ByteDance | Seed

Seed-Prover: Deep and Broad Reasoning for Automated Theorem Proving

ByteDance Seed AI4Math

Abstract

LLMs have demonstrated strong mathematical reasoning abilities by leveraging reinforcement learning with long chain-of-thought, yet they continue to struggle with theorem proving due to the lack of clear supervision signals when solely using natural language. Dedicated domain-specific languages like Lean provide clear supervision via formal verification of proofs, enabling effective training through reinforcement learning. In this work, we propose **Seed-Prover**, a lemma-style whole-proof reasoning model. Seed-Prover can iteratively refine its proof based on Lean feedback, proved lemmas, and self-summarization. To solve IMO-level contest problems, we design three test-time inference strategies that enable both deep and broad reasoning. Seed-Prover proves 78.1% of formalized past IMO problems, saturates MiniF2F, and achieves over 50% on PutnamBench, outperforming the previous state-of-the-art by a large margin. To address the lack of geometry support in Lean, we introduce a geometry reasoning engine **Seed-Geometry**, which outperforms previous formal geometry engines. We use these two systems to participate in IMO 2025 and fully prove 5 out of 6 problems. This work represents a significant advancement in automated mathematical reasoning, demonstrating the effectiveness of formal verification with long chain-of-thought reasoning.

Project Page: https://github.com/ByteDance-Seed/Seed-Prover

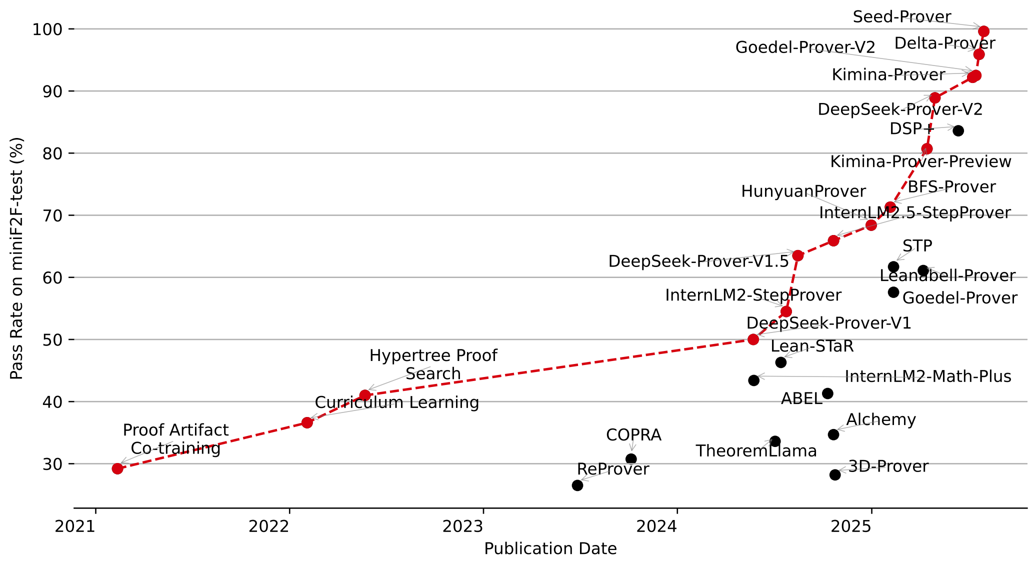

Figure 1 Growth in MiniF2F-Test performance over time.

1 Introduction

Recent advances in large language models (LLMs) have shown extending reasoning lengths through natural language can significantly boost performance on math benchmarks such as MATH and AIME [4, 11]. Training such models requires reinforcement learning (RL) on verifiable answers. It is extremely difficult to automatically, or even manually, verify a proof in natural language since each step must be carefully checked for correctness [13]. This poses significant challenges for applying reinforcement learning to the training of large language models to prove mathematical statements. Unlike natural language, formal languages such as Lean can provide a clear and automatic signal on the correctness of a formalized proof. A noteworthy work from AlphaProof [3] uses Lean to successfully solve 3 problems from the 2024 International Mathematical Olympaid (IMO). AlphaProof demonstrates that LLMs using formal language are capable of proving very challenging problems that LLMs using natural language fail to prove.

There are two types of LLM formal provers, step-level provers [3, 14, 22, 23, 25] and whole-proof generation provers[5, 24]. Step-level provers incrementally generate Lean code line-by-line. While this enables close interaction with the Lean environment, it requires special scaffolding to generate a complete Lean proof, and the interaction is often too granular to allow high-level reasoning. In contrast, whole-proof models generate an entire Lean proof at once, but typically lack interaction with the Lean compiler. Recent work has shown that combining whole-proof models with long chain-of-thought reasoning [9, 15, 21] substantially outperforms step-level provers. In this work, we propose **Seed-Prover**, a whole-proof model with following features:

- **Lemma-Style Proving**: Seed-Prover tries to generate useful intermediate lemmas before proving the main statement. These lemmas serve as shared knowledge across different inference paths.
- **Iterative Proof Refinement**: Seed-Prover iteratively refines its proof based on Lean compiler feedback, previous proved lemmas, and self-summarization.
- **Test-Time Scaling**: We implement a three-tiered inference strategy that enables Seed-Prover to think both deeply and broadly—allocating thinking budget to fine details while exploring interesting properties.
- **SOTA Performance**: Seed-Prover proves 5 out of 6 problems in IMO 2025, saturates MiniF2F [31] (shown in Figure 1), and outperforms prior work by up to 3× on multiple formal benchmarks.

Due to the lack of sufficient geometry support in Lean, Seed-Prover incorporates a dedicated geometry reasoning engine **Seed-Geometry**. Similar to existing line of efforts in AlphaGeometry [2, 19] and TongGeometry [30], Seed-Geometry follows the forward-chaining design in the reasoning engine implementation, where the system derives all known facts by checking applicable rules until closure is reached. By backward-tracing fact dependencies, Seed-Geometry identifies the minimum dependency relations in a geometry problem's configuration, seperating the problem context from the auxiliary constructions necessary to prove a problem. Using statistics derived from more than past 20 years of math olympiad competitions, Seed-Geometry performs extensive search in the geometry space defined by its dedicated domain-specific language and establishes a repository of 230 million unique geometry problems requiring auxiliaries. A Seed model trained on such dedicated geometry data becomes an exceptionally effective neuro-symbolic geometry prover, where it fills in the missing auxiliary geometry elements and the geometry reasoning engine performs step-by-step forward-chaining, completing the final proof of a problem. In experiments, Seed-Geometry solves 43 of the IMO-AG-50 (*vs.* 42 by AlphaGeometry 2), a benchmark that curates geometry problems of IMO from 2000 to 2024. It also sets a new state-of-the-art on the IMO shortlist geometry problems from 2000 to 2022, and notably solves the geometry problem of IMO 2025 under just 2 seconds.

2 Approach

Here we introduce the two systems we used in IMO 2025, Seed-Geometry and Seed-Prover.

2.1 Seed-Geometry

Seed-Geometry builds on the success of TongGeometry [30] and performs a major redesign. From a global perspective, both systems leverage trained neural models to complete missing auxiliary constructions and specialized reasoning engines to forward-chain derivation. However, Seed-Geometry is a substantial upgrade over TongGeometry in the following aspects.

2.1.1 Extended Domain-Specific Language

Seed-Geometry constructs geometric diagrams in the principle of ruler-and-compass construction. However, plain ruler-and-compass construction steps can be long and cumbersome, making the language representation of the constructions overly verbose, introducing unnecessary burden on both the neural processing of Transformer-based large language model and the symbolic derivation of the backend engine. To mitigate these issues, Seed-Geometry groups particular action sequences into specific actions, making the representation of the problem concise enough. Of particular note, Seed-Geometry has several such composite actions: isogonal conjugate with respect to a triangle and a point, exsimilitude center of two circles, insimilitude center of two circles. All three actions can be represented with primitive ruler-and-compass actions; yet the construction sequence in itself is non-trivial and unnecessarily clumsy.

2.1.2 Extremely Fast Reasoning Engine

Seed-Geometry improved its reasoning engine's performance by rewriting its backend in C++ and making it accessible to Python users through Pybind11. This change led to roughly 100-fold speed increase compared to the Python implementation in TongGeometry. The C++ implementation handles memory more efficiently and benefits from compiler optimizations, allowing for much faster **deep** searches within the reasoning engine. This is particularly crucial because the engine's forward-chaining design typically slows down considerably when the search tree expands widely at deeper levels.

2.1.3 Exceptional Large Language Model

Seed-Geometry utilizes a high-performing large language model from the Seed family [17]. This particular Seed model has undergone extensive pre-training on vast datasets of coding and mathematics, granting it a wide array of specialized skills. The specific model size was chosen considering the number of data tokens. We considered training two models in an actor–critic setup initially: the policy model that proposed possible next auxiliary element to construct and a value model that predicted the number of steps to go from the state. However, we note from preliminary experiments that a single Seed model serving as the policy would suffice, contradicting the design of both a policy model and a value model in existing work [30]. We also note that a policy model unspecialized to the specific goal makes both training and solving more manageable and therefore, we only trained the model on pairs of problem context and auxiliaries, without the target fact goal in the prompt.

2.1.4 Extensive Search

When presented with a new problem, Seed-Geometry first transforms the representation into a canonical form. If the reasoning backend successfully finds the goal fact to prove in the reasoning process, the problem is considered immediately solved. Otherwise, Seed-Geometry initiates a search process. In particular, Seed-Geometry employs beam search, with the policy model generating proposals for each beam in the buffer. With the extremely fast reasoning engine, Seed-Geometry supports running each new generated proposal in time. If any one of the proposal leads to the proof, the problem is considered proven; otherwise we select the top few proposals for the next step of expansion in the search tree based on each proposal's cumulative negative log likelihood. The search process terminates until a fixed number of steps have been consumed. Seed-Geometry's search process is made efficient and scalable. Compared to TongGeometry, Seed-Geometry's solving process supports a distributed setup where each GPU process communicates with each other and blocked only at the beam selection point. Each GPU process is also equipped with a CPU thread pool that asynchronously executes the reasoning step during language model inference such that the reasoning cost can be overlapped with language model inference.

Whole proof

```
theorem eg : 1 + 2 = 3 := by
  have h0 : 1 + 1 = 2 := by
    ring

  have h1 : 1 + (1 + 1) = 3 := by
    ring

  linarith [h0, h1]

#print axioms eg
```

Lemma-style

```
lemma round1_h0 : 1 + 1 = 2 := by
  ring

lemma round1_h1 : 1 + (1 + 1) = 3 := by
  ring

theorem eg : 1 + 2 = 3 := by
  linarith [round1_h0, round1_h1]

#print axioms eg
```

Figure 2 An example of whole proof and lemma-style proof in Lean 4.

2.2 Seed-Prover

Seed-Prover is a large language model specialized in formal reasoning in Lean 4. Its most significant distinction from prior work lies in its adoption of lemma-style proving as the proof paradigm, which places **lemmas** at the center of the reasoning process. This approach offers several key advantages: it enables clear identification of lemmas that have and have not been proved, indicating the progress made in solving the main problem; lemmas can be processed independently and combined freely; lemmas from different inference trajectories can be combined to address more challenging problems. Both the training and inference procedures of Seed-Prover are designed around lemmas.

2.2.1 Lemma-Style Proving

Previous works [9, 15, 21] trained the model to generate whole proofs starting with the keyword *theorem*. In contrast, we first require the model to generate some useful lemmas—each introduced by the keyword *lemma*—before generating the main proof using *theorem* by applying the generated lemmas. An example is shown in Figure 2. This lemma-style proof provides following merits. First, it allows clear identification of the lemmas that have been successfully proved, and those that need further refinement. Second, lemmas are modular—they can be compiled independently, stored independently, and combined freely. Additionally, proofs of lemmas may provide inspiration to the model for proving unproved lemmas and the main problem. To enable this workflow, we establish a lemma pool for each difficult problem, which stores comprehensive data from all our inference runs, including lemma statements, lemma names, complete proofs, proof difficulties, and dependency relations. The lemma pool is typically used to (1) retrieve the most relevant lemmas by name or formal statement; (2) sample the most difficult lemmas according to their proof difficulty.

2.2.2 Conjecture Proposing

When tackling challenging contest-level math problems, human contestants often identify interesting properties of the problem and use them to guide their reasoning. Seed-Prover is trained to propose such potentially useful properties by chain-of-thought reasoning. Notably, this process differs from chain-of-thought reasoning to solve the problem directly; rather, it emphasizes broad exploration of the problem space without committing to a particular approach. Take functional equations as an example: one might conjecture that the function is *injective*, *surjective*, *bijective*, *monotonic*, or *periodic*—all without engaging in deep, targeted reasoning about the problem itself. In a sense, many useful properties of a problem may be enumerated before full problem resolution. The proposer module accepts an unsolved problem and, optionally, some already proved lemmas as input, and generates 10–50 candidate conjectures about properties of the problem. Proposing multiple conjectures in parallel significantly increases diversity and the likelihood of covering the valuable properties. For each problem, we may repeat this process multiple times to create a large conjecture pool.

This approach differs from Draft, Sketch, Prove [7], which presumes the ability to fully solve the problem upfront. In contrast, our method performs broad exploratory searches over the problem space enabling

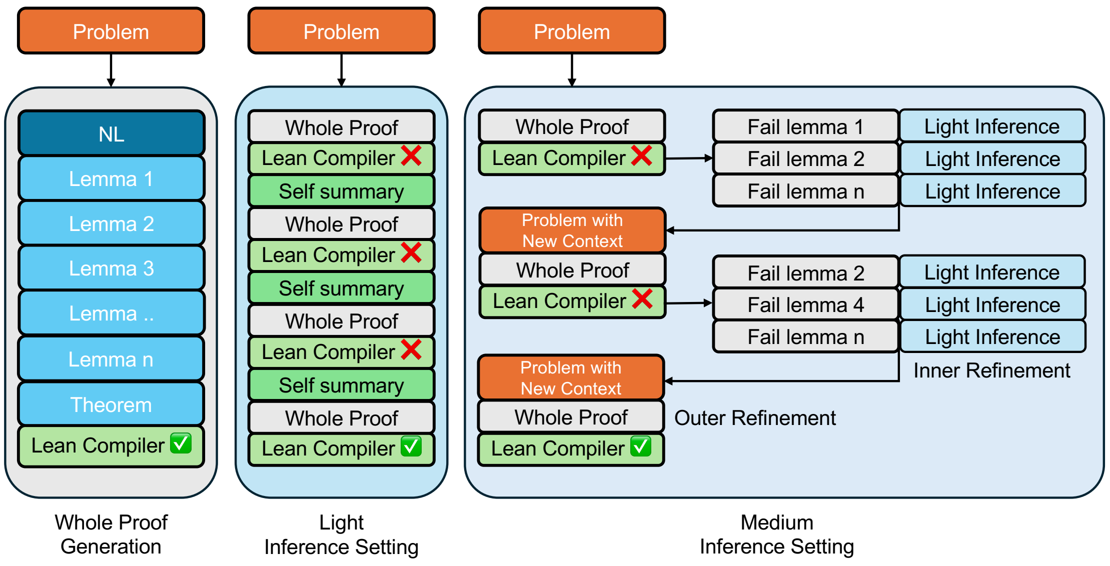

Figure 3 The workflows of single-pass whole proof generation, light, and medium inference settings.

discovery of useful properties, which makes it possible to solve problems that the model cannot solve directly in natural language.

2.2.3 Training

To enable seamless interaction between Seed-Prover and Lean, we adopt multi-stage, multi-task reinforcement learning (RL) based on VAPO [29]. The RL reward is 1 if the formal statement is successfully proven, and 0 otherwise. Additionally, a formatting penalty is applied to encourage the model to generate lemmas before attempting the main theorem. As training progresses, problem diffculty, problem quality, and maximum output length are progressively increased. The training dataset comprises a combination of open-source datasets [1, 8, 12, 27, 28] and in-house formalized problems. For problems that are too difficult for single-pass generation, we use our proposer to generate easier problem variants and put these into the training dataset. We also exclude problems which are too easy (i.e. proof rate above $1/4$) from RL training.

Unlike prior work [15, 21] that only utilize formal statements as prompts for RL training, our approach randomly incorporates natural language hints, natural language proofs, similar lemmas, proved lemmas, failed lemmas, failed attempts, summaries of previous attempts, and Lean compiler feedback into the prompt. This diverse prompting strategy enhances the model's adaptability within our inference pipeline by enabling it to understand and utilize various types of input.

2.2.4 Test-Time Scaling

Here, we introduce our approach to test-time scaling of Seed-Prover. Depending on available inference budgets and problem difficulties, we developed three levels of strategies, which are illustrated in Figure 3 and Figure 4.

Light Previous work evaluated LLM theorem provers by Pass@k. We find that iteratively refining proofs using Lean compiler feedback [32] in conjunction with self-summarization can surpass the limits of single-pass inference token budgets and improve proof ability significantly. In the light setting, each proof attempt is refined up to 8–16 times and evaluated under Pass@8–16. We denote the sample budget of Pass@n and up to m refinements as $n \times m$, so the sample budget of the light setting is equivalent to generating the whole proof at Pass@64–256. This setting completes in 1–2 hours. Under this setting, Seed-Prover proves IMO 2022 P2 ($MOHS = 15$[1]), whereas without refinement, the same problem can only be proved in Pass@8192. We

[1]MOHS is Math Olympiad Hardness Scale: `https://web.evanchen.cc/upload/MOHS-hardness.pdf`. We note that the hardness scale for human contestants may not be well-aligned with the difficulty of proving it in Lean using an LLM.

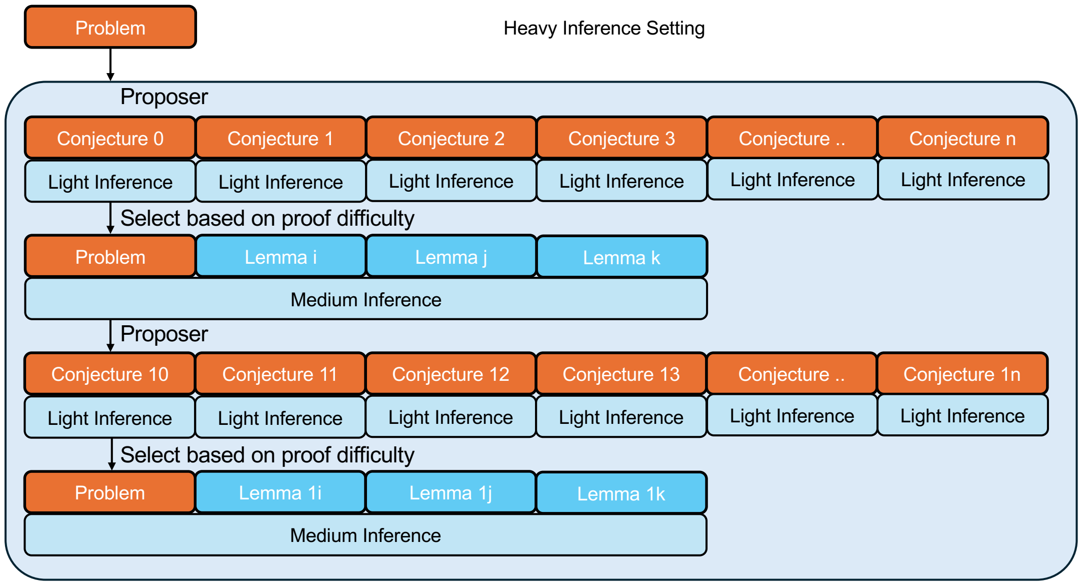

Figure 4 The workflow of heavy inference setting.

observed two dominant behaviors of Seed-Prover in the light inference setting. First, it fixes Lean syntax errors in response to Lean compiler feedback. Second, it refines initial proof sketches, a process that might entirely alter the reasoning trajectory.

Medium Proofs for challenging competition problems are often lengthy and structurally complex. The medium test-time setting introduces both outer and inner refinement processes. The outer refinement process mirrors the light setting which refines the proof of the original main problem. The inner refinement process targets difficult lemmas that the outer refinement process generates but fails to prove, using a light setting with an 8×8 budget to handle finer details. If any inner refinement successfully proves at least one lemma (indicating meaningful progress), the outer refinement process will update this information into the prompt and continue its refinement. Under this setting, Seed-Prover solves harder problems like IMO 2003 P6 ($MOHS = 35$), IMO 2020 P5 ($MOHS = 20$), IMO 2024 P1 ($MOHS = 5$) and IMO 2025 P5 ($MOHS = 15$), and final proofs potentially exceeding 1000 lines of code.

Heavy While the medium setting encourages reasoning in **depth** over proof details, it lacks the **breadth** needed to explore diverse properties of the given problem. Under the heavy inference setting, Seed-Prover begins with a conjecture pool and an empty lemma pool for the given problem. Initially, the proposer generates thousands of conjectures (by default 5000) to populate the conjecture pool. During inference, Seed-Prover tries to prove or disprove every conjecture in the conjecture pool using the light setting. Successfully proved conjectures are moved into the lemma pool. Seed-Prover leverages the lemma pool and Lean compiler feedback to refine the proof attempts. Additional conjectures are proposed based on proved lemmas. After days of thinking, the lemma pool accumulates several thousand nontrivial math facts. Each lemma is scored based on its proof rate, semantic relevance, and proof length. The proof rate serves as a strong indicator of lemma value; empirically, lemmas with low proof rate are often crucial to the final proof. Semantic relevance is assessed by an LLM judge. Unrelated lemmas or lemmas with short proof lengths are removed. We take hundreds of the top-ranked lemmas to help Seed-Prover finish the proof of the main problem using the medium inference setting. Seed-Prover has been trained to select and integrate these lemmas into a complete proof during RL. IMO 2017 P2 ($MOHS = 40$), IMO 2025 P3 ($MOHS = 25$) and IMO 2025 P4 ($MOHS = 15$) are proved under the heavy inference setting.

3 Evaluation

3.1 Seed-Geometry

Using Seed-Geometry's backend, we performed large-scale problem generation. In particular, we took a similar approach to TongGeometry [30] by collecting data statistics of geometry problems over more than the past 20 years, and ran the problem generation program on the trees. In more than 7 days, the problem generation program found over 230 million unique problems, reaching 8x the search efficiency compared to the Python implementation. After necessary preprocessing using a pretrained byte-pair encoding [18], the dataset totaled 38B tokens. We trained both a policy model to complete the auxiliary objects given the context and a value model to estimate the number of steps remaining under the current state. The policy model was initialized from a pretrained Seed model and the value model was initialized with the trained policy model. However, in the experiments, we found that under extensive search, the value model could harm the general performance due to large errors in value estimation. We also compared generating auxiliary actions step-by-step with beam search and generating the whole auxiliary sequence in one go. Results from the latter were significantly inferior to the former. Therefore, in the final evaluation, we used step-by-step generation with beam search in a distributed setup, with each GPU process hosting a policy model for proposal generation.

Table 1 lists the performance of Seed-Geometry in IMO geometry problems from 2000 to 2024. Using the accounting method in IMO-AG-50, Seed-Geometry achieves 43 problem solves compared to AlphaGeometry 2, reaching 1 more solution. The problems that Seed-Geometry cannot solve but AlphaGeometry 2 does are not proof-based problems but rather computation-based problems, which AlphaGeometry 2 could potentially address using its algebraic engine. Table 2 lists the performance of our model in the much harder IMO shortlist problems from 2000 to 2022. The original benchmark of IMOSL-AG-30 was claimed to be established from the problems in the benchmark, but omitted many of the hard problems. Here, we include all hard-level problems in the shortlist and present the full results. As shown in the tables, AlphaGeometry 2 solved 19 of the 39 problems and our Seed-Geometry solved 22 of them. In addition, Seed-Geometry solved IMO 2025 P2 in 2 seconds after it received the human-provided problem formulation.

Based on the results provided, Seed-Geometry has established a new state-of-the-art in automated geometry problem solving, surpassing the performance of its predecessor, AlphaGeometry 2. In summary, by solving more problems in both the standard IMO and the more difficult IMO shortlist benchmarks, Seed-Geometry demonstrates a superior capability in handling complex, proof-based geometry challenges, setting a new performance standard in the field.

Table 1 Performance comparison of AlphaGeometry 2 (AG2) and Seed-Geometry (SG) in IMO geometry problems from 2000 to 2024. Note that in IMO-AG-50, the 2002 P2, 2003 P4, 2004 P5, 2008 P1, and 2009 P4 are separated into two, whereas we merge each of them into one.

ID	AG2	SG
2000 P1	✓	✓
2000 P6	✓	✓
2001 P1	X	X
2001 P5	✓	X
2002 P2	✓	✓
2002 P6	X	X
2003 P3	X	X
2003 P4	✓	✓
2004 P1	✓	✓
2004 P5	✓	✓
2005 P1	✓	✓
2005 P5	✓	✓
2006 P1	X	✓
2006 P6	X	X
2007 P2	✓	✓
2007 P4	✓	✓
2008 P1	✓	✓
2008 P6	✓	✓
2009 P2	✓	✓
2009 P4	✓	X
2010 P2	✓	✓
2010 P4	✓	✓
2011 P6	✓	✓
2012 P1	✓	✓
2012 P5	✓	✓
2013 P3	✓	✓
2013 P4	✓	✓
2014 P3	✓	✓
2014 P4	✓	✓
2015 P3	✓	✓
2015 P4	✓	✓
2016 P1	✓	✓
2017 P4	✓	✓
2018 P1	✓	✓
2018 P6	X	✓
2019 P2	✓	✓
2019 P6	✓	✓
2020 P1	✓	✓
2020 P6	X	X
2021 P3	✓	✓
2021 P4	✓	✓
2022 P4	✓	✓
2023 P2	✓	✓
2023 P6	X	✓
2024 P4	✓	✓

Table 2 Performance comparison of AlphaGeometry 2 (AG2) and Seed-Geometry (SG) in IMO shortlist geometry problems from 2000 to 2022. Note that in IMOSL-AG-30, many geometry problems are ignored and here we fill in those missing geometry problems for comparison, with 2016 G7 merged into one.

ID	AG2	SG
2002 G7	✓	✓
2002 G8	✓	✓
2003 G5	✓	✓
2004 G7	✗	✓
2004 G8	✗	✓
2005 G5	✓	✓
2005 G6	✗	✓
2006 G9	✓	✓
2007 G8	✗	✗
2009 G6	✓	✓
2009 G7	✓	✗
2009 G8	✓	✗
2010 G5	✓	✓
2011 G3	✗	✗
2011 G4	✗	✓
2011 G5	✗	✓
2011 G6	✓	✗
2011 G7	✓	✗
2012 G6	✗	✓
2012 G7	✗	✗
2012 G8	✗	✓
2014 G7	✓	✓
2015 G5	✓	✓
2015 G7	✗	✗
2015 G8	✗	✗
2016 G5	✓	✓
2016 G6	✓	✓
2016 G7	✓	✓
2016 G8	✗	✗
2017 G7	✗	✗
2017 G8	✗	✗
2018 G7	✓	✓
2019 G6	✓	✓
2019 G8	✗	✗
2020 G8	✓	✓
2021 G8	✗	✗
2022 G6	✗	✗
2022 G7	✗	✗
2022 G8	✗	✗

Table 3 Performance comparison of Seed-Prover against previous systems across formal math tasks. The performance on PutnamBench is using the number of proved statements instead of percentage following previous works [20].

Metric	Seed-Prover	Previous SOTA
IMO 2025	4/6 (Heavy, 5/6 post-competition)	5/6 (Natural language, Gemini)
Past IMO	78.1% (Heavy)	—
MiniF2F-valid	100.0% (Medium[1])	90.6% (DeepSeek-Prover-V2 [15])
MiniF2F-test	99.6% (Medium[2])	92.2% (Kimina-Prover [21])
PutnamBench	331/657 (Medium)	86/657 (Goedel-Prover-V2 [9])
CombiBench	30.0% (Medium)	10.0% (Deepseek-Prover-V2 [15])
MiniCTX-v2	81.8% (Light)	44.3% (o4-mini [16])

[1]One problem used Heavy. [2]One problem failed under Heavy.

3.2 Seed-Prover

To evaluate the performance of Seed-Prover, we tested it on IMO 2025[2], past IMO problems, MiniF2F [31], PutnamBench [20], CombiBench [10], and MiniCTX-v2 [6] covering a wide range of mathematical domains. For PutnamBench and CombiBench, we first evaluate using the light setting and use the medium setting for unsolved problems. For IMO problems and MiniF2F, we also use the heavy setting for the remaining unsolved problems. The results are shown in Table 3. Unless otherwise specified, we use Lean v4.14.0 with its corresponding Mathlib version.

IMO 2025 During the IMO 2025 contest, all problems were translated into formal statements by human experts. For fill-in-the-blank problems ("determine x such that ..."), initial solution candidates were generated by Seed1.6-Thinking before translation. We conducted searches for IMO 2025 Problems 1, 3, 4, and 5 using both the medium and heavy inference settings in parallel. As required by the IMO committee, all proof submissions were due by July 18th. Seed-Geometry solved Problem 2 instantly, and Seed-Prover derived the proof for Problem 5 under the medium inference setting, while proofs for the other three problems required the heavy inference setting. Notably, the proof for Problem 1 was finished after the deadline.

[2]ByteDance was officially invited to participate in IMO 2025.

Past IMO To evaluate our system's performance on past IMO problems, we curated a dataset consisting of 155 past IMO problems. Most problems were adapted from Compfiles[3] and MiniF2F [31]. Additionally, a subset of problems was manually added or corrected by human experts. For problems prior to 2017, we used light and medium settings. For problems after 2017, we use the heavy inference setting if the medium inference setting failed. Seed-Prover successfully proves 121/155 problems, achieving an overall success rate of 78.1%. By difficulty, Seed-Prover proves 47/55 easy problems (P1 or P4), 47/56 medium problems (P2 or P5), and 27/44 hard problems (P3 or P6). By subject area, it proves 72/85 algebra problems, 42/55 number theory problems, and 7/14 combinatorics problems. This demonstrates that the Seed-Prover's performance at IMO 2025 reflects consistent capability on IMO problems across all years.

MiniF2F [31] Under the medium setting, we prove 99.6% problems on both MiniF2F-valid and MiniF2F-test. We used the heavy inference setting to tackle the last problem in both splits (i.e. IMO 1990 P3 and IMOSL 2007 Algebra P6). Seed-Prover successfully proved IMO 1990 P3 and failed on IMOSL 2007 Algebra P6. Interestingly, among the most challenging problems solved in MiniF2F are ones such as AMC12A 2021 P12, AMC12A 2021 P25, and AMC12A 2020 P9, which are relatively straightforward to reason about in natural language, but pose significant challenges when formalized in Lean. This difficulty arises primarily from obstacles in applying Vieta's formulas or the non-triviality of counting roots.

PutnamBench [20] We evaluated Seed-Prover on PutnamBench using the light and medium inference settings. Under the light inference setting only, Seed-Prover proved 201/657 problems from PutnamBench. Using the medium inference setting improved this performance to 331/657 problems. This result shows a significant performance jump compared to previous works on undergraduate math problems.

CombiBench [10] Previous benchmarks have primarily focused on number theory and algebra problems. CombiBench is a benchmark specifically centered on combinatorial problems, where the problems often involve newly-defined concepts. Here, we evaluate Seed-Prover on CombiBench using the medium inference setting. Our model proves 30 out of 100 problems from CombiBench, outperforming previous work. Nonetheless, relative to other benchmarks, our model still struggles with proving combinatorics problems.

MiniCTX-v2 [6] To test our system on a broader range of mathematical subjects from real-world formalization projects—including the ability to understand new definitions, notations, and lemmas—we evaluated Seed-Prover on MiniCTX-v2. This dataset includes context-rich problems from formalization repositories in subjects such as axiomatic systems, high-energy physics, analysis, and research-level number theory, all of which were written after Nov. 2024 to prevent data contamination. For evaluation purposes, we used the light inference setting under Lean v4.16.0. Our system successfully achieved 81.8% of MiniCTX-v2, which demonstrates its strong potential in real-world automated theorem proving, generalizing beyond standalone competition problems. For comparison, the baseline o4-mini solved 44.3% statements at Pass@8 [16].

4 Conclusion

In this work, we presented Seed-Geometry and Seed-Prover—two formal reasoning frameworks that integrate the capabilities of large language models. Both systems substantially outperform previous formal reasoning frameworks. Seed-Geometry accelerates verification and scales the search mechanism. Seed-Prover leverages iterative refinement and a three-tiered test-time inference strategy to achieve state-of-the-art. Notably, we successfully proved 5 out of 6 problems in IMO 2025, demonstrating the efficacy of these formal systems. Formal languages like Lean offer rapid proof verification, making them far more cost-effective than human experts and more reliable than LLM judges. Our future work will focus on combining formal systems with large language models to tackle open conjectures.

[3]https://github.com/dwrensha/compfiles

References

[1] Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and Nick Haber. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models, 2025. URL https://arxiv.org/abs/2502.17387.

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

[3] DeepMind. Ai solves imo problems at silver medal level. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, 2024.

[4] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[5] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL https://arxiv.org/abs/2502.00212.

[6] Jiewen Hu, Thomas Zhu, and Sean Welleck. miniCTX: Neural theorem proving with (long-)contexts. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=KIgaAqEFHW.

[7] Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In The Eleventh International Conference on Learning Representations, 2023.

[8] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. [https://github.com/project-numina/aimo-progress-prize](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

[9] Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. arXiv preprint arXiv:2502.07640, 2025.

[10] Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, et al. Combibench: Benchmarking llm capability for combinatorial mathematics. arXiv preprint arXiv:2505.03171, 2025.

[11] OpenAI. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

[12] Zhongyuan Peng, Yifan Yao, Kaijing Ma, Shuyue Guo, Yizhe Li, Yichi Zhang, Chenchen Zhang, Yifan Zhang, Zhouliang Yu, Luming Li, et al. Criticlean: Critic-guided reinforcement learning for mathematical formalization. arXiv preprint arXiv:2507.06181, 2025.

[13] Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating llms on 2025 usa math olympiad, 2025. URL https://arxiv.org/abs/2503.21934.

[14] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. arXiv preprint arXiv:2202.01344, 2022.

[15] ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. arXiv preprint arXiv:2504.21801, 2025.

[16] Tate Rowney, Jiewen Hu, Thomas Zhu, and Sean Welleck. miniCTX leaderboard. https://cmu-l3.github.io/minictx/leaderboard.html, 2025. Accessed July 28, 2025.

[17] ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. arXiv preprint arXiv:2504.13914, 2025.

[18] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909, 2015.

[19] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. Nature, 625(7995):476–482, 2024.

[20] George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[21] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. 2025. URL http://arxiv.org/abs/2504.11354.

[22] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems, 2024. URL https://arxiv.org/abs/2410.15700.

[23] Zijian Wu, Jiayu Wang, Dahua Lin, and Kai Chen. Lean-github: Compiling github lean repositories for a versatile lean prover, 2024. URL https://arxiv.org/abs/2407.17227.

[24] Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

[25] Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving, 2025. URL https://arxiv.org/abs/2502.03438.

[26] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. Advances in Neural Information Processing Systems, 36:21573–21612, 2023.

[27] Huaiyuan Ying, Zijian Wu, Yihan Geng, JIayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[28] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. arXiv preprint arXiv:2503.14476, 2025.

[29] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

[30] Chi Zhang, Jiajun Song, Siyu Li, Yitao Liang, Yuxi Ma, Wei Wang, Yixin Zhu, and Song-Chun Zhu. Proposing and solving olympiad geometry with guided tree search. arXiv preprint arXiv:2412.10673, 2024.

[31] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In International Conference on Learning Representations.

[32] Yichi Zhou, Jianqiu Zhao, Yongxin Zhang, Bohan Wang, Siran Wang, Luoxin Chen, Jiahui Wang, Haowei Chen, Allan Jie, Xinbo Zhang, et al. Solving formal math problems by decomposition and iterative reflection. arXiv preprint arXiv:2507.15225, 2025.

Appendix

A Contributors

The names are sorted alphabetically. An asterisk * indicates a member who left Seed.

Algorithm Luoxin Chen, Liankai Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, Jiahui Wang, Siran Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Huajian Xin, Fan Yang, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang*, Yichi Zhou, Thomas Hanwen Zhu

Data Jinming Gu, Wenhao Huang, Zhicheng Jiang, Xiaoran Jin, Kaijing Ma, Jiawei Shen, Tong Sun, Chenrui Wei, Shufa Wei, Yuchen Wu, Yihang Xia, Huaiyuan Ying*, Zheng Yuan, Ge Zhang

Infra Cheng Ren, He Sun, Zhihong Wang, Tianyun Zhao*, Jianqiu Zhao, Thomas Hanwen Zhu

B LooKeng: An Easy-to-Use and Effective Python Interface for Lean

Interacting with Lean poses significant challenges that limit the flexibility of Lean-based workflows. The most popular interface, LeanDojo [26], only supports earlier versions of Lean 4, restricting users from accessing Lean's newest updates. Furthermore, LeanDojo requires creating a Lean repository for interaction, which makes it impractical to use considering the massive scale of Lean interaction during model development and inference. To address these issues, we introduce LooKeng, a REPL[4]-based Python interface designed to simplify and accelerate the interaction process. LooKeng offers powerful features for developers while providing a user-friendly interface for end-users. The core functionality of LooKeng includes 'init_state', 'run_tac', and 'verify_proof'. One can use LooKeng to interact with Lean step-by-step or verify an entire proof directly. The key features of LooKeng are summarized as follows:

- **Stateless Design**: A Lean state can be simultaneously processed using different LooKeng instances, enabling effortless scaling and sharing.
- **Complex Tactics**: Complex tactics such as `apply?` and `all_goals` are fully supported, with enhanced infotree integration to prevent false positive proofs.
- **Version-Free**: The LooKeng CLI allows users to manage and switch between different Lean versions.
- **Memory Control**: Users can easily track the memory consumption of the Lean backend, set custom thresholds, and automatically terminate processes when memory usage exceeds the limit.
- **Proof Verification**: LooKeng provides a straightforward method, 'verify_proof', to rigorously verify the final proof using the native Lean interface, ensuring correctness and reliability.
- **Proof Simplification**: LooKeng can remove useless tactics and hypothesis in the proof to obtain a simpler proof.
- **Statement Negation**: LooKeng is able to generate the negated statement of a statement.
- **Multi-Concurrency Support**: LooKeng can run as a service, handling thousands of concurrent requests via async architecture and resource isolation.

[4]https://github.com/leanprover-community/repl