DeepSeekMath-V2: Towards Self-Verifiable Mathematical Reasoning

Zhihong Shao*, Yuxiang Luo*, Chengda Lu*†, Z.Z. Ren*

Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, Xiaokang Zhang

DeepSeek-AI

zhihongshao@deepseek.com

`https://github.com/deepseek-ai/DeepSeek-Math-V2`

Abstract

Large language models have made significant progress in mathematical reasoning, which serves as an important testbed for AI and could impact scientific research if further advanced. By scaling reasoning with reinforcement learning that rewards correct final answers, LLMs have improved from poor performance to saturating quantitative reasoning competitions like AIME and HMMT in one year. However, this approach faces fundamental limitations. Pursuing higher final answer accuracy doesn't address a key issue: correct answers don't guarantee correct reasoning. Moreover, many mathematical tasks like theorem proving require rigorous step-by-step derivation rather than numerical answers, making final answer rewards inapplicable. To push the limits of deep reasoning, we believe it is necessary to verify the comprehensiveness and rigor of mathematical reasoning. Self-verification is particularly important for scaling test-time compute, especially for open problems without known solutions. Towards self-verifiable mathematical reasoning, we investigate how to train an accurate and faithful LLM-based verifier for theorem proving. We then train a proof generator using the verifier as the reward model, and incentivize the generator to identify and resolve as many issues as possible in their own proofs before finalizing them. To maintain the generation-verification gap as the generator becomes stronger, we propose to scale verification compute to automatically label new hard-to-verify proofs, creating training data to further improve the verifier. Our resulting model, DeepSeekMath-V2, demonstrates strong theorem-proving capabilities, achieving gold-level scores on IMO 2025 and CMO 2024 and a near-perfect 118/120 on Putnam 2024 with scaled test-time compute. While much work remains, these results suggest that self-verifiable mathematical reasoning is a feasible research direction that may help develop more capable mathematical AI systems.

1. Introduction

The conventional approach to reinforcement learning (RL) for mathematical reasoning involves rewarding large language models (LLMs) based on whether their predicted final answers to quantitative reasoning problems match ground-truth answers (Guo et al., 2025). This methodology suffices to allow frontier LLMs to saturate mathematical competitions that primarily evaluate final answers, such as AIME and HMMT. However, this reward mechanism has two fundamental limitations. First, it serves as an unreliable proxy for reasoning correctness – a model can arrive at the correct answer through flawed logic or fortunate errors. Second, it is

*Core contributors †Work done during internship at DeepSeek-AI.

inapplicable to theorem proving tasks, where problems may not require producing numerical final answers and rigorous derivation is the primary objective.

Consequently, LLMs trained on quantitative reasoning problems with such final answer reward still frequently produce mathematically invalid or logically inconsistent natural-language proofs. Moreover, this training approach does not naturally develop the models' ability to verify proof validity – they exhibit high false-positive rates, often claiming incorrect proofs are valid even when they contain obvious logical flaws.

The lack of a generation-verification gap in natural-language theorem proving hinders further improvement. To address this, we propose developing proof verification capabilities in LLMs. Our approach is motivated by several key observations:

- Humans can identify issues in proofs even without reference solutions – a crucial ability when tackling open problems.
- A proof is more likely to be valid when no issues can be identified despite scaled verification efforts.
- The efforts required to identify valid issues can serve as a proxy for proof quality, which can be exploited to optimize proof generation.

We believe that LLMs can be trained to identify proof issues without reference solutions. Such a verifier would enable an iterative improvement cycle: (1) using verification feedback to optimize proof generation, (2) scaling verification compute to auto-label hard-to-verify new proofs, thereby creating the training data to improve the verifier itself, and (3) using this enhanced verifier to further optimize proof generation. Moreover, a reliable proof verifier enables us to teach proof generators to evaluate proofs as the verifier does. This allows a proof generator to iteratively refine its proofs until it can no longer identify or resolve any issues. In essence, we make the model explicitly aware of its reward function and enable it to maximize this reward through deliberate reasoning rather than blind trial-and-error.

Built on DeepSeek-V3.2-Exp-Base (DeepSeek-AI, 2025), we developed **DeepSeekMath-V2**, a large language model optimized for natural-language theorem proving that demonstrates self-verifiable mathematical reasoning. Our model can assess and iteratively improve its own proofs, achieving gold-level performance in premier high-school mathematics competitions including IMO 2025 and CMO 2024. On the Putnam 2024 undergraduate competition, it scored 118/120, exceeding the highest score of 90 [1] obtained by human participants.

2. Method

2.1. Proof Verification

2.1.1. Training a Verifier to Identify Issues and Score Proofs

We developed high-level rubrics $\mathcal{I}_v$ for proof evaluation (see Appendix A.2) with the goal of training a verifier to evaluate proofs according to these rubrics, mirroring mathematical experts' assessment process. Specifically, given a problem X and a proof Y, the verifier $\pi_\varphi(\cdot|X, Y, \mathcal{I}_v)$ is designed to produce a proof analysis that first summarizes identified issues (if any) and then assigns a score based on three levels: 1 for complete and rigorous proofs with all logical steps clearly justified; 0.5 for proofs with sound overall logic but minor errors or omitted details; and 0 for fundamentally flawed proofs containing fatal logical errors or critical gaps.

[1] `https://kskedlaya.org/putnam-archive/putnam2024stats.html`

Curating Cold Start RL Data We constructed our initial training data through the following process:

1. We crawled problems from Art of Problem Solving (AoPS) contests [2], prioritizing math olympiads, team selection tests, and post-2010 problems explicitly requiring proofs, totaling 17,503 problems. This problem set is denoted as $\mathcal{D}_p$.
2. We generated candidate proofs using a variant of DeepSeek-V3.2-Exp-Thinking. As this model was not optimized for theorem proving and tended to produce concise but error-prone outputs, we prompted it to iteratively refine its proofs over multiple rounds to improve comprehensiveness and rigor.
3. We randomly sampled proofs across diverse problem types (e.g., algebra and number theory) and had mathematical experts score each proof according to the evaluation rubrics described above.

This process yielded an initial RL dataset $\mathcal{D}_v = \{(X_i, Y_i, s_i)\}$, where each item consists of a problem X_i, a proof Y_i, and an overall proof score $s_i \in \{0, 0.5, 1\}$.

RL Objective. Building on a version of DeepSeek-V3.2-Exp-SFT which was supervised fine-tuned on reasoning data related to mathematics and code, we trained the model with reinforcement learning to produce proof analyses using two reward components:

- **Format reward** R_{format}: An indicator function that enforces the model to generate both a summary of identified issues and a proof score, by checking whether the final response contains the key phrase "Here is my evaluation of the solution:" as well as a score within `\boxed{}` following "Based on my evaluation, the final overall score should be:".
- **Score reward** R_{score}: Rewards based on proximity between predicted score s_i' and annotated score s_i:
$$R_{\text{score}}(s_i', s_i) = 1 - |s_i' - s_i| \tag{1}$$

The RL objective for training the verifier is:

$$\max_{\pi_\varphi} \mathbb{E}_{(X_i, Y_i, s_i) \sim \mathcal{D}_v, (V_i', s_i') \sim \pi_\varphi(\cdot | X_i, Y_i)} \left[R_{\text{format}}(V_i') \cdot R_{\text{score}}(s_i', s_i) \right] \tag{2}$$

where V_i' denotes the verifier's final response and s_i' is the proof score extracted from it.

2.1.2. Introducing Meta-Verification to Review Proof Analyses

The approach described in Section 2.1.1 trains proof verification through RL to align predicted proof scores with expert annotations, but provides no direct supervision on the identified issues themselves. This creates a critical vulnerability: when evaluating flawed proofs (where $s_i < 1$) during training, the verifier can receive full reward by predicting the correct scores while hallucinating non-existent issues, undermining its trustworthiness.

To address this problem, we introduce **meta-verification**: a secondary evaluation process that assesses whether issues identified by the verifier indeed exist and whether these issues logically justify the predicted proof score according to the evaluation rubrics $\mathcal{I}_v$. The complete meta-verification rubrics $\mathcal{I}_{mv}$ are detailed in Appendix A.3.

[2] `https://artofproblemsolving.com/community/c13_contest_collections`

We trained a dedicated meta-verifier using RL to perform this evaluation. By incorporating the meta-verifier's feedback into verifier training, we can improve the faithfulness of the verifier's issue identification.

Meta-Verifier Training Process

1. We obtained an initial verifier π_φ following Section 2.1.1.
2. Mathematical experts scored the quality of verifier responses according to $\mathcal{I}_{mv}$, creating dataset $\mathcal{D}_{mv} = \{(X_i, Y_i, V_i, ms_i)\}$, where V_i is the analysis of proof Y_i and $ms_i \in \{0, 0.5, 1\}$ is the expert-annotated quality score.
3. We trained a meta-verifier $\pi_\eta(\cdot|X, Y, V, \mathcal{I}_{mv})$ to analyze the verifier's proof analysis V. The meta-verifier produces a summary of issues found in the analysis itself, followed by a quality score measuring how accurate and justified the verifier's analysis is. The RL objective follows the same structure as the verifier training, with format and score rewards.

Using the trained meta-verifier π_η, we enhanced the verifier training by integrating meta-verification feedback into the reward function:

$$R_V = R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}} \tag{3}$$

where R_{meta} is the quality score from the meta-verifier.

We trained the enhanced verifier on both the verification dataset $\mathcal{D}_v$ and the meta-verification dataset $\mathcal{D}_{mv}$, using the same reward mechanism on $\mathcal{D}_{mv}$ as used for training the meta-verifier. The resulting model can perform both proof verification and meta-verification tasks.

On a validation split of $\mathcal{D}_v$, the average quality score of the verifier's proof analyses – as evaluated by the meta-verifier – improved from 0.85 to 0.96, while maintaining the same accuracy in proof score prediction.

2.2. Proof Generation

2.2.1. Training a Generator for Theorem Proving

With verifier π_φ serving as a generative reward model, we train a proof generator $\pi_\theta(\cdot|X)$ with the RL objective:

$$\max_{\pi_\theta} \mathbb{E}_{X_i \sim \mathcal{D}_p, Y_i \sim \pi_\theta(\cdot|X_i)} [R_Y] \tag{4}$$

where R_Y is the proof score produced by $\pi_\varphi(\cdot|X_i, Y_i, \mathcal{I}_v)$.

2.2.2. Enhancing Reasoning via Self-Verification

When a proof generator fails to produce a completely correct proof in one shot – common for challenging problems from competitions like IMO and CMO – iterative verification and refinement can improve results. This involves analyzing the proof with an external verifier and prompting the generator to address identified issues.

However, we observed a critical limitation: when prompted to both generate and analyze its own proof in one shot, the generator tends to claim correctness even when the external verifier easily identify flaws. In other words, while the generator can refine proofs based on external feedback, it fails to evaluate its own work with the same rigor as the dedicated verifier.

This observation motivated us to endow the proof generator with genuine verification capabilities. During training, we prompt the generator π_θ to produce a proof Y followed by a self-analysis Z that follows the same format and rubrics $\mathcal{I}_v$ as the verifier (see Appendix A.1). We denote the proof score predicted in the self-analysis as s'.

To ensure faithful self-evaluation, we use the verifier π_φ to assess both components: the proof Y receives score $R_Y = s$, and the self-analysis Z receives a meta-verification score $R_{\text{meta}}(Z) = ms$. The reward function combines these assessments:

$$R = R_{\text{format}}(Y, Z) \cdot (\alpha \cdot R_Y + \beta \cdot R_Z) \tag{5}$$

$$R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z) \tag{6}$$

where $R_{\text{format}}(Y, Z)$ verifies that both the proof and self-analysis follow the specified format, $R_{\text{score}}(s', s)$ rewards accurate self-assessment. We set $\alpha = 0.76$ and $\beta = 0.24$. This reward structure creates the following incentives:

- Faithful acknowledgment of errors is rewarded over false claims of correctness.
- The highest rewards come from producing correct proofs and accurately recognizing their rigor.
- A good strategy to obtain high rewards for the proof generator is to identify and resolve as many issues as possible before finalizing the response.

2.3. Synergy Between Proof Verification and Generation

The proof verifier and generator create a synergistic cycle: the verifier improves the generator, and as the generator improves, it produces new proofs that challenge the verifier's current capabilities. These challenging cases – where the verifier may fail to identify issues in a single attempt – become valuable training data for enhancing the verifier itself.

To retrain and improve the verifier, we need labeled correctness data for newly generated proofs. Manual annotation, while straightforward, becomes increasingly time-consuming as problems grow harder and errors become more subtle. To boost annotation efficiency, we generated multiple verifier analyses per proof to surface potential issues for human review.

From this AI-assisted annotation process, we recognized two facts that make it feasible to push the level of automation a step further:

1. Scaling verifier samples increases the probability of catching real issues in flawed proofs.
2. Reviewing the verifier's identified issues is exactly **meta-verification**, which is easier than identifying issues from scratch. Meta-verification is also more sample-efficient for LLMs to master.

Building on these observations, we developed the following automated labeling process:

1. For each proof, generate n independent verification analyses