Ax-Prover: A Deep Reasoning Agentic Framework for Theorem Proving in Mathematics and Quantum Physics

Marco Del Tredici[1], Jacob McCarran[1], Benjamin Breen[1], Javier Aspuru Mijares[1], Weichen Winston Yin[1], Jacob M. Taylor[1], Frank Koppens[2], and Dirk Englund[1,3]

[1]Axiomatic_AI
[2]Institut de Ciències Fotòniques (ICFO)
[3]Massachusetts Institute of Technology (MIT)

October 17, 2025

Abstract

We present Ax-Prover, a multi-agent system for automated theorem proving in Lean that can solve problems across diverse scientific domains and operate either autonomously or collaboratively with human experts. To achieve this, Ax-Prover approaches scientific problem solving through formal proof generation, a process that demands both creative reasoning and strict syntactic rigor. Ax-Prover meets this challenge by equipping Large Language Models (LLMs), which provide knowledge and reasoning, with Lean tools via the Model Context Protocol (MCP), which ensure formal correctness. To evaluate its performance as an autonomous prover, we benchmark our approach against frontier LLMs and specialized prover models on two public math benchmarks and on two Lean benchmarks we introduce in the fields of abstract algebra and quantum theory. On public datasets, Ax-Prover is competitive with state-of-the-art provers, while it largely outperform them on the new benchmarks . This shows that, unlike specialized systems that struggle to generalize, our tool-based agentic theorem prover approach offers a generalizable methodology for formal verification across diverse scientific domains. Furthermore, we demonstrate Ax-Prover's assistant capabilities in a practical use case, showing how it enabled an expert mathematician to formalize the proof of a complex cryptography theorem.

1 Introduction

Developing Large Language Models (LLMs) that can reason reliably across scientific domains remains a central challenge for AI, both in academia and in industry. LLM-based formal reasoning systems have mainly been developed for mathematics, where they have achieved outstanding results [16, 13]. Recently, considerable effort has been put into training reasoning LLMs for formal theorem proving using Lean [19], an open-source programming language and interactive proof assistant. Together with its community-driven `Mathlib` library [33], Lean provides a rigorous setting where AI systems must engage with symbolic reasoning and structured formalization, building on an evolving body of mathematical knowledge. LLM provers such as the DeepSeek-Prover series [60, 61, 49], Kimina-Prover-72B [58], Goedel-Prover [35, 36], and Seed-Prover [13] have shown that specialized prover models can be distilled from frontier LLMs and trained for theorem proving in Lean, reaching state-of-the-art performance on math benchmarks like miniF2F [65] and PutnamBench [55]. Despite these results, these models face some key limitations. First, since they were mainly trained and tested in the the domain of mathematics, their ability to generalize beyond this domain remains unclear. Relatedly, they are usually trained on fixed versions of the fast-evolving `Mathlib` library, making them brittle to changes introduced in new `Mathlib` versions – such as new or renamed definitions – unless continuously re-trained which adds significant cost.[1] Second, while training improves their ability to produce Lean proofs, it narrows their capabilities relative to general LLMs as they become unable to use external tools and engage in human–AI collaboration. Finally, it is hard to run them as they require high-spec machines and expertise to be

[1]For example, Deepseek-Prover-V2-671B was released on April 30, 2025 and, in our experiments, we observed that, for example, it uses the lemma `sqrt_eq_iff_sq_eq`, which was deprecated in favor of `sqrt_eq_iff_eq_sq` on March 3, 2025.

successfully deployed and used. Together, these issues suggest that scaling increasingly large specialized provers may yield diminishing returns in both flexibility and usability.

In contrast, general-purpose LLMs like Claude [5] and GPT [45] encode substantial knowledge across diverse domains (e.g., mathematics, physics, and computer science), while also exhibiting strong natural language understanding, problem solving skills, and interaction capabilities. Moreover, they are easily accessible through APIs, making them convenient to deploy and integrate into any workflow. Yet, they are not explicitly trained to formalize statements or construct proofs in Lean, nor can they natively interact with the Lean environment. This creates a sharp division: specialized provers are tightly integrated with Lean but narrow in scope and hard to use, whereas general-purpose LLMs are broad in scope and easily accessible but lack the ability to interface with the formal reasoning infrastructure required for theorem proving.

To address this gap, we introduce **Ax-Prover**,[2] a new agentic workflow for theorem proving in Lean that leverages the Model Context Protocol (MCP) [41] to equip general-purpose LLMs with Lean tools from the `lean-lsp-mcp` repository [22]. Ax-Prover combines the reasoning capabilities of LLMs with the formal verification power of Lean. The LLM analyzes unproven theorems, proposes proof sketches, and generates step-by-step Lean code, while the Lean tools allow the LLM to inspect goals, search for relevant results, locate errors, and verify proofs – capabilities essential for rigorous formal theorem proving.

Ax-Prover overcomes the main limitations of state-of-the-art provers. First, using frontier LLMs prevents domain overspecialization while the MCP interface lets the system work with any recent version of `Mathlib` without needing to be retrained. Second, it preserves tool-use and conversational abilities, enabling interactive collaboration with humans. Third, by leveraging existing frontier models, it sidesteps the need to host or deploy specialized systems.

We evaluated Ax-Prover on two public datasets that include mathematics competition problems (NuminaMath-LEAN [44] and PutnamBench [55]) and introduce two new datasets to enable evaluation in new domains. The first one, **AbstractAlgebra**, focuses on algebraic structures such as groups, rings, and fields, testing the provers' abilities to reason in a more abstract, research-oriented setting rather than the competition-driven style of existing datasets. The second one is **QuantumTheorems**, which represents an initial step toward automated theorem proving in a scientific domain beyond pure mathematics, evaluating the models' formal reasoning in quantum mechanics. Our results show that Ax-Prover has competitive performance on PutnamBench and outperforms general-purpose LLMs not equipped with Lean tools as well as state-of-the-art specialized provers on the other datasets, with a significant margin on the ones we introduce. This gap underscores the potential of Ax-Prover to act as the key AI verification tool for mathematically grounded scientific reasoning.

Our contributions are threefold: (i) We design **Ax-Prover**, a lightweight agentic workflow that connects general-purpose LLMs to Lean tools via MCP, and demonstrate that it performs on par with or surpasses both general-purpose LLMs and specialized provers across several scientific domains; (ii) We introduce new formalized **Lean datasets** covering abstract algebra and quantum physics, complementing existing benchmarks; (iii) We showcase Ax-Prover's capabilities as an assistant through a use case where the system collaborated with an expert mathematician to formally verify the proof of a recent cryptography result.

2 Related Work

Automated theorem proving in Lean has roots in classical approaches such as decision procedures [18, 10] and heuristic-guided proof search [30, 51], but they face challenges, specifically the former does not handle general mathematical domains (e.g. transcendental functions and complex numbers) and the latter do not before well out of distribution. More recent work integrates machine learning: from heuristic tuning [56] to premise selection and tactic prediction [28, 27], culminating in transformer-based language models capable of generating Lean proofs [48, 32, 47, 62]. More recent large-scale systems extend this trend by training LLMs on formal proof though distillation, supervised finetuning and reinforcement learning. Current examples of specialized models are Kimina-Prover [58], the DeepSeek-Prover family [60, 61, 49], Goedel-Prover 1 and 2 [35, 36], Prover Agent [9], Apollo [46], and Seed-Prover [13]. All of these are highly specialized provers, which take input a Lean theorem and autonomously produce a proof. A related line of work is exploring agentic flows that include frontier LLMs and a formal verifier; example include Hilbert [57] and Aristotle [2]. Although we also adopt a similar approach, some crucial differences exist, namely: (i) we give the LLM direct access to Lean tools via MCP; (ii) our framework requires no training [2] and does not rely on any

[2] "Ax" stands for "axiomatic", reflecting the base principles in mathematics and physics, the domains explored in this work.

specialized provers [57]; (iii) we demonstrate the effectiveness of our approach on domains other than mathematics; (iv) we showcase the capabilities of our system as an interactive assistant to human researchers.

Finally, a parallel line of work has explored classical machine learning for supporting experts in theorem proving in Lean, for example, in premise selection and tactic prediction [25, 11], and more recently through LLMs that connect to Lean via external interfaces [7, 8, 53]. These approaches illustrate the promise of AI-assisted proving, but they remain resource-intensive and difficult to adapt across scientific domains. Recent efforts, such as [31], attempt to remedy this by emphasizing greater adaptability within Lean. At the same time, there is growing interest in human–AI collaboration: conversational assistants [17] and "copilot"-style integrations [14] suggest how formal tools can augment, rather than replace, human reasoning. Our work builds on this trajectory by closing the gap between heavyweight, specialized provers and lightweight, researcher-friendly systems that can be more readily adapted to the evolving Lean ecosystem.

3 System Architecture

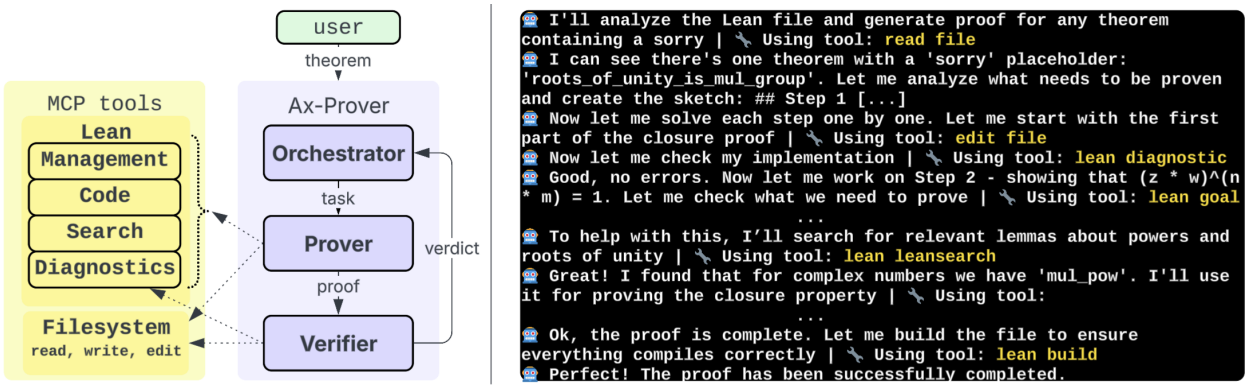

Figure 1: Left: the multi-agent workflow of Ax-Prover. Right: the tool-enhanced reasoning of the Prover.

We implement Ax-Prover as a multi-agent architecture with three agents, each of which is implemented as an LLM with specific prompts: the **Orchestrator**, the **Prover**, and the **Verifier**. Following recent agentic designs for complex tasks such as scientific discovery [26, 63], we avoid a monolithic setup by assigning each specialized agent a distinct role. This separation enables specialization and modularity: agents can be independently optimized, replaced, or extended, allowing researchers to adapt Ax-Prover to their own needs without destabilizing the system.

Figure 1 (left) shows our workflow: the Orchestrator receives an unproven Lean statement and forwards it to the Prover, which iteratively works on the proof by performing reasoning, making calls to MCP Lean tools, and generating Lean code (Figure 1, right). The Verifier then checks the proof and reports back to the Orchestrator. If the proof is complete and no error is found, the Orchestrator ends the task; otherwise, it provides feedback to the Prover, which continues the proving process. Through this closed-loop process, the system incrementally converts unproven theorems into formally verified Lean proofs. Next, we provide details about the agents and tools.

3.1 Specialized Agents

3.1.1 Orchestrator

The Orchestrator's role is analogous to a scheduler in distributed systems: it does not perform computation itself but ensures that computation flows smoothly across agents. It holds three main responsibilities. First, it handles **task assignment**, as it receives user input and instructs the Prover accordingly. Next, it manages **feedback routing** by taking diagnostic outputs from the Verifier and giving structured feedback to the Prover (if errors are found). This separation ensures that proof synthesis and evaluation remain distinct while still enabling iterative refinement. Finally, it decides when to **stop the refinement loop**. Termination occurs either when the Verifier certifies the proof as complete and error-free, or when the number of attempts exceeds a configurable threshold.

3.1.2 Prover

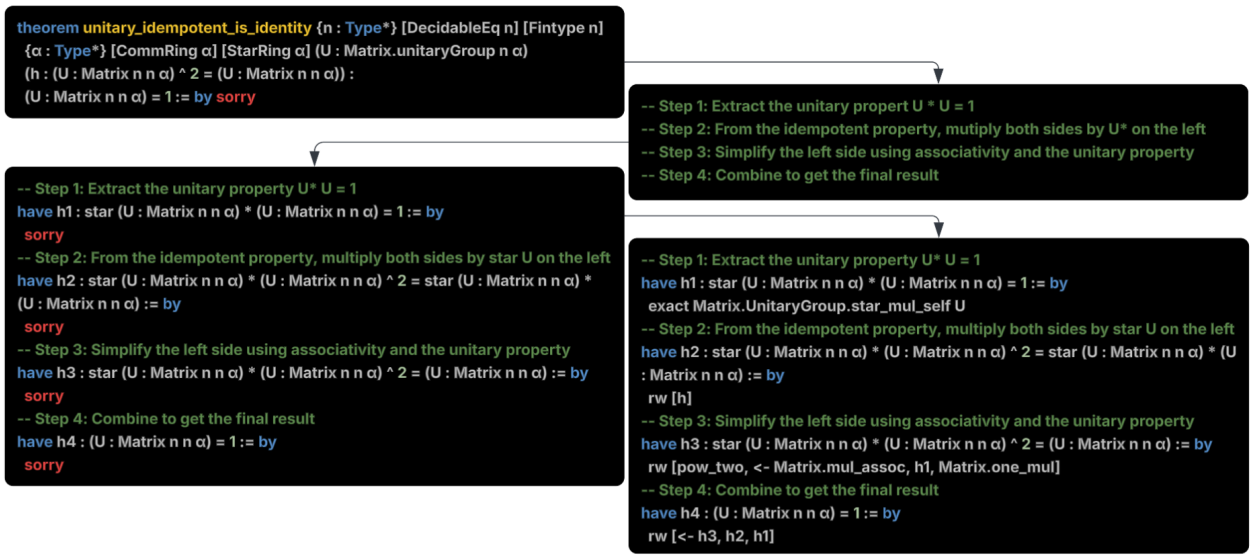

Figure 2: The main steps performed by the Prover to prove a theorem.

The Prover is the constructive core of the system. Its task is to transform unproven Lean theorems into completed proofs. Theorem proving requires both creativity – finding the right lemma or using the right tactic – and rigor – ensuring that the structure and Lean code are syntactically correct. To achieve this, the Prover balances LLM-based heuristic exploration with rigorous Lean formalization aided by the MCP Lean tools made available by the `lean-lsp-mcp` (see Section 3.2).

We instruct the Prover to carry out its task following an incremental, step-by-step approach, and to write each update to the theorem proof to a .lean file. This is for two reasons: first, it satisfies the requirements of the MCP Lean tools, some of which need a `.lean` file path to inspect the code contained within; second, it allows the user to inspect the proof process in real time. In Figure 2, we present Lean code snippets at important stages of this process. Initially, the Prover **identifies target theorems**, by scanning Lean files for unfinished proofs marked with `sorry`, a placeholder tactic indicating an incomplete proof (top-left). Then, it **writes a proof sketch**, a coarse-grained natural language outline of the proof's logical flow which breaks down a complex proof into more manageable steps (top-right). Next is **formalization**, where each step in the sketch is formalized into a Lean statement starting with `have` and ending with `sorry` (bottom-left); this brings into Lean's proof context new statements that are to be proven from earlier hypotheses of the theorem. Then, the Prover goes through each step sequentially, proposing Lean tactics to substitute each `sorry`. After completing each step, the Prover uses a specific Lean tool, `lean_diagnostic_messages` (see Section 3.2) to assess if the generated step is correct. If critical errors are detected or a `sorry` is still present, the Prover attempts to fix the error or correct its reasoning. When all the steps are correctly solved, the Prover **ends its task** (bottom-right).

Tool usage is crucial for the Prover to perform its task. This is clearly illustrated in Figure 1 (right), which is extracted from the LLM log during an experimental run. The figure shows how both exploration and formalization are achieved through tool-enhanced reasoning, where the Prover uses mcp tools to interact with Lean files (`read_file` and `edit_file`); identify goals at various points in the proof (`lean_goal`); search for theorems in `Mathlib` (`lean_leansearch`); and verify the correctness of its proofs (`lean_diagnostic_messages`) This approach allows the Prover to function like an automated yet cautious mathematician: it drafts a plan, incrementally explores and implements ideas, verifies their correctness in Lean, and advances only once each step has been validated.

3.1.3 Verifier

The Verifier serves as the final gatekeeper of correctness in our workflow. It neither generates nor modifies proofs: it only assesses the correctness of the proof generated by the Prover. The Verifier has access to filesystem tools, used to access the file produced by the Prover, as well as a single Lean tool, `lean_diagnostic_messages`, to assess

the correctness of the proof. The Verifier operates in two steps. First, it **compiles** the Lean file produced by the Prover using the `lean_diagnostic_messages` tool, parses the returned diagnostic messages, and generates an error report. Second, it **emits a verdict**: a proof is considered verified if and only if no level-1 error exists (see Section 3.2) and there are no `sorry` (or the equivalent tactic `admit`) present in the file.

At first glance, the Verifier may seem redundant, since it uses the same `lean_diagnostic_messages` tool as the Prover. However, it is needed for two reasons: (i) the Prover may run out of steps (see Section 5.1) and return an incomplete or incorrect proof, and (ii) it sometimes terminates early despite remaining errors. An independent Verifier is thus crucial to ensure robustness, mirroring software pipelines where aggressive testing is always checked by a conservative compiler.

3.2 MCP Tools

As described above, tool use is essential in our approach. We provide the LLM with access to tools via the MCP, a standard interface that lets LLM agents invoke external services in a uniform, controlled way [41]. We implement two categories of tools: **Filesystem tools** and **Lean tools**. Filesystem tools handle file operations such as `read_file`, `write_file`, and `list_directory` (see Appendix A.1). Lean tools allow Ax-Prover to perform a variety of actions crucial for theorem proving. We access these tools through the `lean-lsp-mcp` project [22], which provides a standardized interface to the Lean environment and ensures that Ax-Prover always operates on the latest version of `Mathlib`. This ensures that we can maintain compatibility for imports, theorem references, and proof construction without relying on the LLM's knowledge of the version (or multiple incompatible versions) of `Mathlib` on which it was trained. The Lean tools we use fall into four main groups, summarized in Table 1.

Category	**Tools**
Project and File Management	`lean_build`: Compile and build the Lean project `lean_file_contents`: Get contents of a Lean file `lean_declaration_file`: Find which file contains a declaration
Diagnostics and Feedback	`lean_diagnostic_messages`: Compile code and return diagnostic messages `lean_goal`: Get the current proof goal at a position `lean_term_goal`: Get goal information for a term `lean_hover_info`: Get hover information for symbols
Code Assistance	`lean_completions`: Get completion suggestions `lean_multi_attempt`: Try multiple proof attempts `lean_run_code`: Execute Lean code
Search and Reasoning	`lean_leansearch`: Search for theorems and lemmas `lean_loogle`: Search for lemmas by type signature `lean_state_search`: Search proof states `lean_hammer_premise`: Use automated theorem proving

Table 1: Lean tools available on `lean-lsp-mcp`, organized by functionality.

Note that `lean_diagnostic_messages` returns an error log containing scalars indicating the severity of the errors or messages the tool has returned: 0 if no error is found; 1 for erroneous/incorrect proofs; and 2 for a valid but incomplete proof, e.g. with `sorry`, warning, or linter error. A proof is considered correct and complete only if it compiles, no severity 1 errors are found, and no `sorry` or `admit` is present.

4 Datasets

While the application of LLMs to mathematical verification in Lean is evolving rapidly, the availability of comprehensive datasets remains limited. At present, only a few open-source datasets are available, with some of the most notable being **MiniF2F** [65], **PutnamBench** [55], and **NuminaMath-LEAN** [44]. These benchmarks include hard, high-level math problems from competitions such as the International Mathematical Olympiad (IMO) or the Putnam exam. Other datasets exist, but have clear limitations. For example, the Deepseek-Prover-V1 Train[20] includes 27k LLM-generated statements and proofs, but most of them are very simple, and can be solved in 2–3 line (on average) proofs. Lean Workbook [64] (57k) gathers LLM-generated formalizations of mathematical problems. While it reports

a 93.5% statement-level accuracy after filtering, subsequent analyses note that a nontrivial fraction of examples still suffer from semantic errors and hallucinations [38, 59], which limits its reliability.

Current benchmark datasets focus on mathematics and, even within this domain, are centered narrowly on high school and undergraduate-level competition problems. To enrich the current ecosystem and expand the coverage of Lean datasets, we create two datasets. The first one **AbstractAlgebra** (**AA**), is a Lean 4 dataset of problems drawn from standard abstract algebra textbooks. Unlike existing math benchmarks, which focus on undergraduate level competition-style puzzles, AA targets graduate or research-level mathematics, emphasizing deeper abstract concepts over lengthy step-by-step manipulations. The second dataset is **QuantumTheorems** (**QT**), which covers core topics in foundational quantum mechanics, with problems spanning from density matrices to scaling laws for quantum repeater networks. By bridging theoretical physics with formal verification methods, QT not only offers an unprecedented opportunity to test prover agents on quantum mechanics theorems, but it also represents a key step toward evaluating scientific reasoning models across any scientific discipline grounded in mathematics. In the section below, we provide more information about these two as well as other datasets we used for our experiments.

4.1 AbstractAlgebra

AbstractAlgebra is a curated dataset of 100 Lean problems extracted from the exercises in Dummit & Foote's abstract algebra textbook [23]. The problems were extracted and formalized in Lean using an automatic pipeline (see details and examples in Appendix B.1). The dataset consists of two subsets: 50 easy problems from Chapter 1.1 and 50 intermediate problems from Chapters 1.2–2.5. The two categories thus reflect the increasing level of difficulty of the chapters in the book.

As mentioned above, existing datasets focus on high school to undergraduate-level competition mathematics, which typically involves elementary concepts framed as puzzles that require many reasoning steps. For example, a competition problem may ask to determine all positive integers a, b such that $(a^2 + b^2)/(ab + 1) \in \mathbb{Z}$, which is conceptually elementary but requires a sequence of clever number-theoretic transformations.

In contrast, the AA dataset is aimed toward research-level mathematics, which involves deeper concepts with fewer reasoning steps per exercise. For instance, an AA problem may ask: *Prove that every element $x = sr^i$ in the dihedral group D_n has order 2*. By presenting this kind of problems, AA fills the gap between AI-focused formalization efforts, which largely targets elementary mathematics, and the advanced topics studied by research mathematicians.

Finally, we stress that abstract algebra is foundational to much of mathematics, providing essential tools for research in number theory, geometry, topology, and beyond – indeed, 22 of the 32 primary mathematics categories on arXiv build upon it [1]. It also underpins important domains outside of mathematics, such as cryptography, physics, and chemistry. The broad foundational nature of abstract algebra underscores the importance of developing AI proof systems that perform well on problems in this domain, as this has the potential to accelerate progress across many scientific fields.

4.2 QuantumTheorems

QuantumTheorems includes 134 problems spanning core areas of quantum theory. These problems introduce unique challenges, as they require integrating finite-dimensional linear algebra, complex analysis, and matrix theory with quantum principles such as unitarity, Hermiticity, and measurement postulates. This domain-specific knowledge is absent from existing datasets of formal proofs, making QT a valuable benchmark for testing and advancing formal reasoning in physics. QT was generated through an iterative human-in-the-loop process, combining automated proof synthesis with expert curation (see Appendix B.2 for more details and examples).

We generated problems at two levels of difficulty: basic problems are short (require on average 1–10 line proofs to solve) and often solvable with standard automation tactics (`simp`, `linarith`), e.g., a post-measurement state is an eigenstate of the measurement projector. Intermediate level problems (requiring 10–50 line proofs to solve) are solvable with systematic case analysis and orchestration of rewrite rules, such as proving simultaneous diagonalization of commuting observables.

QT represents a first step toward computer-verified quantum mechanics, addressing the challenge of ensuring correctness in quantum information protocols and algorithms. The dataset has practical importance beyond research: as quantum technologies grow more complex, errors in proofs or hidden assumptions can have serious consequences. For instance, a recent bug in a proof claiming to break lattice-based cryptography – only identified weeks later by

experts – illustrates the risks of unchecked reasoning in high-stakes domains [50, 15]. QT provides a first-of-its-kind resource for developing tools which can help detect these mistakes earlier.

4.3 NuminaMath-LEAN

NuminaMath-LEAN [44] is a very recent (August 2025) large-scale collection of approximately 104,000 competition-level mathematics problems formalized in Lean 4. The dataset is created by the same research group that developed the Kimina-Prover. They derived NuminaMath-LEAN from NuminaMath 1.5 [34], with problems drawn from prestigious contests such as the International Mathematical Olympiad (IMO) and the United States of America Mathematical Olympiad (USAMO).

Each problem includes a formal statement in Lean 4, written either by a human annotator (19.3% of the problems) or by an autoformalizer model (80.7%) [44]. Out of the total problems, 25% were correctly proved by Kimina-Prover during its reinforcement learning (RL) training phase (`Solved-K`), 11% were proved by humans (`Solved-H`), while the remaining 64% do not have any proof (`Unsolved`) [58, 34, 44]. We analyzed problems across the three groups and observed a clear difficulty gradient: `Solved-K` < `Solved-H` < `Unsolved`. This ordering aligns with the fact that `Solved-H` and `Unsolved` problems could not be handled by Kimina-Prover, providing an implicit measure of difficulty. The fact that `Solved-H` proofs are on average longer than those in `Solved-K` (155 vs. 98 lines) also offers quantitative evidence consistent with our qualitative assessment. For our experiments, we randomly sampled 300 problems – 100 each from `Solved-K`, `Solved-H`, and `Unsolved` – to create a balanced, representative, and more budget-friendly benchmark.

4.4 PutnamBench

PutnamBench [55] is a multi-language benchmark designed to evaluate the ability of neural theorem provers to solve undergraduate-level competition mathematics problems. It includes formalizations of problems from the William Lowell Putnam Mathematical Competition (1962–2024) across three major proof assistants – Lean, Isabelle, and Rocq. The Lean subset contains 660 formalized problems, which we focus on in this work. The problems span a broad range of undergraduate topics, including Algebra, Analysis, Number Theory, Geometry, Linear Algebra, Combinatorics, Abstract Algebra, Probability, and Set Theory.

Unlike MiniF2F, which is now saturated, PutnamBench remains a challenging benchmark for most provers. Moreover, since it is widely adopted by many models, it serves as a high-value testbed for evaluating our approach against the best theorem-proving models currently available.

5 Experiments

In this section, we provide details about the experimental setup we implemented (Section 5.1) and the results (5.2), followed by an analysis of tool usage (5.3) and the challenges and costs of model deployment (5.4).

5.1 Experimental Setup

We divided the benchmarks introduced in Section 4 in two groups: **New Benchmarks** (including AbstractAlgebra, QuantumTheorems, and NuminaMath-LEAN) and **PutnamBench**, reflecting two distinct objectives.

In the tests with New Benchmarks, we evaluated the performance of the Ax-Prover against three strong baselines:

- **Claude Sonnet 4** (**Sonnet**). This baseline allows us to assess how the same LLM used to power our framework (see below) performs if used outside the agentic flow and without access to MCP tools.
- **DeepSeek-Prover-V2-671B** (**DS-Prover**) and **Kimina-Prover-72B** (**Kimina**), two specialized Lean provers.

We evaluated all models using pass@1: While this idea is in sharp contrast with previous studies assessing pass with very high values (see, e.g., [49]), we believe it mirrors real-world usage, where researchers are constrained by time and budget limits, and cannot run a prover multiple independent times in the hope that one succeeds. For transparency and reproducibility, we note that while pass@1, for all the baselines, means trying to formalize the entire proof in a single shot, for Ax-Prover it means performing a sequence of steps (i.e., API calls) in which reasoning and tool calls are interleaved in a singular attempt to build the final proof–i.e., with no parallelization. For these experiments, we

powered Ax-Prover using Claude Sonnet 4 [4]. Furthermore, to stay within budget, we capped Ax-Prover API calls at 200 and set a 25-minute timeout. For all models, we computed the final results by running an external Lean compiler on the generated files, and considered a proof correct if it compiled and included no `sorry`.

The second benchmark group, which includes PutnamBench only, aims to evaluate Ax-Prover on one of the most challenging public benchmarks and compare its performance against existing state-of-the-art provers. Accordingly, we did not run baselines and instead directly compared our results with those reported on the official leaderboard [54]. For this test, we powered Ax-Prover with Sonnet-4.5, removed the 25-minute timeout, and increased the max number of API calls at 400, while still running it with pass@1, as defined above.

5.2 Results

Dataset	Subset	Ax-Prover	Sonnet	DS-Prover	Kimina
NuminaMath	solved-K	81%	7%	48%	100%†
	solved-H	47%	8%	14%	0%†
	unsolved	26%	1%	18%	0%†
	total	51%	5%	28%	31%
AbstractAlgebra	easy	72%	10%	26%	12%
	intermediate	56%	6%	22%	14%
	total	64%	8%	24%	13%
QuantumTheorems	easy	100%	54%	88%	72%
	intermediate	92%	18%	48%	34%
	total	96%	40%	61%	57%

Table 2: Models' performance on NuminaMath, AA, and QT. † The results on NuminaMath for Kimina are reported from [44], and where obtained during its RL training phase with, on average, pass@68.

New Benchmarks We report the results for this group in Table 2. On the Numina dataset, Ax-Prover scored 51% accuracy, outperforming DS-Prover (28%) and Kimina (31%) by a similar margin, while Sonnet only got 5% accuracy. Particularly relevant is the performance of Ax-Prover on `Solved-H`, where it solves almost half of the problems, and on `Unsolved` (26%). Notably, due to autoformalization (see Section 4.3), some theorems are ill-posed: during testing, Ax-Prover spotted them, and reported the error (see Appendix C).

On AA the gap in performance is striking, with Ax-Prover (64%) outperforming DS-Prover by 40%, while both Kimina (13%) and Sonnet get a very poor performance (8%). We suggest this is because the AA dataset is largely out-of-distribution for DS-Prover and Kimina. In fact, these models are trained primarily on `Mathlib`, which covers only a minimal subset of abstract algebra, or on undergraduate competition-level math problems, which are qualitative differently from those in AA (See Section 4.1).

On the QT dataset, Ax-Prover achieves perfect performance on the easy split and 92% accuracy on the intermediate split, yielding 96% accuracy overall. This represents a substantial gap compared to DS-Prover (61%) and Kimina (57%), with Sonnet falling well behind at 40%. Also in this case, we suggest that the performance gap stems from our approach's flexibility to adapt across scientific domains, in contrast to the over-specialization of specialized models. To showcase the differences between the models, let's consider the proofs that quantum observables are Hermitian matrices (full proofs available in Appendix D.1). DS-Prover misused the Hermitian field, misunderstanding its type, while Sonnet made a more sophisticated effort but encountered a rewrite pattern mismatch, which highlights its difficulties in managing Lean environment. In contrast, Ax-Prover succeeded through a systematic approach, explicitly applying the Hermitian property to diagonal elements, using the definition of conjugate transpose, and connecting it to the fact that a complex number equal to its conjugate is real. The case highlights that successful formal theorem proving requires careful, step-by-step reasoning, a solid grasp of type theory, and familiarity with library theorems – demonstrating that clarity and correctness outweigh clever shortcuts in formal verification.

PutnamBench Table 3 reports the results for the top 10 scorers on PutnamBench.[3] In the "Compute" column, `pass@` indicates the number of independent attempts to solve a proof. `avg. pass@` is used for Hilbert, an agen-

[3] See the full leaderboard at [54].

tic framework that parallelizes reasoning and verification at different levels [57]. The exact definition of this metric is unclear; our best assumption is that it reflects the average number of calls to Hilbert's sub-agents. Similarly, `medium` refers to a specific test setup for Seed-Prover, in which the model is evaluated with parallelized refinement processes [13]. On this benchmark, Ax-Prover achieves 14% accuracy, placing third in the table. While this result is lower than the top scorers, it is important to note that it was obtained at a fraction of the cost of the top-performing models (see the "Compute" column).

Overall, the results in this section indicate that Ax-Prover delivers strong performance across the board, ranking among the top models in mathematics and outperforming others in physics. Also, they highlight two key limitations of current approaches: specialized provers fail to generalize beyond their training domains, while general-purpose LLMs, while creative, cannot produce rigorous Lean proof. The fact that Ax-Prover nearly doubles the performance of the standalone LLM using the same model (Sonnet) demonstrates that combining agentic reasoning with Lean tool integration is essential for robust theorem proving across domains. We examine this aspect in more detail in the next section.

Model	**Accuracy**	**Compute**
Hilbert	72% [462]	avg. pass@1840
Seed-Prover	51% [329]	medium
Ax-Prover*	14% [92]	pass@1†
Goedel-Prover-V2	13% [86]	pass@184
DeepSeek-Prover-V2	7% [47]	pass@1024
DSP+	4% [23]	pass@128
Bourbaki	2% [14]	pass@512
Kimina-Prover-7B-Distill	2% [10]	pass@192
Self-play Theorem Prover	1% [8]	pass@3200
Goedel-Prover-SFT	1% [7]	pass@512
Gemini-2.5-Pro	0.5% [3]	pass@1
GPT-4o	0.2% [1]	pass@10
Claude-3.7-Sonnet	0% [0]	pass@1

Table 3: Accuracy results on PutnamBench (% and absolute number of solved problems). † Remember that for Ax-Prover, pass@1 is made of multiple steps, see Section 5.1.

5.3 Analysis of Tool Usage

To measure the impact of tool usage on our approach, we analyzed the tool calls done by the Prover over the 100 problems we tested on the challenging NuminaMath `Unsolved` subset. We found that the Prover made an average of 100.76 tool calls per run. Tool usage is highly reliable, with success rates above 99%.[4]

Table 4 reports the 10 most frequently used tools. At the top is `edit_file`, as the Prover updates the Lean file at each step. `lean_diagnostic_messages` follows, reflecting explicit instructions to verify each proof step. `lean_goal` exposes the current proof state, while `lean_loogle` and `lean_leansearch` enable the Prover to search for relevant theorems in the library. Importantly, these tools are used autonomously, without additional guidance. Collectively, these statistics illustrate how Ax-Prover leverages a tight feedback loop of editing, goal inspection, search, and diagnostics.

Our assumption is that tool usage enhances proof quality by allowing Ax-Prover to use less common tactics. To test this, we analyzed the unique tactics used in the proofs generated by Ax-Prover, Kimina, and DeepSeek, under the hypothesis that a larger set of tactics reflects greater creativity (see the full list of tactics per model in Table 5). The three models share 28 tactics, but Ax-Prover uses 9 tactics not employed by DS-Prover or Kimina, whereas these models combine to use only three tactics absent in Ax-Prover. This finding supports our hypothesis that integrating frontier LLMs with Lean tools enhances creative exploration in proof construction.

Tool	**Calls**
edit_file	36.79
lean_diagnostic_messages	30.73
lean_goal	8.17
lean_loogle	5.88
lean_leansearch	4.32
file_contents	3.00
write_file	2.71
read_text_file	2.24
lean_run_code	2.05
lean_hover_info	1.76

Table 4: Tool usage statistics.

5.4 Deployment Analysis

Besides performance, deployment complexity is critical when using AI models in real-world scenarios. Here we compare prover systems in this respect. DS-Prover and Kimina require GPU-accelerated, high-spec machines and are not available through model as a service (MaaS) providers.[5] We hosted DeepSeek and Kimina on Google Cloud: DeepSeek on an A3 Ultra VM

[4] The only exception is `search` in Filesystem tools, with 80%. However, this results from the Prover searching for files that do not exists.

[5] For instance, while DeepSeek-Prover-V2-671B was previously hosted by Novita [21], this endpoint now redirects to the general DeepSeek-V3 model

with eight H200-141GB GPUs, and Kimina on an A2 High GPU VM with eight A100-40GB GPUs. Deployment is burdensome and demands MLOps expertise – users must match hardware specs, configure distributed runtimes, debug serving issues, and contend with scarce GPU availability, since cloud providers enforce strict quotas and long queues for H100/H200 GPUs. This hinders reproducibility even for well-funded teams. In contrast, Ax-Prover relies only on API calls, requiring no infrastructure beyond basic client access, and it can be executed locally on a client machine or remotely in a lightweight container.

On monetary costs, running DS-Prover and Kimina on 1000 datapoints cost approximately $300 and $2000, respectively, while Ax-Prover cost about $4000. At first glance, our approach appears more expensive but only because we evaluate specialized models with pass@1. Had we followed the common practice of running them with higher pass@n values, the cost of these specialized models would have far exceeded ours. Furthermore, consider that on PutnamBench, DS-Prover was run with a pass@1024, thus leveraging way more computational resources, and only got 47 correct theorems, while Ax-Prover got 92. Moreover, general-purpose LLMs are on a rapid trajectory of improvement: each new generation delivers stronger reasoning at lower cost, suggesting that the relative efficiency of Ax-Prover will only increase over time.

The deployment and cost barriers of specialized models also help explain why they have not achieved widespread use beyond benchmark settings such as IMO-style mathematics problems. For most researchers, the need to manage specialized hardware, navigate GPU quotas, and bear high costs makes these systems effectively unusable in practice. Ax-Prover is more accessible to researchers not only because it eliminates these barriers, but also because it was explicitly designed to act as a supportive assistant, as we show in the next section.

6 Use Case: Researcher-Friendly Verification

Ax-Prover is able to engage in productive collaborations with human researchers by verifying intermediate proof steps, providing precise feedback, and guiding the overall direction of the proof. This capability differentiates our framework from existing specialized provers, which generally attempt to complete proofs in a single pass without any interaction with their external environment. Thus, Ax-Prover should be viewed not merely as a backend system but as an active collaborator in mathematical and scientific reasoning. In this Section, we present a concrete demonstration of Ax-Prover's capabilities as an assistant.

One of the authors of this paper is an expert mathematician who collaborated with Ax-Prover to prove a recent cryptography result, *A New Algorithm for Computing Branch Number of Non-Singular Matrices over Finite Fields* [40] (see the full case study in Appendix F). The task involved two main steps, namely: formalizing the paper's statements in Lean, and verifying the main claim. The mathematician jointly worked with Ax-Prover to structure the proof, validate lemmas, and complete the verification. Ax-Prover supported the whole process by checking intermediate lemmas and guiding proof strategies. In what came as a surprise, it also revealed an error in the original approach. This shows that Ax-Prover can not only reproduce known mathematical results at the cutting edge of research, but also advance the state of knowledge by providing formal verification of correct results and expose incorrect ones. Notably, the whole process lasted two working days and was carried out locally on the laptop of the mathematician. The outcome of the effort is a proof of over 2000 lines of Lean code.[6]

To better understand the value of Ax-Prover's contribution to the proof defined above, consider the formalization of the Prime Number Theorem. Terence Tao and Alex Kontorovich initiated the Lean translation, but the project was only completed *weeks later* by Math, Inc.'s Gauss agent running on Morph.AI's Infinibranch cluster [39]. While ultimately successful, this represented a massive engineering effort spanning several weeks for a well-known proof of comparable difficulty to our cryptography case study, which, by contrast, required only two days, a single mathematician, and a laptop.

This comparison underscores the power of Ax-Prover in enabling fast and efficient verification of research-level proofs of new results. Besides its ability to perform end-to-end tasks, Ax-prover is able to act as a collaborative teammate, by exposing intermediate reasoning, suggesting next steps, and accepting and providing guidance.

[6] The resulting Lean blueprint can be made available upon request.

7 Conclusions

We introduce **Ax-Prover**, a novel agentic workflow that combines the broad reasoning capabilities of general-purpose LLMs with the formal rigor of Lean's proof environment. Our system addresses three major limitations of current specialized provers: (i) limited generalizability to scientific domains beyond mathematics and rapid obsolescence as libraries like `Mathlib` evolve; (ii) inability to collaborate effectively with human experts and utilize external tools; and (iii) high engineering and maintenance costs.

Evaluations show that Ax-Prover ranks third on the challenging PutnamBench and outperforms baselines on the public NuminaMath-LEAN dataset as well as on **AbstractAlgebra** and **QuantumTheorems**, two new datasets we introduce that focus on research-level mathematics and physics. These benchmarks not only provide new testbeds for cross-domain reasoning in future agents but also represent a crucial milestone in evaluating reasoning models in any scientific discipline grounded in mathematics.

These results highlight Ax-Prover's superior domain generalization, in contrast to specialized models, which struggle to adapt to novel domains beyond their training data. More importantly, they show that Ax-Prover has the potential to serve as a deep formal reasoning assistant for scientific artificial intelligence in domains requiring extended chains of rigorous inference. The combination of multi-disciplinary reasoning with rigorous formal verification positions the system to support AI-driven scientific discovery wherever verifiably error-free reasoning is essential. We attribute this performance to our multi-agent architecture and its tight integration with Lean tools via the MCP. By iteratively editing proofs, inspecting goals, and diagnosing errors, Ax-Prover behaves like a cautious mathematician, systematically exploring and verifying each step. The frequency and effectiveness of tool use in our experiments confirm their essential role in improving proof quality and enabling human-like debugging.

Furthermore, we showed in our case study on cryptography that Ax-Prover is not only able to prove theorems autonomously, but it can also engage in fruitful collaboration with human researchers. Working alongside it, the mathematician used it as a partner for structuring arguments, validating intermediate lemmas, and diagnosing proof failures. This interaction demonstrates how Ax-Prover can adapt to expert guidance, accelerate verification workflows, and even surface errors in the informal reasoning.

Looking ahead, we plan to enhance Ax-Prover by introducing parallelized agents, enabling the framework to explore multiple proof paths simultaneously. This will increase its creativity and success rate in formalizing complex proofs. We also plan to integrate a long-term memory module to store information from past proofs and human interactions. This capability will allow Ax-Prover to participate not only in standalone problems but also in extended, collaborative research projects requiring sustained expert guidance. These developments will advance us towards our broader goal of verifiable scientific artificial intelligence, where AI systems contribute to scientific discovery through formally validated reasoning.

A Tools

A.1 File system

Full list of Filesystem tools:

- `read_file`
- `read_multiple_files`
- `write_file`
- `edit_file`
- `create_directory`
- `list_directory`
- `list_directory_with_sizes`
- `directory_tree`
- `move_file`

- `search_files`
- `get_file_info`
- `list_allowed_directories`

B Datasets

B.1 Abstract Algebra

B.1.1 Dataset Generation

We used a basic pipeline to build the AbstractAlgebra dataset. First, we extracted all raw text from PDFs of *Abstract Algebra* by Dummit and Foote [23] and *Abstract Algebra: Theory and Applications* by Judsen [29] using Mistral's API. We then processed the raw text by using Claude-Sonnet-3.7 [3] to extract a list of natural language mathematical statements, which include exercises, derivations, lemmas, propositions, and theorems within the text. Next, we used a Claude-Sonnet-3.7 agent to formalize each natural language statement in Lean. To ground the formalization in `Mathlib` and prevent the agent from reinventing definitions, we passed the agent a Lean file at the start of the process containing relevant definitions for that section, e.g., *dihedral groups*, *roots of unity*, or the *field extension* $\mathbb{Q}(\sqrt{2})$. The agent could reference these definitions and was required to add each formalized statement directly to this file, but was explicitly prohibited from introducing new definitions. The agent generated the top 3 formal statements in Lean for each natural language statement and refined each attempt up to 3 times with feedback from the Lean compiler. We then built the dataset by retaining only those pairs of natural language and formal language statements that corresponded to exercises from the source texts.

B.1.2 Example

This is an example of a theorem statement in the AbstractAlgebra dataset, formalized from Exercise 3 in Chapter 1.2 of Dummit and Foote [23].

```
import Mathlib

-- Variables for dihedral group
variable {n : Nat} {i : Int}
local notation "D_n" => DihedralGroup n
local notation "r" => DihedralGroup.r (1 : ZMod n)
local notation "s" => DihedralGroup.sr (0 : ZMod n)

/-- Use the generators and relations to show that every element of D_n not a power of
    r has order 2. -/
theorem exercise_3_part1 {x : D_n} (h : x = s * r ^ i) : orderOf x = 2 := by
  sorry
```

B.2 QuantumTheorems

B.2.1 Dataset Generation

The dataset was generated through an iterative human-in-the-loop process combining automated proof synthesis with expert curation. The initial set of 134 theorems were manually extracted from [43]. An automated coding agent (Claude Opus [6]) first generated formal statements and proof attempts for the theorems, producing both complete proofs and partial derivations. A quantum physics expert then reviewed each statement and proof, identifying gaps, correcting errors, and standardizing operator definitions to ensure that each question was well formed and solvable. The final dataset replaces these proofs with `sorry` statements.

B.2.2 Example

This is an example of a theorem statement in the QuantumTheorems dataset, stating that a post-measurement state is an eigenstate of the measurement projector. Notably, the problem setup involves a number of custom definitions of concepts in quantum mechanics.

```
import Mathlib.Analysis.InnerProductSpace.Basic
import Mathlib.Data.Complex.Basic
import Mathlib.Data.Matrix.Basic
import Mathlib.LinearAlgebra.Eigenspace.Basic

open BigOperators Complex

/-- Quantum state as normalized vector -/
structure QuantumState (n : ℕ) where
  vec : Fin n → ℂ
  normalized : Σ i, Complex.normSq (vec i) = 1

/-- Projector as idempotent matrix -/
structure Projector (n : ℕ) where
  mat : Matrix (Fin n) (Fin n) ℂ
  idem : mat * mat = mat
  hermitian : mat.conjTranspose = mat

/-- Matrix-vector multiplication -/
noncomputable def matVec {n : ℕ} (M : Matrix (Fin n) (Fin n) ℂ) (v : Fin n → ℂ) :
    Fin n → ℂ :=
  fun i => Σ j, M i j * v j

/-- Measurement probability -/
noncomputable def measureProb {n : ℕ} (P : Projector n) (ψ : QuantumState n) : ℝ :=
  Σ i, Complex.normSq ((matVec P.mat ψ.vec) i)

/-- Norm of a vector -/
noncomputable def vecNorm {n : ℕ} (v : Fin n → ℂ) : ℝ :=
  Real.sqrt (Σ i, Complex.normSq (v i))

/-- Scale a vector by a real number -/
noncomputable def scaleVec {n : ℕ} (r : ℝ) (v : Fin n → ℂ) : Fin n → ℂ :=
  fun i => r · (v i)

/-- Check if a vector is an eigenvector with eigenvalue lambda -/
def isEigenvector {n : ℕ} (M : Matrix (Fin n) (Fin n) ℂ) (v : Fin n → ℂ) (lambda : ℂ
    ) : Prop :=
  v ≠ 0 ∧ matVec M v = fun i => lambda * v i

  /-- QT_366: Post-measurement state is eigenstate of measurement projector -/
theorem QT_366_post_measurement_eigenstate {n : ℕ} (P : Projector n) (ψ :
    QuantumState n)
    (h_nonzero : measureProb P ψ ≠ 0) :
    let ψ' := matVec P.mat ψ.vec
    isEigenvector P.mat ψ' 1 := by
    sorry
```

C Detected Autoformalization Error

As noted in Section 5.2, 19.7% of Numina's problems were generated using autoformalization models. While these pipelines enable large-scale dataset construction, they occasionally produce ill-posed theorems that cannot be satisfied

in Lean.

During evaluation, Ax-Prover successfully identified such a case and proved the negation of the statement.

```
import Mathlib

theorem number_theory_3098 (p1 p2 p3 p4 : ℕ) (hp1 : p1.Prime) (hp2 : p2.Prime)
    (hp3 : p3.Prime) (hp4 : p4.Prime) (h1 : p1 < 100) (h2 : p2 < 100) (h3 : p3 < 100)
    (h4 : p4 < 100) (h5 : p1 ≠ p2) (h6 : p1 ≠ p3) (h7 : p1 ≠ p4) (h8 : p2 ≠ p3)
    (h9 : p2 ≠ p4) (h10 : p3 ≠ p4) (h11 : p1 = 1 ∨ p1 = 2 ∨ p1 = 3 ∨ p1 = 4 ∨ p1 = 5
   ∨ p1 = 6 ∨ p1 = 7 ∨ p1 = 9)
    (h12 : p2 = 1 ∨ p2 = 2 ∨ p2 = 3 ∨ p2 = 4 ∨ p2 = 5 ∨ p2 = 6 ∨ p2 = 7 ∨ p2 = 9)
    (h13 : p3 = 1 ∨ p3 = 2 ∨ p3 = 3 ∨ p3 = 4 ∨ p3 = 5 ∨ p3 = 6 ∨ p3 = 7 ∨ p3 = 9)
    (h14 : p4 = 1 ∨ p4 = 2 ∨ p4 = 3 ∨ p4 = 4 ∨ p4 = 5 ∨ p4 = 6 ∨ p4 = 7 ∨ p4 = 9)
    (h15 : p1 ≠ p2 ∧ p1 ≠ p3 ∧ p1 ≠ p4 ∧ p2 ≠ p3 ∧ p2 ≠ p4 ∧ p3 ≠ p4) :
   p1 + p2 + p3 + p4 = 190 := by  sorry
```

The first line of the proof sketch that Ax-Prover generated for this problem was

This theorem has contradictory premises: the sum must be 17, not 190.

Upon inspection, it is clear that 4 natural numbers belonging to the set $\{2, 3, 5, 7\}$ cannot sum to 190. As an additional exercise, we changed

```
p1 + p2 + p3 + p4 = 190 := by  sorry
```

to its negation

```
(p1 + p2 + p3 + p4 = 190) = False := by  sorry
```

changing the original theorem statement to prove the negation which Ax-Prover was able to do, thus proving that the original theorem was not provable. This behavior highlights two strengths of the agentic loop:

1. **Robustness to noise.** The agent does not blindly attempt to complete invalid statements but can detect contradictions early.
2. **Transparency.** By surfacing diagnostic messages directly from Lean, Ax-Prover provides clear evidence that the statement is ill-posed, enabling researchers to filter or repair dataset items.

We see this as an important complement to raw accuracy metrics: beyond solving valid theorems, a reliable prover should be able to recognize when formalization errors make a proof impossible.

D Proofs

D.1 QuantumTheorems Example Analysis

We present here the attempts to prove that diagonal matrix elements of a quantum observable in a finite-dimensional Hilbert space are real. Concretely,

Let A be a quantum observable in an n-dimensional Hilbert space. Then for every $i = 1, \cdots, n$, the imaginary part of the element A_{ii} vanishes.

This entry in the dataset comes with a custom definition of an observable:

```
/-- An observable is a Hermitian operator -/
structure Observable (n : ℕ) where
  matrix : Matrix (Fin n) (Fin n) ℂ
  hermitian : matrix.IsHermitian
```

D.1.1 DeepSeek's Attempt (Failed)

DeepSeek's approach was fundamentally flawed from the start:

```
theorem observable_diagonal_real {n : ℕ} (A : Observable n) (i : Fin n) :
    (A.matrix i i).im = 0 := by
  have h := A.hermitian
  have h₁ := congr_arg (fun x => x i i) h
  simp [Matrix.ext_iff, Matrix.conjTranspose_apply, Complex.ext_iff] at h₁ ⊢
  <;>
    aesop
```

Critical Error:

```
tactic 'aesop' failed, made no progress
```

Analysis: DeepSeek attempted to use `A.hermitian` directly as a proof, but this term has type `Matrix.IsHermitian`, not the equality type required by `congr_arg`. The error message indicates that the tactic `aesop` failed because there was a fundamental type mismatch. This shows a misunderstanding of Lean's type system. The `hermitian` field is a proof that the matrix satisfies the Hermitian property, but it's not directly the equality we need to prove.

D.1.2 Sonnet's Attempt (Failed)

Sonnet made a more sophisticated attempt but still failed:

```
theorem observable_diagonal_real {n : ℕ} (A : Observable n) (i : Fin n) :
    (A.matrix i i).im = 0 := by
  have h := A.hermitian
  rw [Matrix.IsHermitian.ext_iff] at h
  specialize h i i
  rw [Matrix.conjTranspose_apply, Matrix.star_apply] at h
  exact Complex.ext_iff.mp h.symm |>.2
```

Critical Error:

```
Tactic `rewrite` failed: Did not find an occurrence of the pattern
  Matrix.conjTranspose ?M ?j ?i
in the target expression
  star (A.matrix i i) = A.matrix i i
```

Analysis: Sonnet understood that it needed to work with the Hermitian property and tried to extract information about diagonal elements, but made an error in pattern matching during the rewrite step. The approach was on the right track but was executed incorrectly.

D.1.3 Ax-Prover's Attempt (Successful)

Ax-Prover succeeded with a systematic, step-by-step approach:

```
theorem observable_diagonal_real {n : ℕ} (A : Observable n) (i : Fin n) :
    (A.matrix i i).im = 0 := by
  -- Step 1: Use the fact that A is Hermitian
  have h1 : A.matrix.conjTranspose = A.matrix := by
    exact A.hermitian

  -- Step 2: Apply this to diagonal element (i,i)
  have h2 : (A.matrix.conjTranspose i i) = (A.matrix i i) := by
    rw [h1]

  -- Step 3: By definition of conjugate transpose
  have h3 : (A.matrix.conjTranspose i i) = star (A.matrix i i) := by
    exact Matrix.conjTranspose_apply _ _ _
```

```
-- Step 4: Combine to get star (A.matrix i i) = A.matrix i i
have h4 : star (A.matrix i i) = A.matrix i i := by
  rw [← h3, h2]

-- Step 5: A complex number equals its conjugate iff it's real
have h5 : (A.matrix i i).im = 0 := by
  have : (starRingEnd ℂ) (A.matrix i i) = A.matrix i i := h4
  exact Complex.conj_eq_iff_im.mp this

exact h5
```

E Tactics

In Table 5 we report the tactics used by each model.

F Case Study: Verifying math in classical cryptographic papers

In this case study, we illustrate how one of our researchers used Ax-Prover to verify the correctness of mathematical results used in cryptographic research.

As a concrete example, we focus on the recent (May 2024) cryptographic paper *A New Algorithm for Computing Branch Number of Non-Singular Matrices over Finite Fields* from arXiv [40]. This work introduces a novel algorithm for computing the *branch number* – a fundamental metric used to assess the strength of block ciphers such as AES [42], PRINCE [12], and Grøst [24].

The paper begins with **Theorem 1**, which offers an alternative characterization of the branch number. Traditionally, for an invertible $n \times n$ matrix M of order $n > 1$ over a finite field $\mathbb{F}_q$ of order q, the branch number is defined as

$$\mathcal{B}(M) = \min \left\{ w_h(x) + w_h(Mx) : x \in \mathbb{F}_q^n \text{ where } x \neq 0 \right\}, \tag{1}$$

where $w_h(x)$ is the Hamming weight (the number of nonzero entries in x). Theorem 1 gives an alternate definition of the branch number that is more amenable to computation than the classical version:

The branch number of an invertible matrix $M \in M_n(\mathbb{F}_q)$ is given as

$$\mathcal{B}(M) = \min \left\{ \min\{h(M,x), h(M^{-1},x)\} : x \in \mathbb{F}_q^n, 1 \leq w_h(x) \leq \left\lfloor \frac{n+1}{2} \right\rfloor \right\}, \tag{2}$$

where $h(M,x) = w_h(x) + w_h(Mx)$.

For cryptographers, this makes a practical difference: it enables fast evaluation of candidate matrices when designing new lightweight or high-performance ciphers. The authors demonstrate in Theorem 4 [40] that their algorithm achieves significant complexity gains over the naive $O(n^2 q^n)$ approach for finite fields of order $q \geq 4$ and square matrices of order $n \geq 4$.

Tactic	Ax-agent	DeepSeek	Kimina
apply	X	X	X
assumption	X	X	X
by_cases	X	X	X
calc	X	X	X
cases	X	X	X
change	X		
classical	X	X	
congr	X	X	X
constructor	X	X	X
contradiction	X	X	X
decide	X	X	
exact	X	X	X
exact_mod_cast	X	X	X
exfalso	X	X	X
ext	X	X	X
funext		X	X
generalize	X		
induction	X	X	X
injection	X		
intro	X	X	X
intros	X		
left	X		X
native_decide	X		X
norm_cast	X		
obtain	X	X	X
omega	X	X	X
push_cast	X		
rcases	X	X	X
refine	X	X	X
replace	X		X
rfl	X	X	X
right	X		X
rintro	X	X	X
rw	X	X	X
rwa	X		X
show	X		X
simp	X	X	X
simp_all	X	X	X
simpa	X	X	X
specialize			X
subst	X	X	X
subst_vars		X	
suffices	X		
trans	X		
unfold	X		

Table 5: Tactics used by Ax-agent, DeepSeek, and Kimina. An ”X” indicates the model uses the tactic.

F.1 Formalize: Single Step

Note that for the second term of the right-hand side of Equation (2), $h(M,x) = w_h(x)+w_h(Mx) > 2\left\lfloor\frac{n+1}{2}\right\rfloor+1 \geq n+1$. However, we know that the upper bound for $\mathcal{B}(M)$ is $n+1$. Thus, we conclude that the second term of the right-hand side of (2) will not contribute to the computation of the branch number.

Therefore, from (1) and (2), we have

$$\mathcal{B}(M) = \min\left\{\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, 1 \leq w_h(x) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\}, \min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor\frac{n+1}{2}\right\rfloor < w_h(x) \leq n, w_h(Mx) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\}\right\}. \tag{3}$$

Again, we note that

$$\left\{h(M,x) \mid x \in \mathbb{F}_q^n, 1 \leq w_h(x) \leq \left\lfloor\frac{n+1}{2}\right\rfloor, w_h(Mx) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\} \subseteq \left\{h(M,x) \mid x \in \mathbb{F}_q^n, 1 \leq w_h(x) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\}.$$

To formally verify the math in this paper, we used an autoformalization agent to formalize statements, verified that the formalization was correct, before passing those statements to Ax-Prover.

We show the process of proving one step in the paper (the full lean certificate can be provided upon request). The figure below shows the current verification state highlighted in green, while the next step awaiting verification appears in yellow.

F.2 Verification: Single Step

We highlighted the relevant theorems or lemmas and prompted Ax-Prover to generate a complete proof for each selected statement. Ax-Prover iteratively refined the proof, checking each step for correctness until the system confirmed its validity. Unlike standard LLM output, this process ensured full mathematical soundness, with Ax-Prover automatically searching for relevant results, reading and updating the file, diagnosing errors, and managing proof goals.

Some expert human oversight was required on 2 steps to properly formalize statements passed to the prover, but beyond these isolated interventions, the system operated autonomously. After several iterations, it produced a fully verified proof of each step in just a few minutes. The following prompt illustrates this process for the example above:

Prompt

Prove this statement (with the formalized step above highlighted as context).

Response

The statement is a simple set-theoretic observation. We start with all low-weight vectors x and look at the values $w_H(x) + w_H(Mx)$, where w_H is the Hamming weight and M is some matrix.

- Without constraints: We consider all low-weight vectors.
- With constraints: We only consider those low-weight vectors where the output M x also has low weight.

Clearly, if you add an extra condition ("M x has low weight"), you can only make the set smaller or equal—never larger—because you're restricting the set of vectors you're looking at.
The proof just formalizes this obvious fact:

1. Take any element in the constrained set.
2. By definition, it comes from some vector x that satisfies the extra condition.
3. But x also satisfies the conditions for the unconstrained set (because the unconstrained set has fewer requirements).
4. So the element belongs to the unconstrained set as well.

This shows the constrained set is a subset of the unconstrained set.

We now present an example that required expert oversight. Specifically, the Ax-Prover was unable to verify the correctness of statement (3) shown below:

> Note that for the second term of the right-hand side of Equation (2), $h(M,x) = w_h(x)+w_h(Mx) > 2\left\lfloor\frac{n+1}{2}\right\rfloor+1 \geq n+1$. However, we know that the upper bound for $\mathcal{B}(M)$ is $n+1$. Thus, we conclude that the second term of the right-hand side of (2) will not contribute to the computation of the branch number.
>
> Therefore, from (1) and (2), we have

$$\mathcal{B}(M) = \min\left\{\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, 1 \leq w_h(x) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\},\right.$$
$$\left.\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor\frac{n+1}{2}\right\rfloor < w_h(x) \leq n, w_h(Mx) \leq \left\lfloor\frac{n+1}{2}\right\rfloor\right\}\right\}. \tag{3}$$

To guide Ax-Prover, an expert user provided the following prompt:

Prompt

You have defined the sets B, C, D with $A = \min\big(B, \min(C, D)\big) \leq n+1$. Lets do this by cases: If $\min C > \min D$ and $\min D > n+1$, then $\min(C, D) > n+1$, so $\min(B, \min(C, D)) = \min(B)$. The other case is easier.

The Ax-Prover then successfully proved the statement using the provided hint:

Response

VERIFICATION COMPLETE - STEPS:

1. **Define the minima:** Let

$$B := \min\{\, w_H(x) + w_H(Mx) \mid x \in L\},$$
$$C := \min\{\, w_H(x) + w_H(Mx) \mid x \in H_1\},$$
$$D := \min\{\, w_H(x) + w_H(Mx) \mid x \in H_2\}.$$

where L = low-weight set, H_1 = high-weight with $w_H(Mx) \leq (n+1)/2$, H_2 = high-weight with $w_H(Mx) > (n+1)/2$. Then

$$A := \text{Branchnumber}(M) = \min(B, \min(C, D)).$$

2. **Upper bound:** For invertible M,

$$A \leq n + 1.$$

3. **High–high set bound:** If $x \in H_2$, then

$$w_H(x) + w_H(Mx) > n + 1 \quad \Longrightarrow \quad D > n + 1.$$

4. **Case analysis:**

- $C \leq D \quad \Rightarrow \quad \min(C, D) = C \quad \Rightarrow \quad A = \min(B, C)$.
- $C > D \quad \Rightarrow \quad \min(C, D) = D > n + 1$. Since $A = \min(B, D) \leq n + 1$, we get $B \leq n + 1$ and thus $A = B = \min(B, C)$.

5. **Conclusion:**

$$A = \min(B, C).$$

The term D never affects the branch number.

F.3 Consolidation: Complete Certificate

Once all steps were verified, they were integrated into a single proof, producing a final file that formally certifies the alternate definition is equivalent to the original, exactly as proposed in the paper.

F.4 Correctness, Verification, and Takeaways

The outcome of our work is a formally verified certificate ensuring the correctness of Theorem 1. During the course of constructing this certificate, we identified a critical error in the original proof. Specifically, the issue arises in **Step 2** of the proof:

$$\begin{aligned}
&\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor \frac{n+1}{2} \right\rfloor < w_h(x) \leq n\right\} \\
=&\min\left\{\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor \frac{n+1}{2} \right\rfloor < w_h(x) \leq n, w_h(Mx) \leq \left\lfloor \frac{n+1}{2} \right\rfloor\right\},\right. \\
&\left.\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor \frac{n+1}{2} \right\rfloor < w_h(x) \leq n, w_h(Mx) > \left\lfloor \frac{n+1}{2} \right\rfloor\right\}\right\}.
\end{aligned} \tag{2}$$

Here, the authors fail to ensure that the sets over which they take minima are non-empty. For example, in the simplest case where $M = I$ (the identity matrix), the middle term reduces to

$$\min\left\{h(M,x) \mid x \in \mathbb{F}_q^n, \left\lfloor \frac{n+1}{2} \right\rfloor < w_h(x) \leq n, w_h(x) \leq \left\lfloor \frac{n+1}{2} \right\rfloor\right\}.$$

In this case, the constraints

$$\left\lfloor \tfrac{n+1}{2} \right\rfloor < w_h(x) \quad \text{and} \quad w_h(x) \leq \left\lfloor \tfrac{n+1}{2} \right\rfloor$$

are contradictory, so the underlying set is *empty*. Nevertheless, the original proof proceeds under the assumption that this minimum is well-defined, a subtle yet significant oversight.

This matters for two reasons:

1. **Logical correctness:** Reasoning about the empty set is problematic (all statements are *vacuously true*) which can lead to unsound conclusions. For example, let

   $$S = \{\, x \in \mathbb{Z} \mid x = 3 \text{ and } x \text{ is even} \,\}.$$

   Take $y \in S$, then $y = 3$ and y is even, so this implies that 3 is even.

2. **Software implementation:** Computing the minimum of an empty set is undefined in standard programming environments and would trigger a runtime error if translated directly into code.

Our formal verification system flagged these issues because it could not establish the truth of the corresponding statements, revealing logical gaps in the proof. Nevertheless, the authors' final result remains correct despite the critical error in their proof.

G Case Study 2: Quantum Cryptography

Quantum cryptography seeks security guarantees that are statistical; derived from the laws of physics and information theory, rather than computational. An important example is quantum key distribution (QKD), a key–establishment protocol in which two parties certify secrecy by testing quantum correlations rather than assuming limits on an adversary's computing power. Because these guarantees rest on first principles in linear algebra, probability, and quantum mechanics, the field is an especially natural fit for automated theorem proving: formal proofs can turn widely cited derivations into reusable components that compose into end-to-end security arguments.

Our second use case focuses on the Lo–Chau security framework [37], reproduced in this work. That paper laid the foundations of modern QKD by reducing security to verifiable statements about entanglement and then to classical probabilistic reasoning, and it informed subsequent analyses such as the Shor–Preskill proof of BB84 [52]. Within

that framework, a key step is to convert a physical test, high fidelity to R EPR pairs, into an information-theoretic guarantee of low entropy and therefore limited eavesdropper information.

Using Ax-Prover in interactive mode, we proved in Lean the entropy bound that implements this conversion, Lo–Chau's *Lemma 1 (High fidelity implies low entropy)*, and packaged it as a library lemma for downstream use (see Section G). This result illustrates how a tool-based prover can assist domain experts in translating physics-style arguments into machine-checked mathematics, strengthening the statistical-security foundations on which modern quantum cryptography is built.

In what follows, we cite and explain in detail the lemma and corresponding lean certificate.

G.1 Quantum Key Distribution Lemma

Let ρ be a density matrix on a 2^{2R}-dimensional Hilbert space. If its fidelity with the ideal R-singlet state satisfies $\langle R \text{ singlets} \mid \rho \mid R \text{ singlets}\rangle > 1-\delta$ with $0 < \delta \ll 1$, then

$$S(\rho) \;<\; -(1-\delta)\log_2(1-\delta) \;-\; \delta\log_2\Big(\frac{\delta}{2^{2R}-1}\Big).$$

This is the form reproduced in this work from the Lo–Chau reprint [37]. Here, however, instead of the statement $\delta \ll 1$ we used: $\delta \leq 1 - \frac{1}{2^{2R}-1}$)

The fidelity condition implies the largest eigenvalue of ρ is at least $1-\delta$. Since von Neumann entropy is Schur-concave, the maximal entropy under this constraint is achieved by the extremal spectrum $\big(1-\delta, \frac{\delta}{2^{2R}-1}, \ldots, \frac{\delta}{2^{2R}-1}\big)$, which yields the stated bound. Our Lean certificate for this result follows this reduction and checks the necessary concavity and normalization facts. This lean certificate can be made available upon request.

In the Lo–Chau security reduction, the lemma turns "almost-perfect EPR pairs" into a quantitative entropy bound that, together with standard information-theoretic tools, limits an eavesdropper's knowledge of the final key. Making this step machine-checked enables principled composition with other verified components in formal QKD security proofs. Thus, Ax-Prover bridges formal reasoning and quantitative quantum information theory: results such as the Lo–Chau entropy bound no longer have to be taken as a black box, but instead become certified components, ready for use in end-to-end formal verification of QKD security proofs.

References

[1] Mathematics (arxiv archive). arXiv, 2025. Accessed: 2025-09-24.

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

[3] Anthropic. Claude 3.7 sonnet and claude code. https://www.anthropic.com/news/claude-3-7-sonnet, 2025. Accessed: YYYY-MM-DD.

[4] Anthropic. Claude 4. https://www.anthropic.com/news/claude-4, 2025. Accessed: 2025-09-16.

[5] Anthropic. Claude documentation, 2025. Accessed: 2025-09-19.

[6] Anthropic. Claude opus 4.1. https://www.anthropic.com/claude/opus, 2025. Accessed: 2025-10-14.

[7] Ethan Ayers et al. Leanlm: Large language models for lean theorem proving. *arXiv preprint arXiv:2306.09264*, 2023.

[8] Zhenisbek Azerbayev et al. Formal proving with llms: Lean as a benchmark. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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

[10] Haniel Barbosa, Clark Barrett, Pascal Fontaine, and Andrew Reynolds. Satisfiability modulo theories: An appetizer. *Communications of the ACM*, 65(6):69–77, 2022.

[11] Bart Blaauwbroek et al. Tactician: Lean proof automation with knn. In *Proceedings of the International Conference on Interactive Theorem Proving (ITP)*, pages 348–366. Springer, 2020.

[12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. Prince – a low-latency block cipher for pervasive computing applications. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, Lecture Notes in Computer Science, pages 208–225, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

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

[14] Mark Chen et al. Evaluating large language models trained on code. In *arXiv preprint arXiv:2107.03374*, 2021.

[15] Yilei Chen. Quantum algorithms for lattice problems. Technical report, Cryptology ePrint Archive, Report 2024/555, 2024. Updated April 18: algorithm contains an unfixable bug invalidating the main claim (see Section 3.5.9, Page 37).

[16] 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.

[17] Emily Collins et al. Llms as conversational partners for mathematicians. *arXiv preprint arXiv:2305.XXXX*, 2023.

[18] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340. Springer, 2008.

[19] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *Automated Deduction – CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[20] DeepSeek-AI. Deepseek-prover-v1 dataset. https://huggingface.co/datasets/deepseek-ai/DeepSeek-Prover-V1, 2024. Accessed: 2025-08-24.

[21] deepseek ai. Deepseek-prover-v2-671b, 2025. Accessed: 2025-09-24.

[22] Oliver Dressler. Lean-lsp-mcp: Tools for agentic interaction with the lean theorem prover, 3 2025. Accessed: 2025-08-24.

[23] David S Dummit and Richard M Foote. Abstract algebra. john wile & sons. *Inc., Hoboken, NJ*, 2004.

[24] Praveen Gauravaram, Lars Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a sha-3 candidate. Submission to nist, NIST, September 2008. Available at http://www.groestl.info/.

[25] Thérèse Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to prove with tactics. In *Proceedings of the International Conference on Automated Deduction (CADE)*, pages 275–294. Springer, 2021.

[26] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

[27] Daniel Huang et al. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, pages 2654–2663, 2019.

[28] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, et al. Deepmath - deep sequence models for premise selection. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2235–2243, 2016.

[29] Thomas W Judson. *Abstract algebra: theory and applications*. 2020.

[30] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 1–35. Springer, 2013.

[31] Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. Leanagent: Lifelong learning for formal theorem proving. *arXiv preprint arXiv:2410.06209*, 2024.

[32] Guillaume Lample and François Charton. Deep reinforcement learning for theorem proving. In *International Conference on Learning Representations (ICLR)*, 2022.

[33] Lean Prover Community. Mathlib statistics. https://leanprover-community.github.io/mathlib_stats.html, 2025. GitHub repository for generating statistics plots for Mathlib; accessed 2025-08-24.

[34] 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://huggingface.co/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

[35] 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.

[36] Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025.

[37] Hoi-Kwong Lo and Hoi Fung Chau. Unconditional security of quantum key distribution over arbitrarily long distances. *science*, 283(5410):2050–2056, 1999.

[38] Zeyuan Lu, Guodong Zhang, Junxiao Chen, Huan Chen, Yilun Chen, Zonglin Li, Yiping Li, Lianmin Wang, Yao Lin, Ce Zhang, and Jie Chen. Process-driven autoformalization in lean 4. In *The Thirteenth International Conference on Learning Representations*, 2025.

[39] Math Inc. Introducing gauss, an agent for autoformalization, 2025. Announcement of autoformalization agent for formal verification in mathematics.

[40] P. R. Mishra, Yogesh Kumar, Susanta Samanta, and Atul Gaur. A new algorithm for computing branch number of non-singular matrices over finite fields. *arXiv preprint arXiv:2405.07007*, 2024.

[41] Model Context Protocol. What is the model context protocol (mcp)? https://modelcontextprotocol.io/docs/getting-started/intro, 2024. Accessed: 2025-10-05.

[42] National Institute of Standards and Technology. Advanced encryption standard (aes). Federal Information Processing Standards Publication FIPS 197-upd1, U.S. Department of Commerce, Gaithersburg, MD, 2001. Published November 26, 2001; Updated May 9, 2023.

[43] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[44] Numina-Team. Numinamath-lean dataset. https://huggingface.co/datasets/AI-MO/NuminaMath-LEAN, 2025. Accessed: 2025-08-24.

[45] OpenAI. Openai models documentation, 2025. Accessed: 2025-09-19.

[46] Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. Apollo: Automated llm and lean collaboration for advanced formal reasoning. *arXiv preprint arXiv:2505.05758*, 2025.

[47] Stanislas Polu et al. Formal mathematics statement curriculum learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[48] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. In *International Conference on Learning Representations (ICLR)*, 2020.

[49] Z.Z. Ren, Zhihong Shao, Wenfeng Liang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. https://arxiv.org/abs/2504.21801, 2025. Accessed: 2025-08-24.

[50] Sébastien Rousseau. Bug discovered in quantum algorithm for lattice-based crypto. https://sebastienrousseau.com/2024-04-22-bug-discovered-in-quantum-algorithm-for-lattice-base index.html, April 22 2024. Accessed: [add access date here].

[51] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. E prover 2.0: Integrating equational and first-order logic. In *Proceedings of the International Conference on Automated Deduction (CADE)*, pages 523–541. Springer, 2019.

[52] Peter W Shor and John Preskill. Simple proof of security of the bb84 quantum key distribution protocol. *Physical review letters*, 85(2):441, 2000.

[53] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

[54] Trishullab. Putnambench leaderboard. https://trishullab.github.io/PutnamBench/leaderboard.html, 2025. Accessed: 2025-10-07.

[55] 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. *Advances in Neural Information Processing Systems*, 37:11545–11569, 2024.

[56] Josef Urban, Geoff Sutcliffe, Stefan Petrov, and Josef Vyskočil. Machine learning preselected proof steps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2046–2051, 2011.

[57] Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. *arXiv preprint arXiv:2509.22819*, 2025.

[58] Haiming Wang et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. https://arxiv.org/abs/2504.11354, 2025. Accessed: 2025-08-24.

[59] Qihao Wu, Haotian Zhang, Jialin Chen, Yizhou Li, Xingjian Zhang, Ce Zhang, and Jie Chen. Autoformalization in the era of large language models: A survey. *arXiv preprint arXiv:2505.23486*, 2025.

[60] Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. https://arxiv.org/abs/2405.14333, 2024. Accessed: 2025-08-24.

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

[62] Zhangir Xin et al. Leandojo: Theorem proving with large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[63] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

[64] 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. https://arxiv.org/abs/2406.03847, 2024. Accessed: 2025-08-24.

[65] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: A cross-system benchmark for formal olympiad-level mathematics. https://arxiv.org/abs/2109.00110, 2021. Accessed: 2025-08-24.