Step-Wise Formal Verification for LLM-Based Mathematical Problem Solving

Kuo Zhou
Peking University , Beijing , China
zhoukuo@pku.edu.cn

Lu Zhang
Peking University , Beijing , China
zhanglu@sei.pku.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated formidable capabilities in solving mathematical problems, yet they may still commit logical reasoning and computational errors during the problem-solving process. Thus, this paper proposes a framework, MATH-VF, which includes a Formalizer and a Critic, for formally verifying the correctness of the solutions generated by large language models. Our framework first utilizes a Formalizer which employs an LLM to translate a natural language solution into a formal context. Afterward, our Critic (which integrates various external tools such as a Computer Algebra System and an SMT solver) evaluates the correctness of each statement within the formal context, and when a statement is incorrect, our Critic provides corrective feedback. We empirically investigate the effectiveness of MATH-VF in two scenarios: 1) Verification: MATH-VF is utilized to determine the correctness of a solution to a given problem. 2) Refinement: When MATH-VF identifies errors in the solution generated by an LLM-based solution generator for a given problem, it submits the corrective suggestions proposed by the Critic to the solution generator to regenerate the solution. We evaluate our framework on widely used mathematical benchmarks: MATH500 and ProcessBench, demonstrating the superiority of our approach over existing approaches.

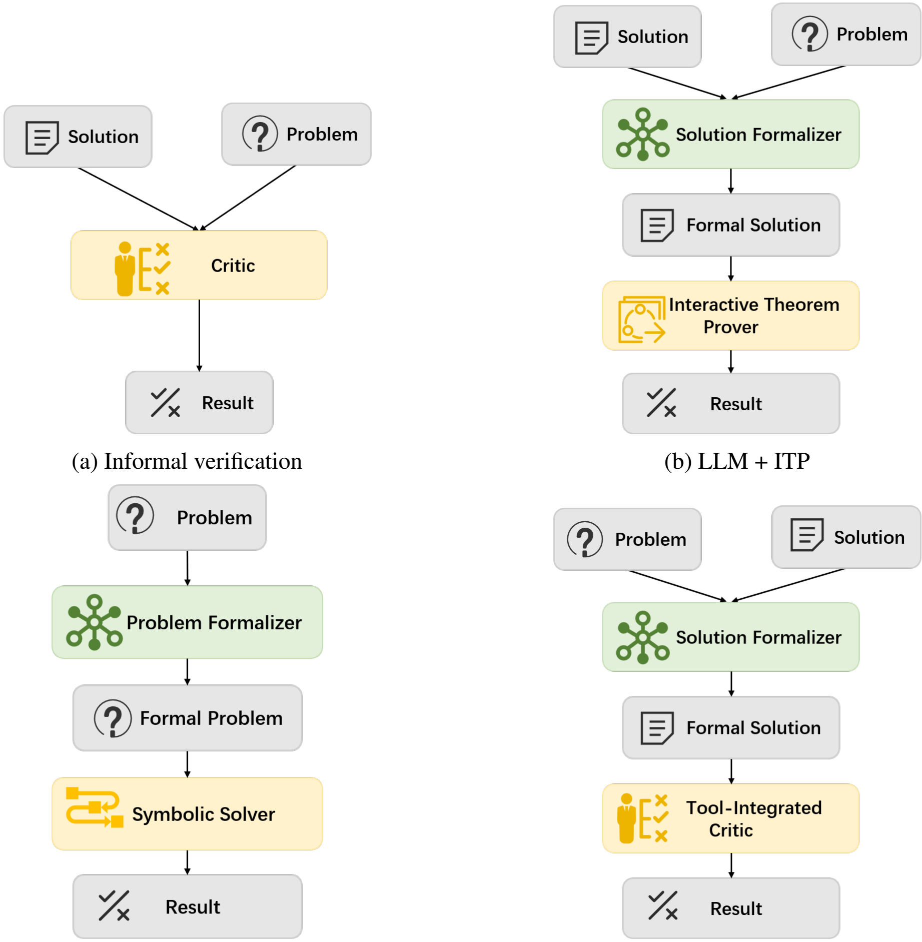
1 Introduction

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., 2024b; Sun et al., 2025), particularly when the process involves complex logical reasoning and calculations. Therefore, verifying the correctness of solutions generated by LLMs becomes a very important issue. In addition to verifying the correctness of the solutions, the verifier should also provide feedback on incorrect answers to help the LLM-based generator produce the correct responses.

In general, as shown in figure 1 (a) - (c), existing verification methods in this context can be divided into three main categories:

• **Informal verification for natural language reasoning.** Two types of models have been proposed to verify natural language reasoning: process reward models (PRMs) (Lightman et al., 2023; Wang et al., 2024; Khalifa et al., 2023) and Critic models (Luo et al., 2023; Khalifa et al., 2023). A PRM assigns a confidence score to each step of the reasoning process. In contrast, a Critic model provides an evaluation of the correctness of each step in the reasoning process in textual form. However, both PRMs and Critic models cannot avoid the weakness of large language models in handling complex mathematical calculations(Lin et al., 2024). Their assessment of the correctness of each step in the solution remains unreliable. For example, recent studies (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. Additionally, models, despite being more powerful than PRMs in some cases, still face challenges in accurately identifying errors, especially when dealing with complex reasoning tasks (Zheng et al., 2024).

• **Formal verification using interactive theorem provers (ITPs).** Approaches in this category typically convert solutions generated by LLMs into a formal language (e.g. Lean(Moura and Ullrich, 2021), Coq(Huet et al., 1997) ,Isabelle(Blanchette et al., 2011)) and then use interactive theorem provers to verify the formal solutions. Compared to approaches in the first category, approaches in this category are of stronger reliability. However, these approaches still face the following issues: 1) Many



(a) Informal verification

(b) LLM + ITP

(c) Formalizing problem and solving

(d) Our method: MATH-VF

Figure 1: Methods for verifying mathematical reasoning. (a) - (c) from previous work , (d) is our work

solutions expressed in unstructured natural language are difficult to formalize completely (Raza and Milic-Frayling, 2025). 2) Formal proofs generated by the interactive theorem provers can hardly be used as feedback to further guide the LLMs due to the limitations of formal languages in accessibility and usability (Liu et al., 2024b).

- **Autoformalizing the problem.** Recent approaches (Pan et al., 2023; Zhou et al., 2024a; Olausson et al., 2023; Ye et al., 2024) have focused only on formalizing the problem itself rather than solving it. These works use a symbolic solver to solve the formal problem, which brings about issues: 1) When the problem is outside the scope of what the symbolic solver can handle, the solving process is bound to fail. 2) Although these approaches can verify the accuracy of the final answer, they do not detect errors within the intermediate steps of the solving process.

To overcome the limitations of existing verification methods, we propose a novel and effective framework, named MATH-VF, for verifying solutions to mathematical problems. As illustrated in Figure 1(d), MATH-VF consists of two main components based on LLMs: the Formalizer and the Critic. The Formalizer is prompted to convert natural language solutions into SimpleMath — a formal language that we have specifically designed as an extension of classical first-order language.The Critic integrates external tools such as SymPy(Meurer et al., 2017) and Z3-solver(De Moura and Bjørner, 2008) to enhance its ability to verify the correctness of formal solutions. Figure 1 shows the differences between the existing verification methods and MATH-VF. In addition, we have observed a phenomenon concerning the relationships among intermediate steps in problem-solving; specifically, some intermediate steps are directly related, while others are not. As illustrated in Figure 2, Step 4 has direct associations with Step 2 and Step 3, but not with Step 1. Based on this observation, while maintaining the accuracy of judging steps, the complexity of the input information for our Critic has been significantly reduced.

Our main contributions:

1) We propose a formal language (named SimpleMath) based on first-order language, and develop a tool to formalize problem-solving processes expressed in informal languages. The significance

of proposing this language, is that the context constructed by first-order language closely resembles the extensive natural language-based mathematical texts that LLMs have learned during their pre-training phase. This makes formalization easier.

2) We first introduced a critic that integrates the Large Language Model with external tools, such as SymPy and Z3-Solver for determining the correctness of mathematical problem solving steps. Furthermore, we propose a new method to reduce the number of tokens that are entered into our Critic model.

3) Based on Formalizer and Critic, we develop MATH-VF, a training-free framework for step-by-step verification of mathematical reasoning. MATH-VF not only evaluates the correctness of solutions, but also provides constructive feedback for incorrect ones.

4) We evaluate our approach on two widely used mathematical benchmarks: MATH500(Lightman et al., 2023),ProcessBench(Song et al., 2025) ,and our empirical results demonstrate the superiority of our approach over existing approaches.

2 Related Work

2.1 Tool Augmented Language Models

Early efforts to integrate tools for mathematical reasoning mainly focus on leveraging external calculators, code interpreters(Toh et al., 2024), and symbolic solvers to address the limitations of traditional language models. For example, MathSensei (Das et al., 2024) incorporates a knowledge retriever (Bing Web Search), and an executor (Python), and a symbolic equation solver (Wolfram-Alpha API) to achieve improved accuracy on complex mathematical reasoning benchmarks. Similarly, the Multi-tool Integration Application framework combines Math Tool, Code Tool, and CoT Tool to perform basic calculations, generate executable code, and enhance logical coherence through iterative reasoning. Furthermore, a dataset of interactive tool-use trajectories is created, on which the performance of fine-tuned LLMs is significantly enhanced (Gou et al., 2023). In addition to these tools, recent research has also explored the integration of specialized solvers such as Z3 (De Moura and Bjørner, 2008) to handle complex mathematical constraints and symbolic reasoning (Pan et al., 2023). Z3 is a high-performance theorem prover that can efficiently solve a wide range of mathematical problems, including nonlinear polynomial constraints. By integrating Z3 with language models, the researchers aim to leverage its symbolic reasoning capabilities to improve the overall performance of mathematical reasoning tasks.

2.2 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; Azerbayev et al., 2023; **?**) 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. Compared with previous work, the advantage of MATH-VF lies in its ability to achieve high accuracy without fine-tuning.

2.3 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: Process Reward Models (PRMs) and Critic Models.

- **PRMs.** A PRM assigns a score to each individual step in the reasoning process. PRMs are particularly effective in identifying and correcting errors in multi-step mathematical reasoning (Zhang et al., 2025).
- **Critic Models.** The core idea of critic models(Kamoi et al., 2024) is to use an LLM as "Critic" to evaluate the correctness of the reasoning process

Problem 1: A and B are propositions. Prove that A → (B → A) is a tautology.

Solution:

[1] A -- Assumption

[2] B -- Assumption (within the context where A is true)

[3] A -- Directly derived by [1] (A is true in this context)

[4] B → A -- implies_intro ([2], [3]) (close the context where B was assumed)

[5] A → (B → A) -- implies_intro([1], [4]) (close the context where A was assumed)

Figure 2: Examle of Fitch-style proof

and provide feedback. A step-level Critic dataset MathCritic-76k was proposed to fine-tune a Critic 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 Critic each solution step by step, existing process reward models typically fail to generalize to more challenging math problems beyond GSM8K and MATH, and underperform compared to Critic models. However, the previous Critic models did not integrate external tools. Therefore, although they can judge the correctness of the solution, their accuracy is limited by the limitations of the computational and reasoning abilities of large models.

3 Methodology

As shown in Figure 3, we input the problem and its solution into the formalizer to obtain a context of formal solution. The premises and conclusions in the context may not be continuously derived, with gaps between them. Therefore, we use the Critic to determine whether each conclusion is true under its premises.

3.1 Formalizer

LLMs have demonstrated a notable ability to comprehend textual inputs and translate them into formal programs, such as mathematical equations or code. We take advantage of the few-shot generalization ability of LLMs to achieve this. By providing LLMs with detailed instructions about the grammar of the symbolic language and inference rules, together with a few examples in context, we observe that LLMs, such as Deepseek-V3(Liu et al., 2024a) and GPT-4(Achiam et al., 2023), can effectively follow the instructions to translate problems and solutions into a formal context, following our defined grammar and examples.

SimpleMath Language. Our SimpleMath language is an extension of the first-order language, achieved by introducing additional constants and syntactic sugars. For example :

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

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 five types:

• **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** refers to a statement that is accepted as true for the purpose of argument, investigation, or problem-solving, even though it may not be proven or verified.

• **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 term, concept or symbol.

• **Conclusions.** A **Conclusion** refers to a statement derived or inferred from known facts, defini-

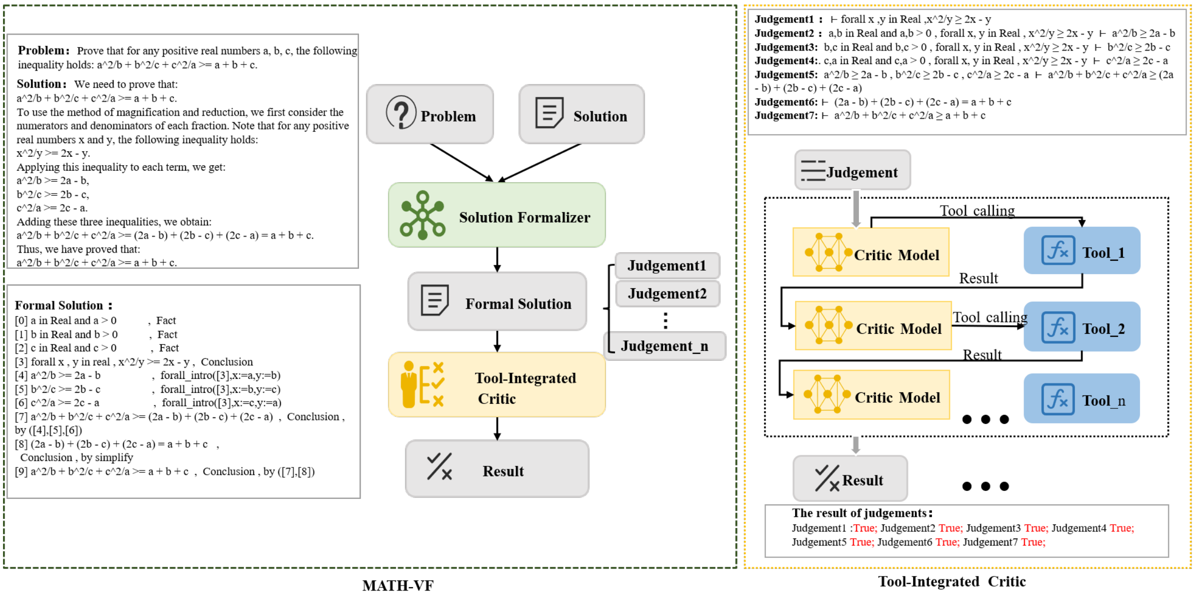


Figure 3: The overview framework of MATH-VF including a solution formalizer and a tool-integreated critic is on the left, and tool-integrated critic is depicted in detail on the right. First, the problem and solution are input into the n solution formalizer, resulting in a context of the formal solution, which can be decomposed into several judgements. And then, we obtain the results through tool-integereated critic which determines the validity of judgments by leveraging both reasoning and tool invocations.

Figure 4: On the left side of the figure is a dense Solution Graph, where each statement is direct conclusion of all previous step's statements. On the right side is a sparse graph, in the solution represented by this graph, statement4 and statement5 have only one premise.

tions, theorems and previously established conclusions within a problem-solving process.

Owing to the similarity between SimpleMath and natural language, the accuracy of the formalization results is extremely high. We tested this on the MATH500 dataset, and found that over 90% of the statements in the natural language solutions generated by the generator can be correctly formalized and verified(sec 4.2).

Solution Graph

We use Solution Graphs 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 findings is that Solution Graphs are often sparse (as illustrated in Figure 4), implying that when a Critic model is verifying the correctness of a particular conclusion, it only needs to input the few statements that are relevant to that conclusion.

Compared to existing methods, our approach effectively leverages the sparsity of the graph, which can significantly reduce the number of input tokens for verification, thereby greatly conserving computational resources. Recent work (Ling et al., 2024) also observes that LLM can verify each reasoning step by only using irrelevant primise. The differences between our method and that work are as follows: 1) Our approach not only extracts information relevant to the conclusion, but also utilizes the relationships among the related information. For example, given the Solution Graph as shown

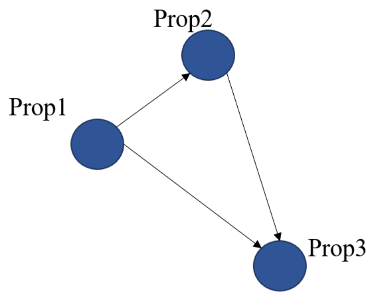


Figure 5: Prop3 is direct conlusion of Prop1 and Prop2 , Prop2 is direct conclusion of Prop2. Therefore we have judgement: Prop1 $\vdash$ Prop3.

in Figure 5, our method can validate a stronger conclusion

$$\text{Prop}_1 \rightarrow \text{Prop}_3,$$

compared to the conclusion

$$\text{Prop}_1 \wedge \text{Prop}_2 \rightarrow \text{Prop}_3.$$

In contrast, previous work(Ling et al., 2024), only considered which premises are relevant to the conclusion, without considering the relationships between premises. So ,the stronger conclusion cannot be validated. We verify the formal context using tools, whereas that work only verifies the context composed of natural language without employing any external tools.

3.2 Critic

After the formulator parses the problem P and the solution S into 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 Critic model is prompted to evaluate the correctness of these judgments and, when a judgment is erroneous, to provide the reasons for the error.

Our Critic model determines the validity of a judgment by leveraging both reasoning and tool invocations. As shown on the right of Figure 3, given a judgment, our Critic model invokes multiple tools during the reasoning process, working together to verify the judgment.

3.2.1 Tools for Our Critic Model

Computer Algebra System: SymPy is a powerful software tool designed to perform both symbolic and numerical computations. It is capable of manipulating mathematical expressions symbolically, performing operations such as differentiation, integration, solving equations, and factoring polynomials.

SMT Solver: Z3 is a high performance Satisfiability Modulo Theories (SMT) Solver developed by Microsoft Research. It is designed to check the satisfiability of logical expressions and generate models for satisfiable formulas. Z3 supports a wide range of theories, including linear arithmetic (both real and integer), bit vectors, arrays, datatypes, strings, and more.

3.2.2 Sparsity of the Solution Graph

As shown in Figure 4, different Solution Graphs lead to different total input lengths for the Critic to evaluate the solution. Given a formal context, there are n statements that need to be verified.Defining $C_1(n)$ as the total number of statements input to the Critic model in previous work, and $C_2(n)$ as the corresponding number in our work. We have:

$$C_1(n) = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}.$$

and:

$$C_2(n) \leq n \times M.$$

, where M is the maximum number of premises of all conclusions. According to our statistics, in almost all formal solutions, $M \leq 4$. Thus, in the vast majority of cases, the total number of input statements is less than or equal to $4 \times n$. Compared to other Critic models (Xi et al., 2024; Zheng et al., 2024), our approach can significantly reduce the number of tokens input into the LLM, especially when the context is long.

3.3 Prompt Methods for Critic

We use the few-shot learning approach for our Critic to call tools for reasoning. A small but important distinction from all previous work is that our Critic agent does not directly generate and execute code. Instead, it passes the problem to three LLM-based agents that are integrated with three specific tools. Our experience has shown that using a Critic model as both a reasoner and a tool-agent caller outperforms the use of a single Critic model without any tool-agent.

3.4 Solution Refinement

For complex problems, generating the correct reasoning may become a challenge for large language models (LLMs), and it often requires multiple attempts. Moreover, learning from errors is extremely important. Recent work (Madaan et al., 2024) proposes a method to improve LLM outputs through iterative feedback and refinement. It uses a single LLM to generate initial responses, provide feedback, and refine them without additional training data. Here, we employ a similar idea: If a solution fails validation, we pass both the solution and the reasons for its failure (provided by our Critic) to the Generator to regenerate the answer. This process repeats until the solution passes validation or reaches the maximum iteration limit.

Generator Model	Acc.	Method	Discrimin.
Deepseek - v2.5	80.2	LLM + Coq	3.9
		Primary Critic	90.6
		MATH-VF	**93.2**
Deepseek - v3	89.5	LLM + Coq	6.8
		Primary Critic	89.5
		MATH-VF	**95.7**
Qwen - 2.5 - 72B - Instruct	82.6	LLM + Coq	4.6
		Primary Critic	87.9
		MATH-VF	**92.4**
Qwen - 2.5 - 14B - Instruct	79.3	LLM + Coq	2.2
		Primary Critic	89.3
		MATH-VF	**94.3**

Table 1: "Acc". represents accuracy of Generator Model; "Discrimin." refers to the accuracy of verifier in the task of determining whether a reasoning path contains errors .

Metric	Qwen2.5-MATH-PRM-72B	MATH-VF
GSM8K	**87.3**	77.2
MATH	**80.6**	73.4
OlympiadBench	74.3	**76.1**
Omni-MATH	**71.1**	69.5
Avg.	78.3	74.1
SD.	0.062	0.029

Table 2: F1 scores of training-required methods and MATH-VF on ProcessBench.

4 Experiments

We conducted a series of experiments to compare MATH-VF with existing approaches on these tasks: **1)** Determining the correctness of the solution; **2)** Identifying the correct solution from the candidates; **3)** Refining solutions.

4.1 Experimental Setup

Dataset: We evaluate MATH-VF on two benchmaks : MATH500(Hendrycks et al., 2021; Lightman et al., 2023) and ProcessBench(Song et al., 2025).

Baselines:

• For task one, our comparison method is as follows: 1) LLM with Coq: We use the LLM to convert the informal solution into a Coq - formatted context, and then use Coq to verify the correctness of the context. The LLM first formalizes the problem and solution into a theorem to be proved. Then, the LLM formalizes the solution. Finally, Coq is used to verify the formal solution. 2) Primary Critic: We use the LLM to directly evaluate the correctness of the informal solution step by step without tool calling (Zheng et al., 2024). 3) Qwen2.5-MATH-PRM-72B(Zhang et al., 2025).

• For task two, we compare MATH-VF to Self-consistency (Wang et al.) and Primary Critic.

• For task three, we compare MATH-VF with the self-refinement method (Madaan et al., 2024). In Self-Refinement, we do not need a Critic; we only require the generator to iteratively produce solutions and conduct self-evaluation.

For all tasks, we use DeepSeek-v3 as Formalizer and Critic.

4.2 Main Results

Task one: Determining the correctness of the solution We report the results of MATH-VF and the baselines in Table 1. We have the following observations: 1) Given a correct informal solution, generating a formal solution that can be verified by Coq is extremely challenging. In our task-one experiment, among all the correct informal solutions generated by LLM, however, less than 10% of their corresponding formal solutions can pass the Coq verification. This indicates that generating Coq code poses a significant challenge for LLMs, if we only use in-context learning.

2) For the solutions generated by different models, the success rate of MATH-VF in determining the correctness of the solutions is similar. This indicates that MATH-VF can be used to assess the correctness of the solutions generated by various models.

A formal proof in Coq must include all the details of the reasoning to pass the check of interactive theorem provers. However, in MATH-VF, gaps between the conclusion and the premises are allowed, making formalization in MATH-VF relatively easier. Moreover, such gaps are often easily fillable. Another key point is that, compared to Coq, which is based on dependent type theory,

Generator Model	Method	Acc.
Deepseek - v2.5	Self-Consistency	83.1
	Primary Critic	82.9
	MATH-VF	**83.6**
Deepseek - v3	Self-Consistency	89.9
	Primary Critic	90.3
	MATH-VF	**90.5**
Qwen - 2.5 - 72B - Instruct	Self-Consistency	**87.1**
	Primary Critic	84.5
	MATH-VF	86.4
Qwen - 2.5 - 14B - Instruct	Self-Consistency	83.7
	Primary Critic	87.6
	MATH-VF	**88.7**

Table 3: Performance of MATH-VF , Self-Constency ,and Primary Critic. "Acc." represents the proportion of the identified solutions that are correct.

SimpleMath is closer to the informal mathematical language that LLMs have learned during their pre-training phase, and this makes the formalization process more straightforward.

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 2, our approach has a lower average F1 score than training-required method : Qwen2.5-MATH-PRM-72B. However The correctness of our approach is more stable and does not show significant decay as the difficulty of the questions increases. We believe the reason that our method 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 the problem. Qwen-PRM-72B requires training on a large amount of data, while our method is training-free. This means that the application scenarios of these two methods are not entirely the same. For example, in our method, we can use closed-source models as formalizers and critics. However, it is infeasible to use data to train closed-source models to serve as PRMs.

Task two: identifying the correct solution from the candidates. 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 MATH-VF, 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. The primary Critic follows the same steps as MATH-VF. Table 3 presents that compared to the other two methods, the solutions identified by MATH-VF are more likely to be correct. When the generator is relatively weak, the identification process leads to more significant improvements.

Generator Model	Method	Acc.
Deepseek - v2.5	Self-Refine	80.9
	MATH-VF	**83.5**
Deepseek - v3	Self-Refine	90.1
	MATH-VF	**92.4**
Qwen - 2.5 - 72B - Instruct	Self-Refine	83.7
	MATH-VF	**86.2**
Qwen - 2.5 - 14B - Instruct	Self-Refine	80.1
	MATH-VF	**82.6**

Table 4: Performance of MATH-VF and Self-Refine.

Task three: Refine solutions. For task three, we use two methods for refinement. As shown in Table 4: Self-Refinement contributes very little to the improvement of accuracy. The main reason MATH-VF outperforms Self-Refinement is that MATH-VF utilizes external tools to offer more accurate suggestions for improvement. Although in the Self-Refinement approach, LLMs evaluate solutions after generation to improve accuracy, it still struggles to overcome the inherent limitations of LLMs.

5 Conclusion

In this work, we propose a novel step-by-step approach for verifying solutions of mathematical problems. In our approach: MATH-VF, the Formalizer first formalizes the informal solution, and then the Critic leverages external tools such as Z3 and SymPy to verify the conclusions in each step of the reasoning. We evaluated MATH-VF on the MATH500 and ProcessBench, showed that :1) compared to existing training-free methods, MATH-VF performs better in verification, identification, and refinement. 2) compared to existing training-required methods, Math-VF shows inferior accuracy, yet demonstrates greater stability. Compared to previous work, the main advantages of MATH-VF are: 1) unlike PRM, our method is training-free. This makes our approach more compatible with closed-source models. 2) our approach reducing the number of premises input into the LLM during the verification process. 3) our approach exhibits stronger stability when dealing with problems of varying difficulty.

Limitations

Our paper has some limitations, which we leave for future work:

First, when Formalizer converts natural language into formal language, the expressions may contain syntax errors, leading to the failure of Critic when calling the solver. Future works could explore the development of using a syntax parser to parse the expressions.

Second, MATH-VF can only assess the correctness of each step in a solution without analyzing the effectiveness of each step (i.e., whether this step brings us closer to the goal). Future research could explore the study of the effectiveness of each step, thereby enhancing MATH-VF's performance in the solution refinement task.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*.

Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. 2011. Automatic proof and disproof in isabelle/hol. In *Frontiers of Combining Systems: 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings 8*, pages 12–27. Springer.

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.

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*.

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. Mathsensei: A tool-augmented large language model for mathematical reasoning. *arXiv preprint arXiv:2402.17231*.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

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

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

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*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. 1997. The coq proof assistant a tutorial. *Rapport Technique*, 178.

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*.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. *arXiv preprint arXiv:2402.14809*.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. 2024. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 36.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. 2024b. Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*.

Liangchen Luo, Zi Lin, Yinxiao Liu, Lei Shu, Yun Zhu, Jingbo Shang, and Lei Meng. 2023. Critique ability of large language models. *arXiv preprint arXiv:2310.04815*.

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.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

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*.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 625–635. Springer.

Theo X Olausson, Alex Gu, Ben Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger P Levy. 2023. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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.

Mohammad Raza and Natasa Milic-Frayling. 2025. Instantiation-based formalization of logical reasoning tasks using language models and logical solvers. *arXiv preprint arXiv:2501.16961*.

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

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.

Vernon Y. H. Toh, Deepanway Ghosal, and Soujanya Poria. 2024. Not all votes count! programs as verifiers improve self-consistency of language models for math reasoning. *Preprint*, arXiv:2410.12608.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 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)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

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*.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2024. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36.

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*.

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*.

Jin Peng Zhou, Charles E Staats, Wenda Li, Christian Szegedy, Kilian Q Weinberger, and Yuhuai Wu. 2024a. Don't trust: Verify–grounding llm quantitative reasoning with autoformalization. In *The Twelfth International Conference on Learning Representations*.

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

A Example of Critic Workflow

Input context:

[1] forall x , x in Real -> f(x) = (3^x)*(x^2) - (3^x)*y - (3^x) + x^2 - y - 1 , fact ;

conclusion to verify:

(f(x) < 0) → (x^2 < y) ;

<begin_criticize>

Given information f(x) = (3^x+1)*(x^2 - y + 1), verify (f(x) = 0) → (x^2 < y).

Maybe we should factor f(x) by symbolic calculater.

call_symbolic_calculater:

factor((3^x)*(x^2) - (3^x)*y - (3^x) + x^2 - y - 1)

end_call_symbolic_calculater

symbolic_calculater_result:

(3^x)*(x^2) - (3^x)*y - (3^x) + x^2 - y - 1 = (3^x+1)*(x^2 - y + 1)

end_symbolic_calculater_result

forall x in real, 3^x + 1 > 0 , so , x^2 - y + 1 < 0 if and only if f(x) < 0,

we can use Z3 to verify (x^2 - y + 1 < 0) → (x^2 < y) ,which is equal to : x^2 - y + 1 < 0 ∧ x^2 ≥ y is unsat .

call_logic_solver:

x,y is Real number, x^2 - y + 1 < 0 ∧ x^2 ≥ y is satisfied ?

end_call_logic_solver

logic_solver_result:

unsat

end_logic_solver_result

The conclusion (f(x) < 0) → (x^2 < y) is true

final result:

True

<end_criticize>

Figure 6: The example of critic workflow. In this example, we input context and the conclusion to verify, and then our Critic first call symbolic calculater and then call logic solver, subsequently concluding that this judgment is true.