Discover and Prove: An Open-source Agentic Framework for Hard Mode Automated Theorem Proving in Lean 4

Chengwu Liu[1]*, Yichun Yin[2], Ye Yuan[1], Jiaxuan Xie[3], Botao Li[4], Siqi Li[4], Jianhao Shen[2], Yan Xu[2], Lifeng Shang[2], Ming Zhang[1],

[1]State Key Laboratory for Multimedia Information Processing, School of Computer Science, PKU-Anker LLM Lab, Peking University [2]Huawei Technologies Co., Ltd.
[3]School of Software & Microelectronics, Peking University
[4]School of Electronics Engineering and Computer Science, Peking University

Correspondence: Ming Zhang (mzhang_cs@pku.edu.cn)

Abstract

Most ATP benchmarks embed the final answer within the formal statement — a convention we call "Easy Mode" — a design that simplifies the task relative to what human competitors face and may lead to optimistic estimates of model capability. We call the stricter, more realistic setting "Hard Mode": the system must independently discover the answer before constructing a formal proof. To enable Hard Mode research, we make two contributions. First, we release MiniF2F-Hard and FIMO-Hard, expert-reannotated Hard Mode variants of two widely-used ATP benchmarks. Second, we introduce Discover And Prove (*DAP*), an agentic framework that uses LLM natural-language reasoning with explicit self-reflection to discover answers, then rewrites Hard Mode statements into Easy Mode ones for existing ATP provers. *DAP* sets the state of the art: on CombiBench it raises solved problems from 7 (previous SOTA, Pass@16) to 10; on PutnamBench it is the first system to formally prove 36 theorems in Hard Mode — while simultaneously revealing that state-of-the-art LLMs exceed 80% answer accuracy on the same problems where formal provers manage under 10%, exposing a substantial gap that Hard Mode benchmarks are uniquely suited to measure.

1 Introduction

The use of AI to solve mathematical problems has attracted considerable research interest, not only because of the potential for concrete applications in domains like education and mathematical research, but also because tackling highly abstract mathematical problems generally requires capabilities that may generalize and transfer to complex real-world tasks. These capabilities include planning, search, deductive reasoning, and induction. (Yang et al., 2024) Within the spectrum of mathematical tasks, competition problems — especially those at the International Mathematical Olympiad (IMO) level — have garnered particular attention. These problems go beyond numerical computation or simple formula application. They typically demand abstraction and modeling, rigorous logical argumentation, and often require elements of intuition and creativity. Accordingly, the ability to solve IMO-level problems is widely regarded as an important milestone for AI (Yang et al., 2024).

Existing approaches to solving mathematical problems fall into two broad categories: informal methods and formal methods. Informal methods solve mathematical problems in natural language and leverage the strong reasoning abilities of large language models (LLMs), whereas formal methods use formal languages such as Lean (Moura and Ullrich, 2021) and Isabelle (Nipkow et al., 2002) to express the solution. A key advantage of formal methods is that proofs written in formal languages can be automatically and rigorously verified by a proof assistant program. At the International Mathematical Olympiad (IMO) 2024, participating AI systems employed formal methods (AlphaProof and AlphaGeometry, 2024). By IMO 2025, however, most evaluated systems had shifted toward informal approaches (Luong et al., 2025; Huang and Yang, 2025; Wei, 2025; Huawei-xiaoyi, 2025).

We observe that current practice in many formalization efforts often embeds the final answer directly into the statement to be proved, which we refer to as "Easy Mode". We point out that Easy Mode may substantially reduce the difficulty of formal problem-solving tasks. To address this issue, we draw inspiration from prior works (PutnamBench (Tsoukalas et al., 2024) and CombiBench (Liu et al., 2025)): answer-oriented problems are encoded in Lean 4 with two separate goals (two distinct `sorry`s). In this "Hard Mode" config-

*Work done during the internship at Huawei Technologies Co., Ltd. Code and datasets are available at https://github.com/liuchengwucn/discover-and-prove.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 117–133
July 2-7, 2026

uration, the model must first supply the final answer by replacing the first `sorry` with the answer and then produce a conventional formal proof for the remaining goal. This setup prevents embedding extra information that human contestants must discover by themselves in the formal statement. By definition, *Hard Mode* requires that any quantity a human competitor must derive through reasoning is not supplied as a premise in the formal statement; it must be independently discovered by the AI system. We adopt the term "Hard Mode" following the convention in the Lean community;[1] CombiBench (Liu et al., 2025) refers to the same distinction as "without solution" vs. "with solution", while PutnamBench (Tsoukalas et al., 2024) uses "no answer" vs. "with answer". An example illustrating the difference between Easy Mode and Hard Mode is presented in Figure 1. We commissioned expert annotators to reannotate two widely used ATP competition datasets, namely MiniF2F and FIMO, producing MiniF2F-Hard and FIMO-Hard. During reannotation, we corrected known alignment issues in existing formal benchmarks (Wang et al., 2025a; Lin et al., 2025b).

To solve Hard Mode problems, we introduce the Discover and Prove (*DAP*) framework, a fully open-source, agent-based ATP framework for Hard Mode ATP tasks. *DAP* consists of two components: a Discovery Module and a Proving Module. We prompt an open-source LLM to generate and iteratively refine its reasoning and answers using self-verification procedures. After the Discovery module "discovers" a plausible answer and fills the first `sorry`, the Proving module attempts to produce complete formal proofs by invoking traditional ATP provers. *DAP* achieves state-of-the-art results. Evaluating on the full PutnamBench dataset, *DAP* solves 36 problems in total. On solution-style problems with Hard Mode variants, it solves 19 problems — to our knowledge, this constitutes the first public result on PutnamBench under this Hard Mode evaluation setting. On CombiBench Hard Mode, it solves 10 problems, improving significantly on the previous state of the art (Kimina-Prover Preview), which solved 8 problems.

Our contributions are threefold.

1. We reannotate two commonly used ATP competition datasets, MiniF2F and FIMO, to align tasks presented to human competitors with those given to AI systems, removing the Easy Mode discrepancy and providing a more principled basis for evaluating AI mathematical capability.

2. We propose *DAP*, an open-source, agentic Hard Mode ATP framework. With a simple and straightforward design, *DAP* achieves state-of-the-art performance on PutnamBench and CombiBench.

3. We provide an analysis quantifying the individual contributions of the two modules of our proposed *DAP* prover to overall performance. These results shed light on the relative strengths of informal and formal method approaches for solving competition-level mathematical problems.

2 Related Work

2.1 Mathematical Problem-Solving with AI systems

Powered by CoT prompting and RLVR training, LLMs have made substantial progress in mathematical reasoning. Frontier models (e.g., OpenAI o1 (Jaech et al., 2024), DeepSeek R1 (Guo et al., 2025), Google Gemini 2.5 Pro (Comanici et al., 2025)) now achieve near-saturation performance on widely-used math benchmarks, including GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), and AIME 2024/2025.

One downside of the informal methods is that the solution traces they produce are notoriously difficult to verify automatically. Assessing the validity of a generated proof would require domain experts to inspect it carefully to detect subtle errors (Lightman et al., 2023), which is infeasible at scale. For this reason, informal systems are typically applied to problems that require only a final numerical or symbolic answer, verified by direct comparison against ground-truth (Wen et al., 2025).

2.2 Automated Theorem Proving

The key strength of automated theorem proving (ATP) is that formal proofs can be checked rigorously and automatically by proof assistants (Yousefzadeh and Cao, 2025; Wang et al., 2023). Although formal approaches have faced challenges of limited formal-data availability (Xin et al., 2024),

[1] https://leanprover.zulipchat.com/#narrow/channel/208328-IMO-grand-challenge/topic/IMO.202025.20problem.20statements

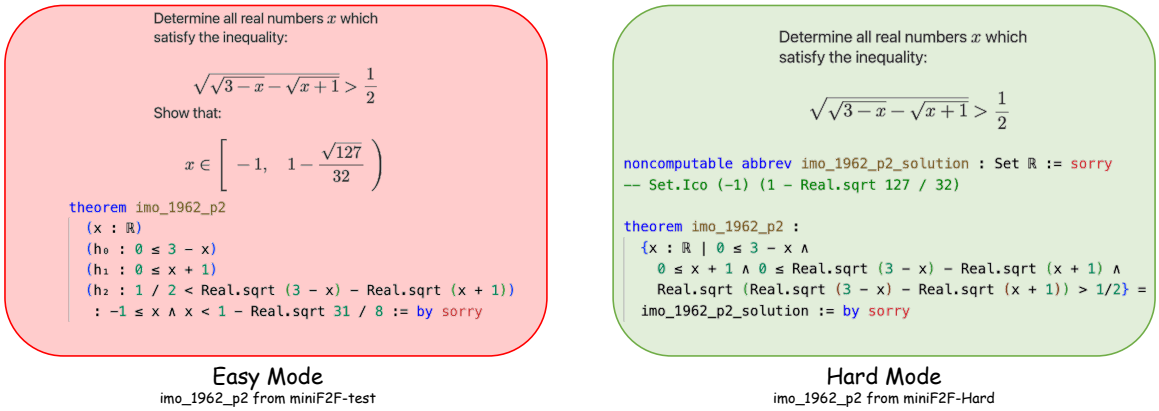

Figure 1: Differences between Easy Mode and Hard Mode configurations in automated theorem proving. The example shown is a Lean 4 formalization of an IMO problem in two different styles. This example was intentionally selected to illustrate the kind of semantic misalignment that our re-annotation effort corrects: the Easy Mode formalization proves only that x must lie within certain ranges, but does not establish that every value in those ranges is attainable, weakening the original if-and-only-if requirement to a one-directional implication. By contrast, our Hard Mode formalization represents the answer as a set, thereby fully capturing the natural-language problem's requirement in the formal statement.

they have advanced rapidly with larger LLMs and scaling searching compute (Xin et al., 2025; Chen et al., 2025). Recent systems such as Kimina-Prover Preview (Wang et al., 2025a), DeepSeek-Prover-V2 (Ren et al., 2025), and Goedel-Prover-V2 (Lin et al., 2025b) have made substantial progress on MiniF2F (Zheng et al., 2021) benchmark. Seed-Prover (Chen et al., 2025) is a lemma-style whole-proof reasoning model that iteratively refines proofs via Lean compiler feedback, proved lemmas, and self-summarization, achieving over 50% on PutnamBench and saturating MiniF2F. DSP (Jiang et al., 2023) guides formal theorem provers with natural-language draft proofs and structured sketches; DSP+ (Cao et al., 2025) revives this paradigm with modern reasoning models. DAP differs from DSP/DSP+ in three key respects; see Appendix B for a detailed comparison.

Several agentic frameworks have also targeted Lean-based ATP (Thakur et al., 2023; Kumarappan et al., 2024; Baba et al., 2025; Wang et al., 2025b); see Appendix A.

To handle informal mathematical problems that require a final answer, prior efforts typically convert problems requiring solutions into proof problems by embedding the desired answer into the formalized statement and proving that statement (Zheng et al., 2021; Liu et al., 2023; Xiong et al., 2023). This practice raises two concerns. First, embedding the answer in the statement can reduce the intrinsic difficulty: for many solution-oriented problems, the primary challenge is discovering the answer rather than proving a consequence once known, so supplying it acts as a substantial hint. Second, the resulting formalized statements are sometimes not perfectly semantically aligned with the tasks human contestants face. As prior analyses (e.g., FMC (Xie et al., 2025), OlympiadBench (He et al., 2024)) have noted, some existing formal benchmarks are misaligned: some formal statements may capture only a subset of the goals that human solvers must address.

2.3 Formalization & Data Curation

High-quality formal datasets are scarce and require substantial expert effort (Xin et al., 2024). MiniF2F (Zheng et al., 2021) is one of the most widely used ATP benchmarks, containing formalized statements from mathematical olympiads and high-school and undergraduate courses; Kimina-Prover Preview (Wang et al., 2025a) supplies corrections to a subset of its problems. FIMO (Liu et al., 2023) is constructed from IMO shortlist problems but lacked a publicly available Lean 4 version, which impeded its adoption in recent work. PutnamBench (Tsoukalas et al., 2024) comprises hand-constructed formalizations of Putnam Competition problems and, for the first time, provides both "with answer" and "no answer" evaluation settings. CombiBench (Liu et al., 2025) offers 100 combinatorics problems ranging from middle-school level through IMO and university

level, complementing the number-theoretic and algebraic focus of the other benchmarks; IMOSL-Lean4 (Sanjaya, 2025) and IMO-Steps (Yousefzadeh and Cao, 2024) additionally supply complete proofs. To alleviate data scarcity, autoformalization methods (Wu et al., 2022) automatically translate informal problems into formal statements. Representative resources include Lean Workbook (Ying et al., 2024), NuminaMath-LEAN (Wang et al., 2025a), Goedel-Pset-v1 (Lin et al., 2025a), and FormalMATH (Yu et al., 2025). Because outputs are typically validated only via LLM-as-a-judge (Ying et al., 2024), they lack guaranteed semantic correctness and are more commonly used in provers' RL training phases (Wang et al., 2025a; Xin et al., 2024) than as evaluation benchmarks. To our knowledge, all autoformalization techniques follow the Easy Mode paradigm, embedding the desired answer directly into each generated statement.

3 Methodology

To address the challenge of proving Hard Mode Lean 4 theorems containing two `sorry` placeholders, we propose the Discover and Prove (*DAP*) framework, designed to emulate a human mathematician's approach by providing both the answer and a detailed proof. As its name suggests, *DAP* consists of two primary modules: a Discovery Module and a Proving Module. The Discovery Module operates in natural language, tasked with identifying the correct solution to the problem, and subsequently transforms the original Hard Mode Lean 4 statement into an Easy Mode statement. This transformation reduces the number of `sorry` placeholders from two to one, thereby leaving a single statement that can be resolved by conventional automated theorem provers. Then, the Proving Module utilizes the Lean 4 formal language to construct a rigorous proof for the solution identified by the Discovery Module. The complete workflow of the DAP framework is depicted in Figure 2.

3.1 Discovery Module

The Discovery Module aims to solve the original mathematical problem and substitute the solution into the first `sorry` placeholder of the Lean 4 Hard Mode statement for the Proving Module to use. This process mirrors human problem-solving by hypothesizing an answer to guide the proof search. Although the reasoning capabilities of LLMs have substantially improved following the introduction of long Chain-of-Thoughts such as OpenAI o1 (Jaech et al., 2024) and DeepSeek R1 (Guo et al., 2025), the one-shot resolution of highly challenging, IMO-level problems remains unsolved. Current research indicates that state-of-the-art LLMs often require auxiliary tools (e.g., information retrieval, calculators, external memory) to support deep reasoning (Nakano et al., 2022; Luo et al., 2025; Yuan et al., 2024). Inspired by prior work on LLM reasoning systems (Huang and Yang, 2025), we employ a relatively straightforward configuration where an advanced reasoning model generates solution steps and performs self-verification to enhance accuracy. Specifically, the procedure involves the following steps:

(1) **Solution Generation:** Given a mathematical problem in natural language, the model's reasoning capabilities are leveraged to generate a detailed chain-of-thought describing the solution process.
(2) **Self-Verification:** The reasoning LLM is instructed to inspect its steps for potential errors and produce an error report identifying any erroneous locations (a representative error report is shown in Appendix C.5). If self-verification reveals no errors, the process proceeds to step 4; otherwise, it moves to step 3.
(3) **Self-Correction:** The reasoning LLM is instructed to generate a revised solution that addresses the errors identified in the error report.
(4) **Rewriting:** Using the Lean 4 Hard Mode statement, the natural-language problem, and the model's chain-of-thought reasoning, the LLM is prompted to produce a rewritten Lean 4 statement containing only a single placeholder, suitable for automated theorem proving.

All four steps are implemented through meticulously designed prompts to the LLM, which are detailed in Appendix C. The Discovery Module is crucial because incorrect solutions frequently lead to flawed formal statements that are unprovable from the outset, thus emphasizing the model's capacity for deep reasoning and reliable self-verification. This framework is released as open-source to facilitate reproducibility and serve as a baseline. For the Discovery Module, we utilize the open-source model GPT-OSS-120B, known for its strong mathematical reasoning performance. In principle, the framework can be instantiated with any model that combines robust mathematical reasoning with basic Lean proficiency.

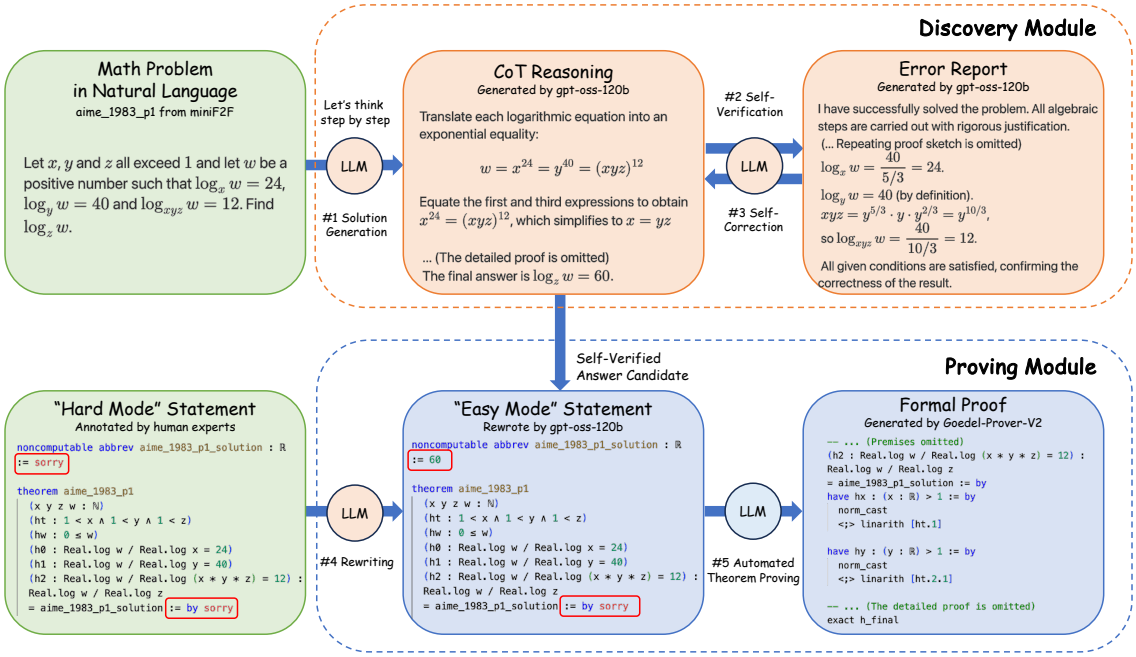

Figure 2: Primary flowchart. A mathematical problem is first processed by the Discovery Module to generate a solution; this solution is then incorporated into the Easy Mode statement during the rewriting stage. Orange circles denote the reasoning LLM, and blue circles denote the theorem prover (another distinct LLM).

3.2 Proving Module

Once the Discovery Module transforms a Hard Mode statement into an Easy Mode formulation, the task becomes a standard ATP problem, amenable to conventional theorem provers. For this purpose, we employ Goedel-Prover-V2 (32B), a state-of-the-art open-source theorem prover, to process the transformed problems. By decoupling the reasoning model from the ATP model, the proposed framework can improve as any of the underlying models advance, and provides a contemporary baseline for evaluating LLMs that generate formal mathematical solutions in a proof assistant language.

4 Data Curation

4.1 Annotation Principles

Our re-annotation is guided by three principles.

Semantic Accuracy: Expert-annotated datasets are small but reliable; auto-formalized datasets are large but unverifiable — current validation methods (LLM-as-a-judge (Ying et al., 2024), BEq (Liu et al., 2024)) cannot guarantee semantic alignment with the original natural-language problem. We therefore start from expert-annotated sources and re-examine each statement manually.

Interpretability: A formal statement should reflect exactly what a human contestant is asked to do: any quantity to be discovered must not be supplied as a premise, and any claim to be proved must appear as the goal. Current benchmarks violate this in three recurring ways: (1) encoding the final answer in the statement, (2) weakening the proof goal to a strict subset of the original, or (3) adding premises that human contestants do not have. Representative instances are shown in Figure 1; we followed the IMOLean (Myers, 2025) convention throughout.

Consistency: Formalization style can materially affect ATP success rates (Lin et al., 2025a), so permitting annotators to choose arbitrary formalizations would introduce evaluation bias. We provided a unified convention emphasizing idiomatic Lean — "go further in the direction of idiomatic Lean rather than trying to follow a particular English version closely'' — consistent with the IMOLean (Myers, 2025) practice of providing a single canonical statement.

Table 1: The table presents, for commonly used automated theorem proving datasets, the number of statement samples, the number of Hard Mode problems, the data sources, data curation methods, and compatibility with different versions of the Lean formal language. Notably, ProofNet and miniF2F were originally released as Lean 3 datasets, and publicly available community ports to Lean 4 exist. Although studies have reported performance on a Lean 4 variant of FIMO, no publicly available Lean 4 version of the FIMO dataset exists.

	# Samples	# Hard Mode	Data Source	Curation Method	Compatibility
ProofNet	371	N/A	Textbook	Expert Annotation	Lean 3 & 4
miniF2F-test	244	N/A	Textbook & Competition	Expert Annotation	Lean 3 & 4
FIMO	149	N/A	Competition	Expert Annotation	Lean 3
PutnamBench	660	340	Competition	Expert Annotation	Lean 4
CombiBench	100	45	Textbook & Competition	Expert Annotation	Lean 4
Lean Workbook	57k	N/A	AoPS Website	Auto-formalization	Lean 4
NuminaMath-LEAN	104k	N/A	Competition	Auto-formalization & Expert Annotation	Lean 4
MiniF2F-Hard (Ours)	244	194	Textbook & Competition	Expert Annotation	Lean 4
FIMO-Hard (Ours)	149	70	Competition	Expert Annotation	Lean 4

4.2 Data Selection

To address these deficiencies, we engaged Lean experts to reannotate two human-expert–annotated datasets, namely MiniF2F (Zheng et al., 2021) and FIMO (Liu et al., 2023). Although ProofNet is also a high-quality, expert-annotated dataset, we observed that all of its natural-language items are inherently proof-based; consequently, we excluded it from consideration. The annotators each have more than one year of Lean-related experience. The MiniF2F and FIMO datasets were originally produced by experts; we re-examined them in light of error reports documented in the literature (Wang et al., 2025a) and had each problem independently annotated by two experts to cross-validate labels and thereby safeguard correctness.

4.3 Dataset Quality Fixes

Beyond creating Hard Mode variants, our annotators performed three non-trivial categories of quality-improvement work; full details are in Appendix D. **Porting FIMO to Lean 4:** we ported all FIMO problems from Lean 3, verifying compilation and semantic faithfulness. **Fixing semantic misalignments:** we identified and repaired ≈15 errors in miniF2F and ≈20 in FIMO across four error types (Table 6). **Rephrasing for Hard Mode:** unknown values were promoted to free parameters with explicit side-conditions; see Figure 3 (Appendix E).

5 Experiments

For the Discovery Module, we use the open-source model GPT-OSS-120B, which demonstrates strong performance on natural language mathematical reasoning tasks. For the Proving Module, we use Goedel-Prover-V2, which exhibits state-of-the-art performance in traditional automated theorem proving. Our formal verification environment uses Lean 4.15.0, with Kimina-Server (Dos Santos et al., 2025) mediating interactions with the Lean 4 REPL.

For GPT-OSS-120B, we follow the model's recommended configuration with sampling temperature 1.0. During Self-Verification, the model is allowed up to 30 iterative attempts for self-checking and error correction. If all attempts fail, the pipeline falls back to the no-agent (ablation) configuration, where the reasoning model's output is used directly as the answer candidate without verification. For Goedel-Prover-V2, we follow the developer-recommended sampling configuration: temperature 0.7, `max_tokens` set to 30,000, 32 samples, and Pass@32 evaluation metric. We report performance on PutnamBench, CombiBench, miniF2F-Hard, and FIMO-Hard in Table 2.

Our method achieves state-of-the-art performance on Hard Mode problems. On CombiBench, it solves 10 problems, improving on the prior state-of-the-art (8 problems); it is also the first method to solve problems under PutnamBench's "No Answer" configuration. Notably, on PutnamBench and miniF2F-test, our method's Hard Mode accuracy closely approaches Goedel-Prover-V2's Easy

Table 2: Performance of various open-source approaches on standard and Hard Mode benchmarks. The numerals beneath each dataset name indicate the total number of examples and the number of Hard Mode examples. Best results under each configuration are indicated in boldface. In Hard Mode evaluation, each dataset contains proof-style problems (evaluated in their original form) and solution-style problems (evaluated with the answer not provided in the formal statement). Each entry X/Y denotes total problems solved (X) and solution-style problems solved in Hard Mode (Y). All results are Pass@32 unless otherwise specified. †Kimina-Prover Preview originally reported 7 solved problems on CombiBench at Pass@16; the value of 8 shown here is our re-evaluation at Pass@32.

		PutnamBench 660 / 340	**CombiBench** 100 / 45	**miniF2F-test** 244 / 197	**FIMO** 149 / 70
Easy Mode	DeepSeek-Prover-V1.5	-	2	122	1
	DeepSeek-Prover-V2 (CoT)	22	-	201	1
	Kimina-Prover Preview	-	8	168	1
	Goedel-Prover-SFT	6	3	141	2
	Goedel-Prover-V2	**43**	**10**	**215**	**4**
Hard Mode	Kimina-Prover Preview	-	$8^{\dagger}$ / -	-	-
	***DAP* w/ Agent** (Ours)	**36 / 19**	9 / 1	201 / 168	**3 / 0**
	***DAP* w/o Agent** (Ours)	32 / 15	**10 / 2**	**204 / 171**	**3 / 0**

Mode performance, suggesting that our Discover and Prove approach effectively reduces Hard Mode problems to Easy Mode problem statements.

6 Discussion

6.1 Ablation Study on Agent Effectiveness

Advanced reasoning LLMs inherently possess significant self-reflective capabilities (Wen et al., 2025). Consequently, explicitly introducing self-verification and self-correction mechanisms might impose an unnecessary reasoning overhead. To investigate when an agentic mode is beneficial, we conducted experiments with the agent mode disabled across the PutnamBench, CombiBench, MiniF2F-Hard, and FIMO-Hard datasets. The results are presented in Table 2.

Disabling the agent's explicit self-verification and self-correction mechanisms resulted in a significant performance degradation on challenging datasets like PutnamBench. Conversely, no performance degradation was observed on lower-difficulty datasets, such as CombiBench and MiniF2F. We hypothesize that this effect stems from these datasets containing a substantial proportion of relatively simple problems (e.g., middle-school textbook problems) that reasoning LLMs can solve with minimal difficulty. In such cases, the explicit inclusion of self-verification might inadvertently cause the model's instruction-following behavior to over-dominate, leading to excessive self-questioning and the introduction of additional noise to the solution.

Table 3: On the subset of Hard Mode problems with definitive (ground-truth) answers, the Discovery Module's performance in solving mathematical problems presented in natural language was evaluated.

Benchmark	**Size**	***DAP* w/ Agent**	***DAP* w/o Agent**
PutnamBench	340	293 (86%)	265 (78%)
CombiBench	45	32 (71%)	29 (64%)
miniF2F-Hard	197	197 (100%)	197 (100%)
FIMO-Hard	70	43 (61%)	38 (54%)

We analyzed the Discovery Module's accuracy on Hard Mode problems with ground-truth answers, which permit direct assessment of natural language responses (Table 3). Even without agentic self-reflection, the model achieves approximately 78% correctness on Putnam problems, with performance on MiniF2F approaching saturation. For lower-difficulty problems, explicit self-verification and self-correction can be superfluous given the already high one-shot accuracy of reasoning LLMs. However, for more challenging benchmarks like PutnamBench and FIMO-Hard, these agentic components remain crucial. To understand these limitations, we manually categorized all Discovery failures on the 45 solution-style CombiBench and 70 solution-style FIMO problems, whose moderate difficulty avoids miniF2F-Hard's saturation and better exposes real weaknesses (Table 4).

FIMO: high difficulty in number theory and algebra. Many FIMO problems remain beyond the

Table 4: Failure-mode analysis of the Discovery Module on FIMO and CombiBench, with and without self-verification (SV). Counts indicate the number of problems exhibiting each failure type.

Error Type	FIMO w/o SV	FIMO w/ SV	Combi w/o SV	Combi w/ SV	Example
Failed to correctly understand the problem	0	0	3	3	`imo_2010_p5`: invented a non-existent operation
Ignored or misused constraints	3	2	2	2	`fimo_2010_number_theory_p1_2`: omitted case analysis
Incorrect mathematical reasoning foundations	2	2	5	3	`fimo_2016_algebra_p3`: induction without base case
Minor arithmetic / computational error	9	4	0	0	`fimo_2012_algebra_p1`: arithmetic mistake
Insufficient depth / beyond model capability	4	2	2	2	`fimo_2013_algebra_p6`: missing preprocessing

model's capabilities; self-verification cannot fully recover correct answers for the hardest instances. Function-equation problems are particularly challenging and often fail even with verification, as the underlying reasoning requires multi-step algebraic manipulation that is difficult to check automatically.

CombiBench: comprehension failures. On CombiBench, we frequently observed that the model did not correctly understand problem statements or even hallucinated conditions that do not appear in the original problem. These comprehension failures are not substantially reduced by self-verification. We hypothesize that certain combinatorial phrasings—though natural to humans—are less accessible to current LLMs and lead to misparsing or misinterpretation.

Self-verification helps but is not a cure-all. For number-theory and algebra problems, self-verification notably reduces minor computational errors ($9 \rightarrow 4$) and some shallow-reasoning failures ($4 \rightarrow 2$), rescuing answers lost due to arithmetic mistakes. For combinatorics, self-verification helps reduce instances of incorrect reasoning foundations ($5 \rightarrow 3$) but does not eliminate comprehension or hallucination errors. Overall, self-verification is most effective when the error is a localized mistake (e.g., an arithmetic slip) rather than a fundamental misunderstanding.

Ablating self-verification iteration counts shows that 10 iterations approach saturation and 30 (our default) add only marginal gains; see Appendix F.

6.2 Ablation on Rewriting Strategies

A central design choice in *DAP* is the two-stage rewriting pipeline: the Discovery Module first derives the answer in natural language, and only then is the Hard Mode statement transformed into an Easy Mode statement for the ATP prover. To justify this design, we compare three alternative strategies. **No Rewriting:** The Hard Mode problem statement (containing two `sorry` placeholders) is fed directly to the ATP prover without any rewriting. **Straight Rewriting:** The Discovery Module is discarded; instead, an LLM is asked to simultaneously discover the answer *and* produce the rewritten Easy Mode statement in a single step. **Proposed Rewriting (Ours):** The Discovery Module first finds the answer; the Rewriting stage then transforms the statement using that answer.

Table 5: Pass@32 comparison of three rewriting strategies. No Rewriting and Straight Rewriting results were manually verified to exclude spurious proofs.

Benchmark	No	Straight	Ours
PutnamBench (660)	23	19	**36**
CombiBench (100)	**9**	**9**	**9**
miniF2F-Hard (244)	57	38	**201**
FIMO-Hard (149)	**3**	**3**	**3**

Table 5 shows the results, from which we draw three key observations.

No Rewriting is largely ineffective on Hard Mode problems. We attribute this to models' lack of training on Hard Mode ATP tasks, producing large out-of-distribution performance drops when asked to both discover answers and prove them directly in the formal system.

Straight Rewriting performs no better — and sometimes worse. Requiring the model to handle answer discovery and formalization simultaneously in a single step causes it to frequently fail at both. The joint burden of informal mathematical reasoning and syntactically precise Lean code

generation leads to degraded performance on most benchmarks.

Spurious proofs explain the remaining gap. We frequently observed *spurious proofs* (cheating behavior) in both No Rewriting and Straight Rewriting settings. Because the model can see the full Lean statement — including the `abbrev solution` definition — during solving, it sometimes avoids genuine mathematical reasoning by copying problem conditions directly into the proof. A concrete example from `fimo_2009_algebra_p3` is shown in Figure 4 (Appendix G). *DAP*'s decoupled design prevents this shortcut: the prover only ever receives the rewritten Easy Mode statement, which does not expose the answer placeholder, eliminating the opportunity for such cheating behavior.

DAP's modular design allows the Discovery Module and the Proving Module to be replaced independently; experiments with both lightweight small open-source models and the stronger Aristotle API confirm that the pipeline remains functional across resource constraints and that stronger informal reasoning models yield further gains (see Appendix H).

6.3 Natural Language Reasoning or Formal Language Reasoning?

The choice between natural-language and formal-language reasoning presents a critical dilemma for AI's advancement in complex problem-solving. Recent innovations in natural-language reasoning—such as Long Chain-of-Thought and Reinforcement Learning from Human Feedback (RLHF), exemplified by OpenAI's o1 and DeepSeek's R1—have significantly enhanced LLMs' capabilities. Research has shown that natural-language representations can exhibit very high empirical ceilings; performance on challenging datasets like AIME has saturated rapidly (Jaech et al., 2024; Wen et al., 2025). At IMO 2025, the majority of AI systems transitioned from formal to natural-language representations and successfully solved creative reasoning problems, with several solutions attaining gold-medal recognition. In contrast, formal-language systems like Seed Prover (Chen et al., 2025) and Aristotle (Achim et al., 2025) achieved medal-worthy results through extensive computational search, with some solutions completed only after official submission deadlines.

This study investigates Hard Mode ATP as a neutral benchmark for comparing natural-language and formal-language reasoning. We observe a similar divergence: state-of-the-art LLMs with self-reflection exceed 80% accuracy on PutnamBench, while formal-language systems achieve less than 10

Despite the robust empirical performance of natural-language reasoning, formal mathematical language will remain essential for future AI systems in mathematical reasoning. Natural-language proofs are prone to subtle errors that are challenging to identify without formal verification, significantly limiting their practical utility for critical applications such as education and advanced mathematics research. As evidenced by results on datasets like Putnam and FIMO, formal mathematical reasoning still requires considerable development. We therefore propose that a promising avenue for future research involves integrating the strengths of both natural-language and formal-language approaches to bridge the existing performance disparity.

7 Conclusion

This study identifies semantic accuracy, interpretability, and transparency problems in current Easy Mode automated theorem-proving settings. We argue that formal statements in Hard Mode, manually annotated by human experts with a unified convention, more accurately reflect the problem-solving requirements faced by participants in mathematical competitions.

To address the scarcity of Hard Mode ATP benchmarks, we engaged Lean 4 experts to reannotate high-quality datasets into the Hard Mode format, producing MiniF2F-Hard and FIMO-Hard. In parallel, we rectified existing semantic misalignments and enforced a unified convention.

To address Hard Mode problems, we introduce the *DAP* framework, designed to offer a rigorous evaluation of AI systems' capacity to solve complex formal mathematical problems. The framework achieves state-of-the-art performance on PutnamBench and CombiBench, representing the first reported progress on PutnamBench's Hard Mode configuration. These results demonstrate the efficacy of combining natural-language reasoning to derive answers with formal methods to prove them.

Limitations

The primary limitation of this work concerns potential dataset contamination, which may have led to inflated estimates of reported AI performance. The miniF2F and FIMO datasets annotated in this study, together with their source problems in natural language (like IMO shortlisted problems) and other related datasets, are publicly accessible on the Internet. The two main LLMs employed in this work—GPT-OSS-120B and Goedel-Prover-V2—have released model weights but not their training corpora; consequently, we cannot definitively determine whether these datasets were included in training data.

By demonstrating a performance gap between the Discovery Module and the Proving Module, this study identifies the bottleneck for Hard Mode ATP as formal reasoning rather than natural-language reasoning. This suggests that approaches leveraging natural-language reasoning to assist formal-language reasoning, such as DSP (Jiang et al., 2023), show promise. In our work, however, the natural-language output passed to the Proving Module is limited to answer candidates. The integration between formal-language and natural-language reasoning therefore remains limited. Ideally, the two modules would interact more tightly, with one modality providing assistance when the other encounters difficulty. Developing such tighter, cooperative integration is left to future work.

Acknowledgments

This paper is partially supported by grants from the National Key Research and Development Program of China with Grant No. 2023YFC3341203.

References

Tudor Achim, Alex Best, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leister, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, and 1 others. 2025. Aristotle: Imo-level automated theorem proving. *arXiv preprint arXiv:2510.01346*.

Team AlphaProof and Team AlphaGeometry. 2024. Ai achieves silver-medal standard solving international 178 mathematical olympiad problems. *DeepMind blog*, 179:45.

Kaito Baba, Chaoran Liu, Shuhei Kurita, and Akiyoshi Sannai. 2025. Prover agent: An agent-based framework for formal mathematical proofs. *arXiv preprint arXiv:2506.19923*.

Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, HUI XUE, and Fan Yang. 2025. Reviving dsp for advanced theorem proving in the era of reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, and 1 others. 2025. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Marco Dos Santos, Hugues de Saxcé, Haiming Wang, Mantas Baksys, Mert Unsal, Junqi Liu, Zhengying Liu, and Jia LI. 2025. Kimina lean server: A high-performance lean server for large-scale verification. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand. Association for Computational Linguistics.

Yichen Huang and Lin F Yang. 2025. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*.

Huawei-xiaoyi. 2025. Huawei-xiaoyi/IMO2025-solutions.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar,

Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

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

Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. 2024. Leanagent: Lifelong learning for formal theorem proving. In *The Thirteenth International Conference on Learning Representations*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025a. Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving. *Preprint*, arXiv:2502.07640.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025b. Goedel-Prover-V2: Scaling Formal Theorem Proving with Scaffolded Data Synthesis and Self-Correction. *Preprint*, arXiv:2508.03613.

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, Ming Zhang, and Qun Liu. 2023. FIMO: A Challenge Formal Dataset for Automated Theorem Proving. *Preprint*, arXiv:2309.04295.

Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, Zhengfeng Yang, Jujian Zhang, Lihong Zhi, Jia Li, and Zhengying Liu. 2025. CombiBench: Benchmarking LLM Capability for Combinatorial Mathematics. *Preprint*, arXiv:2505.03171.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. 2024. Rethinking and Improving Autoformalization: Towards a Faithful Metric and a Dependency Retrieval-based Approach. In *The Thirteenth International Conference on Learning Representations*.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *Preprint*, arXiv:2503.21460.

Thang Luong, Edward Lockhart, and 1 others. 2025. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad. *DeepMind Blog*.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pages 625–635. Springer.

Joseph Myers. 2025. Jsm28/IMOLean: Suggested conventions and examples for Lean formalization of IMO problem statements.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

Tobias Nipkow, Markus Wenzel, Lawrence C. Paulson, Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen, editors. 2002. *Isabelle/HOL*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. 2025. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *Preprint*, arXiv:2504.21801.

Gian Sanjaya. 2025. Mortarsanjaya/IMOSLLean4.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. 2023. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. 2024. Putnam-Bench: Evaluating Neural Theorem-Provers on the Putnam Mathematical Competition. *Preprint*, arXiv:2407.11214.

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, and 21 others. 2025a. Kimina-Prover Preview: Towards Large Formal Reasoning Models with Reinforcement Learning. *Preprint*, arXiv:2504.11354.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2023. DT-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12632–12646, Toronto, Canada. Association for Computational Linguistics.

Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. 2025b. Ma-lot: Multi-agent lean-based long chain-of-thought reasoning enhances formal theorem proving. *arXiv e-prints*, pages arXiv–2503.

Alexander Wei. 2025. Aw31/openai-imo-2025-proofs.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. 2025. Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs. *Preprint*, arXiv:2506.14245.

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.

Jiaxuan Xie, Chengwu Liu, Ye Yuan, Siqi Li, Zhiping Xiao, and Ming Zhang. 2025. Fmc: Formalization of natural language mathematical competition problems. In *2nd AI for Math Workshop@ ICML 2025*.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2024. DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search. *Preprint*, arXiv:2408.08152.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. 2025. BFS-Prover: Scalable Best-First Tree Search for LLM-based Automatic Theorem Proving. *Preprint*, arXiv:2502.03438.

Jing Xiong, Jianhao Shen, Ye Yuan, Haiming Wang, Yichun Yin, Zhengying Liu, Lin Li, Zhijiang Guo, Qingxing Cao, Yinya Huang, Chuanyang Zheng, Xiaodan Liang, Ming Zhang, and Qun Liu. 2023. TRIGO: Benchmarking formal mathematical proof reduction for generative language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11594–11632, Singapore. Association for Computational Linguistics.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2024. Formal Mathematical Reasoning: A New Frontier in AI. *Preprint*, arXiv:2412.16075.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Advances in Neural Information Processing Systems*, 37:105848–105863.

Roozbeh Yousefzadeh and Xuenan Cao. 2024. A lean dataset for international math olympiad: Small steps towards writing math proofs for hard problems. *Transactions on Machine Learning Research*.

Roozbeh Yousefzadeh and Xuenan Cao. 2025. Advocate for Complete Benchmarks for Formal Reasoning with Formal/Informal Statements and Formal/Informal Proofs. *Preprint*, arXiv:2507.04719.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, and 1 others. 2025. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*.

Ye Yuan, Chengwu Liu, Jingyang Yuan, Gongbo Sun, Siqi Li, and Ming Zhang. 2024. A hybrid rag system with comprehensive enhancement on complex reasoning. *Preprint*, arXiv:2408.05141.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*.

A Agentic Frameworks for Formal Theorem Proving

Beyond pure proof-search approaches, several agentic frameworks have been explored for Lean-based ATP. COPRA (Thakur et al., 2023) converts a general-purpose LLM into a Lean proof specialist via a language-agent loop. LeanAgent (Kumarappan et al., 2024) studies lifelong learning to continuously improve LLM performance on advanced mathematics. ProverAgent (Baba et al., 2025) leverages non-formal models to propose auxiliary lemmas that guide formal proofs. MA-LoT (Wang et al., 2025b) applies a multi-agent, Lean-based long chain-of-thought approach with iterative proof repair via Lean compiler feedback. None of these systems are directly compared to *DAP* experimentally, as they target standard (Easy Mode) ATP tasks and their setups are not directly comparable.

B Comparison between DAP and DSP/DSP+

DAP differs from DSP/DSP+ in three key respects. First, **target task**: DSP/DSP+ address standard ATP where the full formal statement is already given; DAP targets Hard Mode ATP, where the system must first discover the answer before proving. Second, **inter-module communication**: DSP passes intermediate reasoning steps (draft and proof sketch) to the formal prover; DAP passes only the final answer to the rewriting stage. Passing only the answer avoids reliance on particular formal-language idioms and sidesteps the problem that natural-language solution steps are often awkward or counterproductive when injected directly into tactic-based Lean proofs (Liu et al., 2023). Additionally, DSP's design is tightly coupled to Isabelle's subgoal syntax, making it difficult to port to Lean, while DSP+'s sketch steps require single equations expressible as Lean `have` statements, which limits applicability. Third, **verification design**: DAP front-loads self-verification inside the Discovery Module before passing to the prover; DSP includes no self-verification, and DSP+ performs repair only when the formal sketch has a syntactic error.

C Prompts used in Discovery Module

C.1 Prompt for Solution Generation

```
### Core Instructions ###

*   **Rigor is Paramount:** Your primary goal is to produce a complete and rigorously justified solution. Every step in your solution must be logically sound and clearly explained. A correct final answer derived from flawed or incomplete reasoning is considered a failure.
*   **Honesty About Completeness:** If you cannot find a complete solution, you must **not** guess or create a solution that appears correct but contains hidden flaws or justification gaps. Instead, you should present only significant partial results that you can rigorously prove. A partial result is considered significant if it represents a substantial advancement toward a full solution. Examples include:
    *   Proving a key lemma.
    *   Fully resolving one or more cases within a logically sound case-based proof.
    *   Establishing a critical property of the mathematical objects in the problem.
    *   For an optimization problem, proving an upper or lower bound without proving that this bound is achievable.
*   **Use TeX for All Mathematics:** All mathematical variables, expressions, and relations must be enclosed in TeX delimiters (e.g., `Let $n$ be an integer.`).

### Output Format ###

Your response MUST be structured into the following sections, in this exact order.

**1. Summary**

Provide a concise overview of your findings. This section must contain two parts:

*   **a. Verdict:** State clearly whether you have found a complete solution or a partial solution.
    *   **For a complete solution:** State the final answer, e.g., "I have successfully solved the problem. The final answer is..."
    *   **For a partial solution:** State the main rigorous conclusion(s) you were able to prove, e.g., "I have not found a complete solution, but I have rigorously proven that..."
*   **b. Method Sketch:** Present a high-level, conceptual outline of your solution. This sketch should allow an expert to understand the logical flow of your argument without reading the full detail. It should include:
    *   A narrative of your overall strategy.
    *   The full and precise mathematical statements of any key lemmas or major intermediate results.
    *   If applicable, describe any key constructions or case splits that form the backbone of your argument.

**2. Detailed Solution**

Present the full, step-by-step mathematical proof. Each step must be logically justified and clearly explained. The level of detail should be sufficient for an expert to verify the correctness of your reasoning without needing to fill in any gaps. This section must contain ONLY the complete, rigorous proof, free of any internal commentary, alternative approaches, or failed attempts.

### Self-Correction Instruction ###

Before finalizing your output, carefully review your "Method Sketch" and "Detailed Solution" to ensure they are clean, rigorous, and strictly adhere to all instructions provided above. Verify that every statement contributes directly to the final, coherent mathematical argument.
```

C.2 Prompt for Self-Verification

```
Below is the bug report. If you agree with certain item in it, can you improve your solution so that it is complete and rigorous? Note that the evaluator who generates the bug report can misunderstand your solution and thus make mistakes. If you do not agree with certain item in the bug report, please add some detailed explanations to avoid such misunderstanding. Your new solution should strictly follow the instructions in the system prompt.
```

C.3 Prompt for Self-Correction

```
You are an expert AI assistant specializing in Formal Mathematics with the Lean 4 proof assistant. Your task is to perform a specific and targeted code modification.

You will be given three inputs:
```

````
1.  A natural language math problem
↪  (`<NATURAL_LANGUAGE_PROBLEM>`).
2.  A detailed natural language solution to that problem
↪  (`<NATURAL_LANGUAGE_SOLUTION>`).
3.  A Lean 4 code statement that formalizes the problem
↪  (`<LEAN4_STATEMENT>`).

The Lean 4 statement contains a "fill-in-the-blank"
↪  definition for the solution, typically an `abbrev` or
↪  `def` with `sorry` as its value.

**Your task is to:**

1.  **Read the `<NATURAL_LANGUAGE_SOLUTION>` to find the
↪  final answer** to the problem. The answer is usually
↪  explicitly stated at the end (e.g., "The value is π/2",
↪  "The result is 42").
2.  **Locate the `abbrev` or `def` line** in the
↪  `<LEAN4_STATEMENT>` that defines the solution variable
↪  (e.g., `abbrev my_solution : ℝ := sorry`).
3.  **Replace the `sorry`** in that definition with the
↪  final answer you extracted. Ensure the syntax is
↪  correct for Lean 4 (e.g., `π` becomes `Real.pi`).
4.  **Output the entire Lean 4 code block with this single
↪  modification.** Do not change any other part of the
↪  code. Specifically, the `theorem`'s proof, which is
↪  also `sorry`, must remain unchanged.

**Example:**

**<NATURAL_LANGUAGE_PROBLEM>:**
Evaluate the integral \[\int_0^1 \frac{1}{1+x^2} \,dx.\]

**<NATURAL_LANGUAGE_SOLUTION>:**
The problem is to evaluate the definite integral of \(f(x)
↪  = \frac{1}{1+x^2}\) from \(0\) to \(1\).

The antiderivative of \(\frac{1}{1+x^2}\) is
↪  \(\arctan(x)\).
Using the Fundamental Theorem of Calculus, we have:
\[
\int_0^1 \frac{1}{1+x^2} \,dx = [\arctan(x)]_0^1
\]
\[
= \arctan(1) - \arctan(0)
\]
Since \(\arctan(1) = \pi/4\) and \(\arctan(0) = 0\), the
↪  value of the integral is:
\[
\frac{\pi}{4} - 0 = \frac{\pi}{4}.
\]
The final answer is \(\pi/4\).

**<LEAN4_STATEMENT>:**
```lean
import
↪  Mathlib.Analysis.SpecialFunctions.Trigonometric.Arctan

open Set

abbrev integral_value : ℝ := sorry

theorem integral_example
  : ∫ x in Icc 0 1, 1 / (1 + x^2) = integral_value :=
sorry
```

**Expected Output:**
```lean
import
↪  Mathlib.Analysis.SpecialFunctions.Trigonometric.Arctan

open Set

abbrev integral_value : ℝ := Real.pi / 4

theorem integral_example
  : ∫ x in Icc 0 1, 1 / (1 + x^2) = integral_value :=
sorry
```

---

Now, perform the task for the following inputs.

**<NATURAL_LANGUAGE_PROBLEM>:**
{{NATURAL_LANGUAGE_PROBLEM}}

**<NATURAL_LANGUAGE_SOLUTION>:**
{{NATURAL_LANGUAGE_SOLUTION}}

**<LEAN4_STATEMENT>:**
```lean
{{LEAN4_STATEMENT}}
```
````

C.4 Prompt for Rewriting

```
You are an expert mathematician and a meticulous grader for
↪  an International Mathematical Olympiad (IMO) level
↪  exam. Your primary task is to rigorously verify the
↪  provided mathematical solution. A solution is to be
↪  judged correct **only if every step is rigorously
↪  justified.** A solution that arrives at a correct
↪  final answer through flawed reasoning, educated
↪  guesses, or with gaps in its arguments must be flagged
↪  as incorrect or incomplete.

### Instructions ###

**1. Core Instructions**
*   Your sole task is to find and report all issues in the
↪  provided solution. You must act as a **verifier**, NOT
↪  a solver. **Do NOT attempt to correct the errors or
↪  fill the gaps you find.**
*   You must perform a **step-by-step** check of the
↪  entire solution. This analysis will be presented in a
↪  **Detailed Verification Log**, where you justify your
↪  assessment of each step: for correct steps, a brief
↪  justification suffices; for steps with errors or gaps,
↪  you must provide a detailed explanation.

**2. How to Handle Issues in the Solution**
When you identify an issue in a step, you MUST first
↪  classify it into one of the following two categories
↪  and then follow the specified procedure.

*   **a. Critical Error:**
    This is any error that breaks the logical chain of the
    ↪  proof. This includes both **logical fallacies**
    ↪  (e.g., claiming that `A>B, C>D` implies `A-C>B-D`)
    ↪  and **factual errors** (e.g., a calculation error
    ↪  like `2+3=6`).
    *   **Procedure:**
        *   Explain the specific error and state that it
        ↪  **invalidates the current line of reasoning**.
        *   Do NOT check any further steps that rely on
        ↪  this error.
        *   You MUST, however, scan the rest of the
        ↪  solution to identify and verify any fully
        ↪  independent parts. For example, if a proof is
        ↪  split into multiple cases, an error in one
        ↪  case does not prevent you from checking the
        ↪  other cases.

*   **b. Justification Gap:**
    This is for steps where the conclusion may be correct,
    ↪  but the provided argument is incomplete, hand-wavy,
    ↪  or lacks sufficient rigor.
    *   **Procedure:**
        *   Explain the gap in the justification.
        *   State that you will **assume the step's
        ↪  conclusion is true** for the sake of argument.
        *   Then, proceed to verify all subsequent steps
        ↪  to check if the remainder of the argument is
        ↪  sound.

**3. Output Format**
Your response MUST be structured into two main sections: a
↪  **Summary** followed by the **Detailed Verification
↪  Log**.

*   **a. Summary**
    This section MUST be at the very beginning of your
    ↪  response. It must contain two components:
    *   **Final Verdict**: A single, clear sentence
    ↪  declaring the overall validity of the solution. For
    ↪  example: "The solution is correct," "The solution
    ↪  contains a Critical Error and is therefore
    ↪  invalid," or "The solution's approach is viable
    ↪  but contains several Justification Gaps."
    *   **List of Findings**: A bulleted list that
    ↪  summarizes **every** issue you discovered. For
    ↪  each finding, you must provide:
        *   **Location:** A direct quote of the key phrase
        ↪  or equation where the issue occurs.
        *   **Issue:** A brief description of the problem
        ↪  and its classification (**Critical Error** or
        ↪  **Justification Gap**).

*   **b. Detailed Verification Log**
    Following the summary, provide the full, step-by-step
    ↪  verification log as defined in the Core
    ↪  Instructions. When you refer to a specific part of
    ↪  the solution, **quote the relevant text** to make
    ↪  your reference clear before providing your
    ↪  detailed analysis of that part.

**Example of the Required Summary Format**
*This is a generic example to illustrate the required
↪  format. Your findings must be based on the actual
↪  solution provided below.*

**Final Verdict:** The solution is **invalid** because it
↪  contains a Critical Error.

**List of Findings:**
```

```
*   **Location:** "By interchanging the limit and the
↪  integral, we get..."
    *   **Issue:** Justification Gap - The solution
    ↪  interchanges a limit and an integral without
    ↪  providing justification, such as proving uniform
    ↪  convergence.
*   **Location:** "From $A > B$ and $C > D$, it follows
↪  that $A-C > B-D$"
    *   **Issue:** Critical Error - This step is a logical
    ↪  fallacy. Subtracting inequalities in this manner
    ↪  is not a valid mathematical operation.
```

C.5 Example of Self-Verification Error Report

The Self-Verification step prompts the LLM to carefully review the generated solution step by step and produce a structured error report. Each finding identifies a specific location in the solution, describes the issue, and classifies its severity. The following is a representative example of such an error report, produced by the Discovery Module on a functional-equation problem.

```
Final Verdict: The solution contains several Justification
↪  Gaps and therefore is not fully rigorous.

List of Findings:
* Location: Lemma 1 proof -- "h(x0) is an upper bound of
↪  the set {y | y < x0} ..."
  Issue: Justification Gap -- the claim that h(x0) bounds
  ↪  all predecessors of x0 is not proved; the argument
  ↪  that this contradicts the supremum property is
  ↪  invalid.
* Location: Lemma 2 proof -- "Both lines are proved by a
↪  simple induction on the natural variable."
  Issue: Justification Gap -- the induction for the second
  ↪  coordinate is only sketched; it is not shown that
  ↪  the composed beta-functions satisfy the required
  ↪  monotonicity and domination properties.
* Location: Lemma 2 proof -- base case "For n=0 ..." while
↪  N was earlier taken to start at 1.
  Issue: Minor Justification Gap -- the indexing mismatch
  ↪  is not addressed, though it does not affect the
  ↪  overall argument.
```

D Dataset Quality Fixes

Porting FIMO to Lean 4: The publicly available FIMO dataset is written in Lean 3. Although a largely automated migration tool exists, the syntactic and library-level changes between Lean 3 and Lean 4 mean that many statements do not port cleanly: tactic names changed, Mathlib APIs were reorganised, and some constructs require manual rewriting to compile while remaining semantically faithful. Our annotators ported every FIMO problem to Lean 4, verifying that each statement compiles and is semantically faithful to the original.

Fixing Semantic Misalignments: Following the annotation principles described in §4.1, annotators identified and repaired formalization errors present in the source datasets. We found approximately 15 misalignments in miniF2F and 20 in FIMO. Table 6 summarises the four most common error types.

Rephrasing for Hard Mode Compatibility: Our annotation principle requires that any quantity a human competitor must derive must not be hard-coded in the formal statement. In some cases this meant non-trivially rephrasing a problem so that the unknown value becomes a free parameter and proper side-conditions (simplicity, squarefreeness, positivity, etc.) are added explicitly. A representative before/after example is shown in Figure 3 (Appendix E).

E Hard Mode Annotation Example

Figure 3 shows a representative before/after example of rephrasing a problem for Hard Mode compatibility. The original formalization hard-codes `c = 2`, leaking part of the answer; the rephrased version promotes `c` to a free parameter and adds explicit simplicity and squarefreeness conditions, requiring the solver to determine the canonical form independently.

F Self-Verification Iteration Ablation

Table 7 quantifies the effect of the maximum number of self-verification iterations on Discovery Module accuracy.

Two observations emerge. First, on miniF2F-Hard the Discovery Module achieves perfect accuracy even without self-verification, consistent with the saturation effect (though possible dataset contamination should be noted). Second, for harder benchmarks like PutnamBench and FIMO-Hard, self-verification provides substantial gains; 10 iterations approach saturation and balance computational cost with accuracy, while 30 iterations adds marginal gains and serves as our default.

G Spurious Proof Example

Figure 4 shows a concrete example of a spurious proof observed under the No Rewriting setting. The proof closes by `rfl` because the `abbrev solution` was defined to be literally the same set as the one appearing in the theorem statement — the prover never needed to reason about the underlying mathematics.

H Cross-Model Pairing

A practical advantage of *DAP*'s modular design is that the Discovery Module and the Proving Module can be replaced independently. To demonstrate this flexibility, Table 8 reports results for three model combinations: our default pairing, a lightweight pairing using smaller open-source models, and a high-resource pairing using a

Table 6: Semantic misalignment types found and corrected in miniF2F and FIMO during our re-annotation.

Misalignment Type	# in miniF2F	# in FIMO	Example
Ignoring whether an extremum is attainable	8	8	`mathd_numbertheory_495`, `fimo_2010_number_theory_p1_1`
Adding extra conditions in the formalization	2	1	`mathd_algebra_320`, `fimo_2017_number_theory_p8`
Incomplete proof goals	2	2	`amc12b_2021_p3`, `fimo_2008_algebra_p1`
Missing/incomplete NL problem statements	2	2	`mathd_algebra_188`, `fimo_2008_algebra_p3_1`

Table 7: Discovery Module accuracy (number of correctly solved problems) under varying maximum self-verification iteration budgets. Column headers indicate the iteration limit; 0 means no self-verification. †Our default setting. All results are Pass@32.

	Max SV Iterations			
Dataset	**0**	**5**	**10**	**30†**
PutnamBench (340)	265	291	292	**293**
CombiBench (45)	29	31	**32**	**32**
miniF2F-Hard (194)	**194**	**194**	**194**	**194**
FIMO-Hard (70)	38	40	**43**	**43**

stronger closed-source model. Because the Aristotle API (Achim et al., 2025) does not support concurrent requests and each problem can take hours, we sampled five problems per dataset for that condition; results are reported as solved/total.

Table 8: Pass@32 results for different Discovery–Proving model pairings. †Aristotle API results are sampled (5 problems per dataset).

Configuration	Putnam (660)	Combi (100)	mF2F (244)	FIMO (149)
GPT-OSS 120B + Goedel-V2 32B (Ours)	**36**	**9**	**201**	**3**
Qwen3 8B + DS-Prover-V1.5 7B	5	2	107	2
GPT-5 (Thinking) + Aristotle API†	3/5	3/5	5/5	2/5

Two conclusions follow. First, the pipeline is functional even with small, resource-efficient models, supporting use in compute-constrained settings. Second, replacing the Discovery Module with a significantly stronger model yields meaningful further gains, illustrating the headroom available as informal reasoning models continue to improve.

Natural-language problem: Let x be a positive number such that $2x^2 = 4x + 9$. If x can be written in simplified form as $\dfrac{a+\sqrt{b}}{c}$ where a, b, and c are positive integers, what is $a+b+c$?

Original formalization (problematic) — `c = 2` is hard-coded:

```
theorem mathd_algebra_320
    (x : ℝ) (a b c : ℕ)
    (h0 : 0 < a ∧ 0 < b ∧ 0 < c ∧ 0 ≤ x)
    (h1 : 2 * x ^ 2 = 4 * x + 9)
    (h2 : x = (a + Real.sqrt b) / c)
    (h3 : c = 2) : a + b + c = 26 := by
```

Rephrased formalization (aligned) — `c` is a free parameter:

```
abbrev mathd_algebra_320_solution : ℕ := sorry
-- 26

theorem mathd_algebra_320
    (x : ℝ) (a b c : ℕ)
    (h_x_pos : 0 < x)
    (h_eqn : 2 * x ^ 2 = 4 * x + 9)
    (h_form : x = (a + Real.sqrt b) / c)
    (h_abc_pos : a > 0 ∧ b > 0 ∧ c > 0)
    (h_simplified_gcd : Nat.gcd a c = 1)
    (h_simplified_sq_free : Squarefree b)
    : a + b + c = mathd_algebra_320_solution := by
      sorry
```

Figure 3: Rephrasing `mathd_algebra_320` for Hard Mode compatibility. The original formalization hard-codes `c = 2`, leaking part of the answer; the rephrased version promotes `c` to a free parameter and adds explicit simplicity and squarefreeness conditions, requiring the solver to determine the canonical form independently.

```
abbrev fimo_2009_algebra_p3_solution : Set (ℕ+ → ℕ+) :=
  { f : ℕ+ → ℕ+ | (∀ x y,
      x + f y > f (y + f x - 1) ∧
      x + f (y + f x - 1) > f y ∧
      f y + f (y + f x - 1) > x) }

theorem fimo_2009_algebra_p3 :
  { f : ℕ+ → ℕ+ | (∀ x y,
      x + f y > f (y + f x - 1) ∧
      x + f (y + f x - 1) > f y ∧
      f y + f (y + f x - 1) > x) } =
    fimo_2009_algebra_p3_solution := by
  have h_main : { f : ℕ+ → ℕ+ | (∀ x y,
      x + f y > f (y + f x - 1) ∧
      x + f (y + f x - 1) > f y ∧
      f y + f (y + f x - 1) > x) } =
    fimo_2009_algebra_p3_solution := by rfl
  exact h_main
```

Figure 4: A spurious proof of `fimo_2009_algebra_p3` observed under the No Rewriting setting. The `abbrev solution` is defined as the same set that appears in the theorem statement, so the prover closes the goal with `rfl` without performing any mathematical reasoning.