen2.5-7B-Instruct as the generation model.

- Best-of-8 : We generate eight candidate solutions for each problem. For self-consistency, we select the answer with the highest consistency score among the candidate answers as the final answer. For Formal-PRM, we choose the solution in which every step is evaluated as correct as the final answer. If more than one solution is evaluated as correct, we select the one with the fewest number of statements. Table 2 presents that compared to baselines, the solutions identified by Formal-PRM are more likely to be correct.

- SolutionRefine : In the solution-refine experiment, we set the maximum number of refinements allowed per step to 3. Table 3 presents that compared with the baseline, Formal-PRM consistently performs better than GenPRM.

In both experiments where PRM is applied to test-time scaling, our method significantly outperforms the baselines. Both GenPRM-7B and Formal-PRM-7B are fine-tuned from the same base model. Using the same base model, our Formal-PRM achieves superior performance compared to the baseline. Moreover, the performance of Formal-PRM-8B, fine-tuned based on Qwen3-8b, is significantly higher than that of Formal-PRM-7B. This demonstrates that when the capability of the base model improves, the performance of the PRM obtained through our framework can also be significantly enhanced.

4 Related Work

4.1 Auto Formalization

There are two types of autoformalization approaches: rule-based approaches and LLM-based approaches. Rule-based approaches (Ranta, 2004; Schaefer and Kohlhase, 2020; Pathak, 2024) are deterministic and transparent, making them easier to debug and understand. However, rule-based approaches often struggle with the diversity and complexity of natural languages, leading to limitations in handling edge cases and generalizing to new problem descriptions.

LLM-based autoformalization leverages large language models (LLMs) to translate mathematical statements from natural languages into formal languages. (Wu et al., 2022) demonstrated that through few-shot learning (Wang et al., 2020; Parnami and Lee, 2022), LLMs can effectively translate informal mathematical statements into formal specifications in Isabelle/HOL, achieving an accuracy of 25.3%. Other works(Xin et al., 2024a,b) for transforming an informal solution into code that can be verified by interactive theorem provers achieve higher accuracy. However, these works require fine-tuning LLMs on datasets containing a large amount of formalized knowledge and introducing search algorithms, such as BFS and MCTS (Browne et al., 2012; Świechowski et al., 2023), in the formalization process.

4.2 Process Supervision

Process supervision is designed to evaluate and improve the reasoning capabilities of LLMs by focusing on the intermediate steps of the reasoning process, rather than just the final output. There are two types of Process Supervised Models: Discriminative PRMs. and Generative PRMs.

Model	GSM8k	MATH	OlympiadBench	OmniMATH	Avg.
RLHFlow-DeepSeek-8B	38.8	33.8	16.9	16.9	26.6
Math-Shepherd-7B	47.9	29.5	24.8	23.8	31.5
Qwen2.5-Math-PRM-7B	68.2	62.6	50.7	44.3	56.5
Qwen2.5-Math-PRM-72B	87.3	80.6	74.3	71.1	78.3
GenPRM-7B	81.0	80.3	72.2	69.8	75.8
GenPRM-32B	83.1	81.7	72.8	72.8	77.6
Formal-PRM-7B	80.2	81.4	78.9	76.5	79.2
Formal-PRM-8B	90.4	95.4	93.8	92.6	93.1
Formal-PRM-14B	96.6	97.9	94.7	94.0	95.8
Formal-PRM-32B	**97.5**	**98.6**	**96.3**	**96.7**	**97.3**

Table 1: Performance comparison on ProcessBench. The "Avg." column represents the average score across the four datasets. The best results in each category are highlighted in **bold**.

PRM Model	MATH	AMC23	AIME24	MinervaMATH
Pass@1	84.5	70.0	6.1	33.6
Cons@8	87.1	75.3	7.6	35.9
PRM800k	86.5	73.8	9.4	36.2
GenPRM-7B	87.9	75.2	10.5	36.7
Formal-PRM-7B	88.3	75.6	10.8	37.1
Formal-PRM-8B	**89.2**	**84.5**	**13.7**	**40.2**

Table 2: Test-Time Scaling: Performance comparison on the Best-of-8 strategy of the generation model **Qwen2.5-Math-7B-Instruct**. The best results are highlighted in **bold**.

Refine Model	MATH	AIME24	AMC23	Minerva
None	76.2	7.1	51.6	34.5
GenPRM-7B	82.9	9.3	62.7	34.9
Formal-PRM-7B	83.6	9.3	63.1	35.3
Formal-PRM-8B	**87.5**	**10.4**	**68.2**	**39.8**

Table 3: Test-Time Scaling performance on Solution-Refine that using **Qwen2.5-7B-Instruct** as generator. "None" indicates the baseline performance without refinement. The best results are highlighted in **bold**.

• **Discriminative PRMs.**A discriminative PRM assigns a score to each individual step in the reasoning process (Lightman et al., 2023). This approach offers notable advantages, particularly in speed and computational efficiency. By directly optimizing the relationship between input features and output labels, discriminative PRMs avoid the complexity of generating intermediate reasoning steps, enabling rapid inference. However, this efficiency comes at a cost. Poor generalization is a critical limitation of discriminative PRMs (Song et al., 2025). Since they prioritize learning task-specific patterns from training data, they often struggle with out-of-distribution scenarios or novel tasks that deviate from the training distribution.

• **Generative PRMs.** The core idea of generative PRM (Kamoi et al., 2024) is to use an LLM as "Checker" to evaluate the correctness of the reasoning process. A step-level generative PRM dataset MathChecker-76k was proposed to fine-tune a generative PRM model (Xi et al., 2024). Recent findings by (Zheng et al., 2024) show that while prompt methods can effectively enable Large Language Models (LLMs) to generate step-wise rewards, they fail to generalize to more challenging math problems beyond GSM8K and MATH, and perform unsatisfactorily compared to generative PRMs.

5 Conclusion

Previously, all generative and discriminative Process Reward Models (PRMs) were designed to assess the correctness of natural language solutions. In this paper, we introduced Formal-PRM, a novel framework designed to verify mathematical reasoning by decomposing the verification process into a Solution Formalizer and a Checker. By transforming natural language solutions into the structured SimpleMath language, our approach effectively bridges the gap between informal reasoning and formal verification.

We empirically evaluated Formal-PRM on the ProcessBench and multiple reasoning datasets (e.g. MATH, AIME24). Compared to previous works, the primary advantages of Formal-PRM on the ProcessBench benchmark are that our method maintains consistent performance across varying difficulty levels by filling reasoning gaps through formalization. The results of our experiments demonstrate that our approach significantly outperforms existing state-of-the-art methods, including both discriminative and generative PRMs. Similarly, in the test-time scaling task, our approach also demonstrates superior performance compared to the baselines.

Limitations

This paper has the following limitations:

1) Our experiments on test-time scaling do not incorporate tree search methods, such as MCTS. We plan to integrate tree search techniques in future work.

2) An important application of PRM is to provide reward signals for reinforcement learning training; however, this paper does not explore this direction. In future work, we will investigate using Formal-PRM for reinforcement learning—particularly by leveraging information from Formal-PRM to reward solutions that contain fewer and less redundant conclusions.

3) Formal-PRM is a type of generative PRM. Although it achieves higher accuracy compared to discriminative PRM, it also requires significantly more computational resources. In future research, we will explore building more efficient generative PRMs.

References

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Si Chen, Wensheng Yu, Guowei Dou, and Qimeng Zhang. 2025. A review on mechanical proving and formalization of mathematical theorems. *IEEE Access*.

Yuri Chervonyi, Trieu H Trinh, Miroslav Olšák, Xiaomeng Yang, Hoang Nguyen, Marcelo Menegali, Junehyuk Jung, Vikas Verma, Quoc V Le, and Thang Luong. 2025. Gold-medalist performance in solving olympiad geometry with alphageometry2. *arXiv preprint arXiv:2502.03544*.

Michael Genesereth and Eric Kao. 2022. *Introduction to logic*. Springer Nature.

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*.

Yuki Ichihara, Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, Kenshi Abe, Mitsuki Sakamoto, and Eiji Uchibe. 2025. Evaluation of best-of-n sampling strategies for language model alignment. *arXiv preprint arXiv:2502.12668*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. 2023. Grace: Discriminator-guided chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15299–15328.

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

Archit Parnami and Minwoo Lee. 2022. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*.

Shashank Pathak. 2024. Gflean: An autoformalisation framework for lean via gf. *arXiv preprint arXiv:2404.01234*.

Aarne Ranta. 2004. Grammatical framework. *Journal of Functional Programming*, 14(2):145–189.

Jan Frederik Schaefer and Michael Kohlhase. 2020. Glif: A declarative framework for symbolic natural language understanding. In *FCR@ KI*, pages 4–11.

Shuaijie She, Junxiao Liu, Yifeng Liu, Jiajun Chen, Xin Huang, and Shujian Huang. 2025. R-prm: Reasoning-driven process reward modeling. *CoRR*.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*.

Yuhong Sun, Zhangyue Yin, Xuanjing Huang, Xipeng Qiu, and Hui Zhao. 2025. Error classification of large language models on math word problems: A dynamically adaptive framework. *arXiv preprint arXiv:2501.15581*.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562.

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, Bangkok, Thailand. Association for Computational Linguistics.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, et al. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision. *arXiv preprint arXiv:2411.16579*.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024a. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. 2024b. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*.

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*.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, et al. 2025. Genprm: Scaling test-time compute of process reward models via generative reasoning. *arXiv preprint arXiv:2504.00891*.

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*.

Congming Zheng, Jiachen Zhu, Zhuoying Ou, Yuxiang Chen, Kangning Zhang, Rong Shan, Zeyu Zheng, Mengyue Yang, Jianghao Lin, Yong Yu, et al. 2025. A survey of process reward models: From outcome signals to process supervisions for large language models. *arXiv preprint arXiv:2510.08049*.

Lexin Zhou, Wout Schellaert, Fernando Martínez-Plumed, Yael Moros-Daval, Cèsar Ferri, and José Hernández-Orallo. 2024. Larger and more instructable language models become less reliable. *Nature*, 634(8032):61–68.

A SimpleMath VS Lean

The complex type C differs from the real type R, but in type theory, every expression must have a unique type. Therefore, the mathematical statement

$$\oint_s \frac{e^z}{i\,(z^2+1)}\,dz \geq \frac{11}{2},$$

(Here, s denotes the circular contour in the complex plane with diameter 2) cannot be directly represented in Lean 4, even though it is mathematically correct.However, in SimpleMath, it can be expressed as:

$$integral(s, \frac{e^z}{i\,(z^2+1)}) \geq \frac{11}{2}.$$

Under the framework of type theory, the treatment of subsets differs significantly from classical set-theoretic approaches, whereas the majority of existing mathematical literature is based on set theory. Consequently, deriving a formal_solution in SimpleMath language from a natural language solution is comparatively easier.