LEAP: Supercharging LLMs for Formal Mathematics with Agentic Frameworks

Po-Nien Kung[1,4], Linfeng Song[2], Dawsen Hwang[3], Jinsung Yoon[1]
Chun-Liang Li[1], Simone Severini[2], Mirek Olšák[3], Edward Lockhart[3]
Quoc V Le[3], Burak Gokturk[1], Thang Luong[3], Tomas Pfister[1], Nanyun Peng[1,4]

[1]Google Cloud AI Research [2]Google Cloud [3]Google DeepMind
[4]University of California, Los Angeles

Abstract

Large Language Models (LLMs) exhibit strong informal mathematical reasoning but struggle to generate mechanically verifiable proofs in formal languages like Lean. We present LEAP (LLM-in-Lean Environment Agentic Prover), an agentic framework that enables general-purpose foundation models to achieve state-of-the-art performance on automated formal theorem proving. LEAP leverages foundation model capabilities, such as informal reasoning, instruction following, and iterative self-refinement. By decomposing complex problems into smaller units, the system bridges formal proof construction with informal blueprints through continuous interaction with the Lean compiler. To provide a rigorous evaluation beyond increasingly saturated benchmarks, we introduce Lean-IMO-Bench, a benchmark of IMO-style problems formalized in Lean, with short statements yet highly non-routine and multi-step proofs across a wide range of difficulty levels. Empirically, on the latest 2025 Putnam Competition, an annual mathematics competition for undergraduate students in North America, LEAP solves all 12 problems, matching recent breakthroughs by frontier formal mathematical models; on Lean-IMO-Bench, LEAP raises the one-shot formal solve rate of general-purpose LLMs from below 10% to 70%. It also compares favorably to the best baseline performance of 48% attained by a specialized system with dedicated ATP components that achieved gold-medal-level performance at the 2025 IMO.Furthermore, we demonstrate LEAP's research-level utility by autonomously formalizing complex proofs for open combinatorial challenges, including a verified proof for a key subproblem in Knuth's Hamiltonian decomposition of even-order Cayley graphs.

1 Introduction

Large Language Models (LLMs) have made impressive progress on mathematical reasoning with natural language, also known as "informal math reasoning", demonstrating strong capabilities in complex reasoning and problem-solving for both math competitions and research level maths [Huang and Yang, 2025, Luong and Lockhart, 2025, Feng et al., 2026c,a,b]. However, as discussed in recent works like Hilbert [Varambally et al., 2025] and Goedel-Prover-V2 [Lin et al., 2026], solutions in natural language frequently suffer from logical fallacies and hallucinations, and they are hard to automatically verify. This difficulty in verification is not merely a limitation of automated systems; even for human mathematicians, verifying complex proofs is a notoriously time-consuming process requiring scarce expert labor [Greiffenhagen, 2024]. A famous example is the proof of the Kepler conjecture [Hales, 2005], which required four years of peer review before the referees could only claim to be "99% certain" of its correctness,[1] eventually necessitating a decade-long formal verification

Corresponding authors: Po-Nien Kung (ponienkung@ucla.edu) and Nanyun Peng (violetpeng@google.com). This work was conducted while Po-Nien Kung was an intern at Google Cloud AI Research.

[1] https://en.wikipedia.org/wiki/Kepler_conjecture

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

effort [Hales et al., 2017]. This verification bottleneck underscores that assessing correctness is a hard task in natural language, motivating the exploration of formal mathematics, where proofs are written in a machine-checkable language and verified by a rigorous kernel like in Lean [Moura and Ullrich, 2021], Isabelle [Nipkow et al., 2002], Coq [Huet et al., 1997], HOL Light [Harrison, 2009], offers automated verification with guaranteed accuracy. Yet, bridging the gap to formal theorem proving remains a significant challenge, and the performance of automated formal provers currently lags substantially behind that of general-purpose LLMs operating in natural language.

To bridge this gap, recent efforts in the research community predominantly fine-tunes specialized prover models (e.g., AlphaProof [Hubert et al., 2026], DeepSeek Prover V2 [Ren et al., 2025], Seed Prover [Chen et al., 2025b], Goedel Prover V2 [Lin et al., 2026]) on formal corpora, with the assumption that general LLMs are ineffective for rigorous formal tasks without specialization. Indeed, according to the FormalProofBench [Ravi et al., 2026][2] and TaoBench [Taylor et al., 2026], general LLMs often underperform compared to specialized prover models.

Several agentic systems combine general LLMs with specialized proving components. For instance, Hilbert [Varambally et al., 2025], Aristotle [Achim et al., 2025], and Seed Prover V1.5 [Chen et al., 2025a] use general LLMs for informal reasoning but rely on specialized models for Lean proving steps. General-purpose LLM agents such as COPRA [Thakur et al., 2024], AlphaProof Nexus [Tsoukalas et al., 2026], Numina-Lean-Agent [Liu et al., 2026], and Archon [Ju et al., 2026] have also demonstrated strong formal proving. Numina-Lean-Agent and Axiom[3] both report solving all 12 Putnam 2025 problems. Unlike Numina's flexible coding-agent interface, LEAP maintains an explicit AND–OR DAG of verified decompositions and reusable lemma dependencies.

In this paper, we show that while general LLMs remain limited in one-shot theorem proving, the bottleneck is not language comprehension but generating long, complex, correct proofs in one attempt. General LLMs offer complementary skills to specialized models: strong informal reasoning, instruction following, tool use, and self-refinement. These make them ideal for agentic ATP frameworks, where proof construction is decomposed and iteratively improved. To this end, we introduce **LEAP (LLM-in-Lean Environment Agentic Prover)**, an agentic framework using *only* general LLMs for formal math. Inspired by human workflow, LEAP generates a high-level blueprint forming a directed acyclic graph (DAG), then generates the Lean proof, iteratively correcting errors via compiler feedback.

To evaluate progress beyond saturated benchmarks such as MiniF2F [Zheng et al., 2022] and PutnamBench [Tsoukalas et al., 2024], we introduce Lean-IMO-Bench, formalizing the challenging informal math benchmark IMO-Bench [Luong et al., 2025] problem statements into Lean. In contrast to existing benchmarks, which either focus on shorter problems or emphasize broad undergraduate coverage, Lean-IMO-Bench targets the complementary regime of elementary statements whose solutions often hinge on highly non-routine insights and unfold through long, multi-step, and structurally intricate proofs, providing a sharper test of formal theorem proving.

Empirically, on the latest 2025 Putnam Competition, a challenging annual undergraduate mathematics competition in North America whose 2025 top score was 110 out of 120 while the median was only 2, LEAP solves all 12 problems in Lean, achieving perfect performance. This matches results reported by Axiom and Numina-Lean-Agent, with lower reported aggregate runtime. On Lean-IMO-Bench, LEAP substantially improves general LLMs' solve rate from under 10% to 70%, surpassing specialized ATP models (5%) and Aristotle (48%), a strong system with specialized ATP components that earned the score for Gold medal at IMO 2025.

The contribution of the paper is three-fold:

- **Workflow-Inspired Agentic Design** We introduce LEAP, an agentic framework that codifies the human mathematical workflow – combining high-level blueprint sketching with low-level formal proof generation and iterative compiler feedback. We focus on how explicit proof-state management, verified decomposition, and hierarchical memoization improve the effectiveness of general-purpose LLM agents without task-specific prover fine-tuning.
- **Lean-IMO-Bench Dataset:** To evaluate progress beyond saturated benchmarks such as MiniF2F and PutnamBench, we introduce Lean-IMO-Bench, a new challenging dataset

[2]The paper is associated with a private dataset with a live leaderboard `https://www.vals.ai/benchmarks/proof_bench`. We contacted them several times to participate on the leaderboard without receiving a response.

[3]`https://github.com/AxiomMath/putnam2025`

that translates informal problem statements from IMO-Bench into formal Lean statements. Resources are available at `https://imobench.github.io`.

- **State-of-the-Art Results and Insights:** LEAP solves all 12 problems on Putnam 2025, and outperforms state-of-the-art baselines on Lean-IMO-Bench by a large margin. Our analysis yields the insight that the primary bottleneck in formal math for general LLMs is not formal language comprehension, but the lack of structured, iterative interaction with the proof environment. The Lean solutions generated by LEAP are available at `https://github.com/google-deepmind/superhuman/tree/main/leap` and the code is available at `https://billkunghappy.github.io/LEAP/`.

2 LEAP: Blueprint-Driven Automated Theorem Proving

2.1 Formalizing Proofs with Blueprints

Formalizing mathematical proofs is rarely a one-shot task: it requires a structured plan for progressively translating a high-level argument into Lean. To manage this complexity, recent formalization efforts often use the Lean Blueprint tool [4], which let mathematicians write a human-readable proof roadmap linked to Lean code and visualized as a directed acyclic graph (DAG), where each node represents a proof obligation. This workflow has been instrumental in coordinating large-scale projects such as the formalization roadmap for Fermat's Last Theorem [5], where a multi-year proof effort is organized through an explicit dependency graph.

Inspired by this workflow, we introduce LEAP, an agent for automated theorem proving with hierarchical decomposition and planning. Rather than synthesizing a complete proof in a single pass, LEAP incrementally drafts blueprints, decomposes Lean goals into supporting lemmas, and maintains the evolving proof plan as an AND-OR DAG.

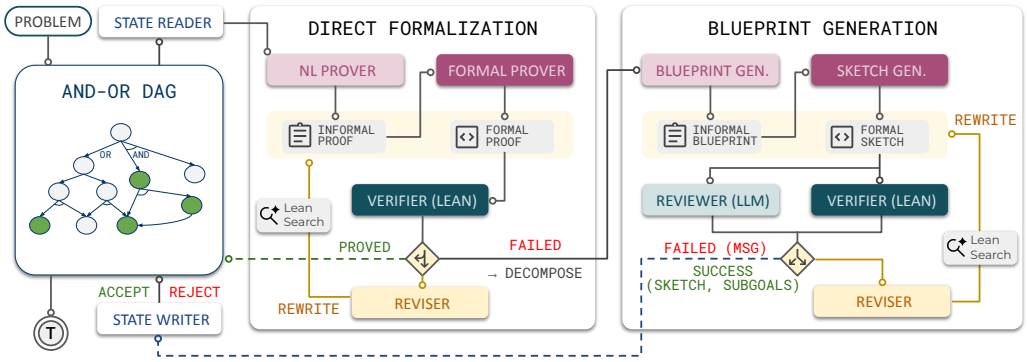

Figure 1: **LEAP workflow.** LEAP first attempts direct formalization with compiler-feedback revision and LeanSearch[Gao et al., 2024] retrieval. If this fails, it generates an informal blueprint and formal proof sketch, adding verified subgoals back to the DAG only when dependencies remain acyclic.

2.2 Overview

Figure 1 illustrates the workflow of LEAP. Given an input theorem, LEAP registers its Lean statement as the root *goal*, [6] represented as an OR node in the AND-OR DAG. To process an open goal, a *state reader* retrieves its statement, dependencies, and related lemmas. LEAP then first attempts a **direct proof** by generating an informal proof, translating it into Lean code, and checking the candidate with the Lean compiler.

If direct proving fails, LEAP shifts to **decomposition**. It first drafts an informal blueprint that proposes intermediate lemmas, then translates the blueprint into a Lean proof sketch. The sketch

[4] `https://github.com/PatrickMassot/leanblueprint`
[5] `https://leanprover-community.github.io/blog/posts/FLT-announcement/`
[6] A *goal* is any theorem or lemma statement to be proved; decomposition introduces *subgoals*. See Appendix D.

proves the current goal assuming only the proposed lemmas: the main theorem body is `sorry`-free, while `sorry` placeholders are permitted only in the newly proposed lemma statements. If the sketch is accepted by the Lean compiler, it is added as an AND node, and the proposed lemmas are added as child OR nodes. This ensures that once all child subgoals are proved, the parent goal is also proved. The verified sketch is then passed to the *state writer*, which checks that the update preserves acyclicity before committing it to the DAG. The agent then recursively processes the newly created subgoals.

This workflow relies on three tightly coupled design choices: **DAG-based hierarchical memoization**, which preserves progress and reuses lemmas across branches; **interleaved informal-formal planning**, which connects natural-language strategies with executable Lean code; and **verification-guided proof search**, which uses compiler feedback and LLM-based review to accept, revise, decompose, or abandon candidate branches.

2.3 Hierarchical Memoization via DAG

LEAP uses the AND-OR DAG not only to record proof progress, but also to structure hierarchical memoization. OR nodes represent open goals or lemma statements, each of which may be resolved by any valid proof strategy, while AND nodes represent candidate decompositions whose success depends on proving all constituent subgoals. Figure 2 illustrates this structure.

The DAG provides two central advantages. First, **monotone refinement**: once a goal is decomposed into supporting subgoals, subsequent search can focus on expanding and resolving these descendants without restructuring the established dependency order. This separates local proof exploration from global proof organization: individual proof attempts may be revised, expanded, or abandoned, while the DAG preserves the stable dependency structure of the overall proof plan. Second, **lemma memoization**: intermediate lemma statements are stored as shared proof nodes and can be reused whenever the same subproblem arises in different branches. This also supports *anticipatory lemma planning*: during blueprint generation, LEAP may propose auxiliary lemma statements that are not immediately required by the current sketch but could support later proof steps. Such prospective lemmas remain available in the graph memory without being necessary for resolving the current AND node. Together, these properties allow independent proof plans to converge on common dependencies while reducing redundant derivations.

The resulting dependency structure also improves transparency: it exposes which goals remain open, which lemmas have been resolved, and which nodes block downstream progress. This helps LEAP identify where additional lemmas, revised decompositions, or stronger assumptions may be needed, while providing an interpretable blueprint-style workspace for human-AI collaboration.

2.4 Interleaved Informal–Formal Planning

As shown in Figure 1, both the direct proof path and the blueprint decomposition path in LEAP pass through an informal proof sketch. This reflects the complementary strengths of LLMs and Lean: LLMs are effective at informal reasoning, strategy generation, and refinement, while Lean provides strict machine-checkable verification.

In direct proving, LEAP first generates an informal argument for the current goal before translating it into a candidate Lean proof. In decomposition, it drafts an informal blueprint explaining how the goal can be reduced to supporting subgoals, then converts this plan into a Lean sketch that records the proposed dependencies. In both cases, the informal sketch provides a planning space before formalization, making proof construction less brittle than direct code generation alone (see Appendix D for examples of informal proofs and blueprints).

This interleaving also makes proof progress more interpretable: each formal attempt is paired with an informal rationale, allowing users to inspect why a proof step or decomposition was proposed rather than only reading Lean code or compiler feedback.

2.5 Verification-Guided Proof Search

As shown in Figure 1, LEAP uses verification at two levels. First, the Lean compiler formally checks candidate proofs and sketches, ensuring that accepted code is syntactically valid and type-correct. For proof sketches, LEAP only permits `sorry` placeholders for the proposed subgoals (lemmas).

This preserves the AND-OR semantics of the proof DAG: once all referenced subgoals are proved, the parent goal is also proved. Second, after a blueprint proposes new subgoals, an LLM reviewer assesses the quality of the decomposition: whether the subgoals are relevant to the parent goal, make the problem easier, and offer a plausible route to completing the proof. This planning-level review is crucial for complex goals, where a Lean sketch can be syntactically valid while introducing subgoals that are ill-posed or no simpler than the original statement. Without this filter, the agent may repeatedly expand weak blueprints, spending search budget on branches that do not make real progress. We study this failure mode through an ablation without the LLM reviewer in subsection 5.3.

The LLM reviewer therefore acts as a search filter: it identifies unpromising decompositions, triggers backtracking, and encourages exploration of alternative strategies. Currently, LEAP uses a simple DFS over the DAG with backtracking. The effectiveness of this reviewer suggests a broader future direction: LLMs may also serve as heuristic evaluators for guiding search in formal proof spaces.

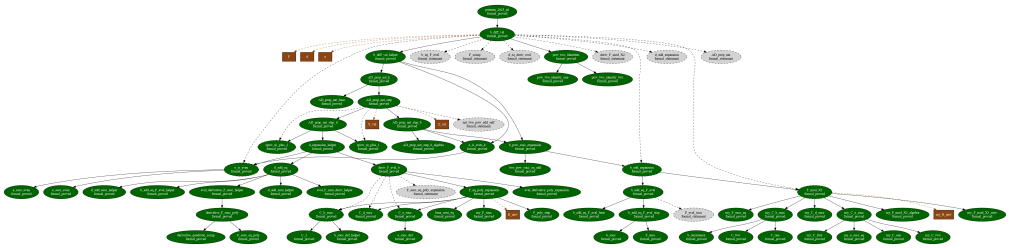

Figure 2: **DAG example for Putnam 2025 Problem A6.** LEAP decomposes the theorem into a proof sketch and supporting lemmas. Through **anticipatory lemma planning**, the agent may also propose auxiliary lemma statements that are not immediately required but could be useful later; these are shown with dashed edges and are not needed to prove the main theorem. Green nodes are proven nodes, and brown blocks denote definitions, structures, or variables introduced at a node.

3 Lean-IMO-Bench: Formalizing IMO Problems in Lean

Table 1: Baseline performance on LEAN-IMO-BENCH across three evaluation tasks. Natural Language Proof performance is based on human expert review.

Task	Model (Metric)	Basic Set (%)	Advanced Set (%)
Natural Language Proof	Gemini 2.5 Pro (Pass@1)	55.2	17.6
Formal Theorem Proving	Gemini 3.1 Pro (Pass@128)	20.0	3.3
	Gemini 3.1 Pro (Avg.@128)	4.6	0.2
Formal Proof Translation	Gemini 3.1 Pro (Pass@128)	20.0	3.3
	Gemini 3.1 Pro (Avg.@128)	4.6	0.8

3.1 Lean-IMO-Bench

We introduce LEAN-IMO-BENCH, a curated collection of 60 problems building upon the foundational work of Luong et al. [2025]. Luong et al. [2025] introduced IMO-PROOFBENCH, a rigorous suite vetted by an expert panel of mathematicians and IMO medalists. The benchmark contains 60 problems split evenly into a *Basic* set and an *Advanced* set of 30 problems each. The *Basic* set spans pre-IMO to IMO-Medium difficulty and includes 8 algebra, 8 combinatorics, 8 number theory, and 6 geometry problems. The *Advanced* set includes novel problems up to IMO-Hard difficulty, with 8 algebra, 8 combinatorics, 6 number theory, and 8 geometry problems. Overall, the benchmark is approximately balanced across algebra, combinatorics, geometry, and number theory.

To ensure the highest level of accuracy in LEAN-IMO-BENCH, Lean experts manually formalized and verified all 60 problem statements. Because these problems are at the IMO level, the required mathematical background is elementary. Consequently, we expect the corresponding Lean solutions to be concise, deliberately removing the overhead associated with formalizing complex, modern mathematical theories.

The dataset can be used to evaluate models across three distinct tasks: **Natural Language Proof**, **Formal Theorem Proving**, and **Formal Proof Translation**, while we focus on formal theorem proving in this paper. The baseline performance on LEAN-IMO-BENCH is summarized in Table 1.

For the Natural Language Proof task, we cite Luong et al. [2025] as reference: Gemini 2.5 Pro shows strong informal reasoning performance. However, as shown in Table 1, this does not directly translate to formal theorem proving: Gemini 3.1 Pro performs substantially worse on Formal Theorem Proving, especially on the Advanced set. Providing a correct informal proof in the Formal Proof Translation task also yields little improvement, with Pass@128 unchanged and only a marginal gain in Average@128.

Table 1 demonstrates a stark gap in the models' Lean capabilities. Because the model can already successfully solve these problems in natural language, mathematical reasoning is not the bottleneck, thus reliably generating valid Lean code remains the primary challenge.

4 Experimental Results

We evaluate LEAP with Gemini-3.1-pro as the backend large language model and compare it against five baselines: **Gemini-3.1-pro**, which tests one-shot proof generation by a strong general-purpose model; **Goedel-Prover-V2-32B** [Lin et al., 2026], a state-of-the-art open-source ATP model specialized for Lean; **Hilbert** [Varambally et al., 2025], an agentic Lean formalization framework that combines Goedel-Prover-V2-32B with Gemini-3.1-pro; **Aristotle** [Achim et al., 2025], a specialized automated theorem-proving system with dedicated ATP components that achieved gold-medal-level performance at the 2025 IMO; and **Numina-Lean-Agent** [Liu et al., 2026], a general-purpose coding agent for Lean.

For the headline Putnam 2025 and Lean-IMO-Bench evaluations, LEAP receives only the Lean statements, without reference informal proofs; the Formal Proof Translation baseline in Table 1 does receive reference informal proofs. We evaluate formal proving ability on two datasets: **Putnam 2025** and our proposed **Lean-IMO-Bench**. Putnam 2025 contains twelve undergraduate-level problems from the 86th William Lowell Putnam Mathematical Competition,[7] a highly challenging North American mathematics competition. In the 2025 competition, the top score was 110 out of 120, the average score was approximately 10, and the median score was 2.

4.1 Results on Putnam 2025

Table 2 presents the Putnam 2025 results. The direct formalization baselines, Gemini-3.1-Pro and Goedel-Prover-V2-32B, fail to solve any problems even at Pass@128, highlighting the difficulty of generating complete, correct proofs without iterative feedback. Agentic methods perform substantially better, demonstrating the value of multi-step proof construction and verification.

Table 2: Putnam 2025 results. Checkmarks indicate solved problems. $^{\diamond}$ denotes Pass@128; † denotes two rollouts. ‡ denotes results reported by the original systems.

Method	a1	a2	a3	a4	a5	a6	b1	b2	b3	b4	b5	b6	Solve Rate (%)
Gemini-3.1-Pro$^{\diamond}$	×	×	×	×	×	×	×	×	×	×	×	×	0.0
Goedel-Prover-V2-32B$^{\diamond}$	×	×	×	×	×	×	×	×	×	×	×	×	0.0
Hilbert†	×	✓	✓	✓	×	×	×	×	×	×	×	✓	33.3
Aristotle†	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	75.0
Numina-Lean-Agent‡	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	100.0
Axiom‡	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	100.0
LEAP†	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	**100.0**

The open-source agentic framework Hilbert solves 4/12 problems, improving considerably over direct generation. Its recursive proof search, however, may repeatedly revisit related subgoals and incur substantial redundant work; we allowed up to seven days per rollout in our evaluation. The proprietary Aristotle system solves 9/12 under two rollouts. Numina-Lean-Agent [Liu et al., 2026]

[7] Mathematical Association of America, *Results of the 86th William Lowell Putnam Mathematical Competition*.

and Axiom also report solving all 12 problems, establishing that perfect Putnam 2025 coverage is achievable by multiple agentic systems.

LEAP likewise solves all 12 problems under two rollouts. Unlike an isolated recursive search, its persistent AND–OR DAG organizes alternative proof sketches, verified subgoals, and reusable intermediate lemmas across the search. This design supports the structured, long-horizon proof search studied in our ablations (subsection 5.2), while the Putnam result demonstrates competitive end-to-end proving capability.

Efficiency comparison. With all three systems achieving 12/12, Table 3 compares their reported aggregate runtimes: 822 minutes for LEAP, 2,682 for Axiom, and 3,889 for Numina-Lean-Agent. Token counts are included for reference, but different models and system settings prevent a controlled architecture-level comparison.

Table 3: Putnam 2025 system-level efficiency comparison, using reported aggregate runtime across the 12 solved problems. Token counts are contextual and not directly comparable across systems; N/R denotes not reported.

System	Solved	Runtime (min)	Reported tokens
Numina-Lean-Agent [Liu et al., 2026]	12/12	3,889	N/R
Axiom	12/12	2,682	105.2M
LEAP	12/12	**822**	28M

4.2 Results on Lean-IMO-Bench

Table 4 presents evaluation results on Lean-IMO-Bench. We include this dataset to test model robustness across a broader spectrum of mathematical disciplines and distinct complexity tiers, providing a complementary challenge to the Putnam benchmark.

Direct formalization baselines (Gemini-3.1-Pro and Goedel-Prover-V2-32B) and the open-source Hilbert framework struggle significantly on this dataset, exhibiting severe performance degradation on the Advanced set. While the proprietary Aristotle system resolves a majority of the Basic problems, its effectiveness drops sharply as complexity increases. Notably, across all evaluated methods, performance in the Geometry category remains near zero. This aligns with the well-established difficulty of formalizing olympiad-level geometry in Lean without the aid of supplementary, domain-specific frameworks. We retain this category strictly to evaluate general-purpose reasoning under extreme formalization constraints.

Against these baselines, including Numina-Lean-Agent in its canonical Opus 4.5 configuration (76.7% Basic / 43.3% Advanced), LEAP achieves the highest overall solve rates, scoring 83.3% on the Basic set and 56.7% on the Advanced set. By effectively leveraging its DAG-based architecture, LEAP demonstrates strong domain generalization, maintaining a 100% solve rate in both Algebra and Number Theory regardless of the difficulty tier.

5 Discussion

5.1 Beyond One-Shot Formalization

A central motivation of LEAP is that general foundation models can be effective iterative formalizers, even when they are not specialized Lean provers. While specialized provers are trained for formal proof synthesis, general models offer complementary capabilities such as instruction following, long-context reasoning, informal planning, tool use, and feedback-based revision.

To isolate this effect, we evaluate the *Direct Formalization* component labeled in Figure 1 under two settings. In the one-shot setting, each model is evaluated with Pass@128 over independently sampled proof attempts. In the iterative setting, each model receives a single initial attempt and up to 20 compiler-feedback revision steps, yielding a Pass@1 result under a smaller sampling budget. As shown in Table 5, Goedel-Prover-V2-32B does not benefit from this feedback loop, while Gemini-3.1-pro improves substantially from 20.0% to 36.6%.

Table 4: Results on LEAN-IMO-BENCH. We report the solve rate (%) across different mathematical categories for the **Basic** and **Advanced** sets separately. Evaluation settings: $^{\diamond}$ indicates pass@128, while † indicates rollout=2. Numina uses Opus 4.5 with Gemini 3.1 Pro as the informal discussion partner; an adapted all-Gemini variant achieves 43.3%/23.3% overall on Basic/Advanced. Best results in each section are in bold.

Method	Algebra	Comb.	Num. Theory	Geometry	**Overall**
Basic Set (%)					
Gemini-3.1-Pro$^{\diamond}$	37.5	12.5	25.0	0	20.0
Goedel-V2-32B$^{\diamond}$	37.5	0	0	0	10.0
Hilbert†	62.5	25	50	0	36.6
Aristotle†	75	**100**	100	**16.7**	76.7
Numina (Opus 4.5)†	87.5	**100**	87.5	**16.7**	76.7
LEAP†	**100**	**100**	**100**	**16.7**	**83.3**
Advanced Set (%)					
Gemini-3.1-Pro$^{\diamond}$	0	12.5	0	0	3.3
Goedel-V2-32B$^{\diamond}$	0	0	0	0	0
Hilbert†	12.5	0	16.6	0	6.6
Aristotle†	37.5	12.5	33.3	0	20.0
Numina (Opus 4.5)†	87.5	**25**	66.6	0	43.3
LEAP†	**100**	**25**	**100**	**12.5**	**56.7**

This suggests that iterative formalization depends on capabilities beyond local Lean proof synthesis. Interpreting compiler errors, maintaining context, and revising proof attempts over multiple steps can be as important as one-shot formal proving accuracy. These results support using a general foundation model as the reasoning backbone of LEAP, while leaving open the possibility of combining it with specialized provers for local proof generation.

Table 5: **One-shot** vs. **iterative formalization** performance on Lean-IMO-Bench Basic set.

Model	**One-shot (Pass@128)**	**Iterative (Pass@1)**
Goedel-Prover-V2-32B	10.0	6.6
Gemini-3.1-Pro	20.0	36.6

5.2 Effect of DAG-Based Memoization

LEAP maintains proof progress as a DAG-based memory rather than an isolated decomposition tree. This allows intermediate lemmas to be stored as shared nodes and reused across branches, while exposing graph context such as existing goals, dependencies, and previously proposed lemmas.

To evaluate this design, we compare LEAP with a tree-structured variant that follows the same workflow but removes global lemma sharing. As shown in Table 6, the tree variant already substantially outperforms Hilbert [Varambally et al., 2025], which achieves 36.6% and 6.6% on the Basic and Advanced sets, respectively (Table 4). This indicates that interleaved informal–formal planning and verification-guided search are effective even without DAG-based memoization. The full DAG version further improves performance from 73.3% to 83.3% on the Basic set and from 40.0% to 56.7% on the Advanced set, showing the benefit of global proof memory.

The improvement is especially pronounced on harder categories, such as Advanced Algebra and Advanced Number Theory, where shared lemmas and graph context are more likely to matter. We attribute this gain to two effects. First, the DAG supports anticipatory lemma planning: auxiliary lemmas proposed at higher-level nodes can later be reused by downstream subgoals (Figure 2). Second, repeated subproblems can be shared across branches, avoiding the need to rediscover or reprove the same lemma multiple times. Together, these properties reduce redundant derivations and improve proof search efficiency.

Table 6: **DAG memoization ablation.** Solve rate (%) by category on Lean-IMO-Bench Basic (B)/Advanced (A) sets.

Config.	Alg.		Comb.		NT		Geo.		Overall	
	B	A	B	A	B	A	B	A	B	A
w/o DAG (Naive Tree)	100	75	75	25	100	66.6	0	0	73.3	40.0
Full DAG	100	100	100	25	100	100	16.7	12.5	83.3	56.7

```
-- Grandparent goal
theorem perms_a_equiv_b_card
  (m : ℕ) (s : Fin (m + 1) → ℤˣ) (k : Fin (m + 2)) :
  Nat.card (PermsA m s k) = Nat.card (PermsB m s k) := by
  rw [← card_perms_a_eq m s k, ← card_perms_b_eq m s k]
  exact valid_perms_end_eq_sigma_card m s k

-- Intermediate lemma (unfolding the definitions)
lemma valid_perms_end_eq_sigma_card
  (m : ℕ) (s : Fin (m + 1) → ℤˣ) (k : Fin (m + 2)) :
  (ValidPermsEnd (m + 1) s k).card = (Finset.sigma ...).card := by
  rw [card_perms_a_eq m s k, card_perms_b_eq m s k]
  exact card_perms_a_eq_card_perms_b m s k

-- Proposed subgoal (syntactically identical to the grandparent goal)
lemma card_perms_a_eq_card_perms_b
  (m : ℕ) (s : Fin (m + 1) → ℤˣ) (k : Fin (m + 2)) :
  Nat.card (PermsA m s k) = Nat.card (PermsB m s k) := by
  sorry
```

Figure 3: **Unproductive decomposition without LLM review.** The proposed subgoal restates the grandparent goal, so the decomposition is formally admissible but does not simplify proof search.

5.3 Toward LLM-Guided Proof Search

Compiler verification checks whether a proof sketch is formally well-typed, but not whether its decomposition is useful. A sketch may prove the parent goal from proposed lemmas that are unhelpful, overly difficult, or nearly equivalent to the original goal. In LEAP, the LLM reviewer acts as a local search heuristic: it filters candidate decompositions by judging whether they meaningfully simplify the parent goal before they are committed to the DAG.

We focus this ablation on Putnam 2025 Problem A5 because it is one of the most challenging cases in our evaluation, requiring the longest runtime and two rollouts for LEAP to formalize the proof successfully. Removing the LLM-based decomposition reviewer causes the agent to fail even after eight rollout attempts. This contrast suggests that local LLM review provides a useful search signal: it rejects weak decompositions early, triggers backtracking, and prevents the agent from spending rollouts on branches that do not make substantive progress. We further inspect decomposition traces from the ablated setting; a representative failure case is shown in Figure 3.

The decomposition in Figure 3 is formally admissible but does not simplify the mathematical state. The agent first unfolds the definitions in the grandparent goal to create an intermediate lemma, then folds them back into a proposed subgoal that is syntactically identical to the original statement. Without semantic review, this duplicate lemma is treated as a new step, causing the agent to repeat the same unproductive decomposition until its search budget is exhausted. This failure highlights the potential of LLM-guided proof search: a reviewer can assess whether a proposed lemma actually advances the proof, prune cyclic or non-simplifying branches, and direct compute toward more promising paths.

5.4 Perspective: General LLMs as Formal Provers: From Zero to Hero

As demonstrated by LEAP, the seemingly insurmountable gap between the poor one-shot theorem proving performance of general LLMs and state-of-the-art results can be effectively bridged by a well-

designed agentic framework. By shifting the paradigm away from relying solely on small specialized LLMs, we show that the extensive knowledge, instruction following, and self-correction capabilities of foundation models are more than sufficient. When scaffolded correctly, these foundation models can progress from near zero formal math performance to solving highly complex problems.

While small specialized LLMs lack the overarching agentic capabilities of their foundation counterparts, we acknowledge that they still hold value. A hybrid architecture combining the high-level, structural reasoning of a foundation model with the focused, formal step generation of a fine-tuned specialized model could be a highly effective design pattern. However, exploring this hybrid approach remains outside the scope of this paper, as our primary goal is to highlight the standalone power of general purpose LLMs in an agentic workflow.

6 Case Studies: Research-Level Formalization

Hamiltonian Decomposition of Directed Cayley Graphs. To evaluate LEAP on a highly complex mathematical task, we targeted a recently solved open problem in combinatorics: the Hamiltonian decomposition of the directed Cayley graph $\Gamma_m = Cay(\mathbb{Z}_m^3, \{e_1, e_2, e_3\})$ for even m. Originally posed by Donald Knuth, the problem asks whether the graph's directed arcs can be partitioned into exactly three distinct, spanning Hamiltonian cycles . The informal mathematical proof for the even-case construction is exceptionally intricate, relying on heavy combinatorial analysis and localized defect routing across different layers of the graph. We focused our formalization efforts on a critical subproblem: rigorously verifying that the 2D planar projection of a single color class's routing dynamics forms an unbroken mathematical cycle of length m^2. The informal arguments for this specific dynamic span roughly 20 pages of dense piecewise maps, parity-dependent intervals, and complex cross-row transitions. To tackle a formal verification of this magnitude, we deployed LEAP, which successfully decomposed the monolithic informal proof into a granular, highly structured proof graph. By autonomously and systematically resolving the interdependent nodes of this graph, LEAP managed to fully verify the complex cycle-merging dynamics, ultimately synthesizing over 5000 lines of rigorous Lean 4 code to complete the formal proof for this subproblem. Full problem descriptions and informal proofs are available at `https://github.com/dpwoodru/knuthCycles/tree/main`.

Formalizing Erdős Problem 457. We further tested LEAP on Erdős Problem 457, a previously resolved number-theory problem concerning prime divisibility in products of consecutive integers. Given an LLM-generated reconstruction of the known informal proof, LEAP translated and refined the argument into a mechanically verified Lean 4 proof without manual formal-proof engineering.

We additionally tested LEAP with Gemini-3.1-Pro on research-level Erdős problems previously addressed by AlphaProof Nexus [Tsoukalas et al., 2026]. For the latest progress on open problems, see `https://billkunghappy.github.io/LEAP/`.

Formal statements and detailed problem descriptions for the case studies above are provided in Appendix C.

7 Conclusion and Future Work

The success of LEAP underscores a broader implication: modern general LLMs already possess the strong reasoning capabilities required to tackle rigorous, domain specific challenges, provided they are embedded in an appropriate structural design. In the realm of formal mathematics, this naturally takes the form of logical problem decomposition. By utilizing the language model's reasoning ability to break down complex theorems into smaller, manageable sub-problems, and strictly verifying these steps via the Lean compiler, we create a robust pathway to mathematical truth.

Moving forward, a critical challenge will be optimizing how we navigate the resulting proof trees. As problems are decomposed into increasingly intricate sub-tasks, the search space grows rapidly. Future work must focus on how to efficiently search through this space, exploring methods to guide the agent's focus and manage the complexity of large scale proof generation. Mastering this efficient exploration will be the key to unlocking even more complex mathematical milestones.

Acknowledgments and Disclosure of Funding

We thank Michael P. Brenner, Honghao Lin, David Woodruff, Vahab Mirrokni for providing the informal proof of the even-case of Knuth's Cycles problem. We would also like to thank Ashley Aragorn Khoo, Paul Lezeau, Calle Sönne, and Moritz Firsching for formalizing the Lean problem statements in Lean-IMO-Bench.

References

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Mario Carneiro. Metamath zero: The cartesian theorem prover, 2020. URL https://arxiv.org/abs/1910.10703.

Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, Zhihong Wang, Mingxuan Wang, Chenrui Wei, Shufa Wei, Huajian Xin, Fan Yang, Weihao Gao, Zheng Yuan, Tianyang Zhan, Zeyu Zheng, Tianxi Zhou, and Thomas Hanwen Zhu. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience, 2025a. URL https://arxiv.org/abs/2512.17260.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Yuchen Wu, Yihang Xia, Huajian Xin, Fan Yang, Huaiyuan Ying, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang, Ge Zhang, Tianyun Zhao, Jianqiu Zhao, Yichi Zhou, and Thomas Hanwen Zhu. Seed-prover: Deep and broad reasoning for automated theorem proving, 2025b. URL https://arxiv.org/abs/2507.23726.

Tony Feng, Junehyuk Jung, Sang hyun Kim, Carlo Pagano, Sergei Gukov, Chiang-Chiang Tsai, David Woodruff, Adel Javanmard, Aryan Mokhtari, Dawsen Hwang, Yuri Chervonyi, Jonathan N. Lee, Garrett Bingham, Trieu H. Trinh, Vahab Mirrokni, Quoc V. Le, and Thang Luong. Aletheia tackles firstproof autonomously, 2026a. URL https://arxiv.org/abs/2602.21201.

Tony Feng, Trieu Trinh, Garrett Bingham, Jiwon Kang, Shengtong Zhang, Sang hyun Kim, Kevin Barreto, Carl Schildkraut, Junehyuk Jung, Jaehyeon Seo, Carlo Pagano, Yuri Chervonyi, Dawsen Hwang, Kaiying Hou, Sergei Gukov, Cheng-Chiang Tsai, Hyunwoo Choi, Youngbeom Jin, Wei-Yuan Li, Hao-An Wu, Ruey-An Shiu, Yu-Sheng Shih, Quoc V. Le, and Thang Luong. Semi-autonomous mathematics discovery with gemini: A case study on the erdős problems, 2026b, 2026b. URL https://arxiv.org/abs/2601.22401.

Tony Feng, Trieu H. Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung, Joonkyung Lee, Carlo Pagano, Sang hyun Kim, Federico Pasqualotto, Sergei Gukov, Jonathan N. Lee, Junsu Kim, Kaiying Hou, Golnaz Ghiasi, Yi Tay, YaGuang Li, Chenkai Kuang, Yuan Liu, Hanzhao Lin, Evan Zheran Liu, Nigamaa Nayakanti, Xiaomeng Yang, Heng-Tze Cheng, Demis Hassabis, Koray Kavukcuoglu, Quoc V. Le, and Thang Luong. Towards autonomous mathematics research, 2026c. URL https://arxiv.org/abs/2602.10177.

Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241, 2023.

Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. A semantic search engine for mathlib4. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8001–8013, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.470. URL https://aclanthology.org/2024.findings-emnlp.470/.

Christian Greiffenhagen. Checking correctness in mathematical peer review. *Social studies of science*, 54(2): 184–209, 2024.

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

Thomas Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. *Forum of mathematics, Pi*, 5, 2017.

Thomas C Hales. A proof of the kepler conjecture. *Annals of mathematics*, 162(3):1065–1185, 2005.

John Harrison. Hol light: An overview. In *International Conference on Theorem Proving in Higher Order Logics*, pages 60–66. Springer, 2009.

Yichen Huang and Lin F. Yang. Winning gold at IMO 2025 with a model-agnostic verification-and-refinement pipeline. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=svz4xDjRC1.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi: 10.1038/s41586-025-09833-y. URL https://www.nature.com/articles/s41586-025-09833-y.

Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant a tutorial. *Rapport Technique*, 178:113, 1997.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Haocheng Ju, Guoxiong Gao, Jiedong Jiang, Bin Wu, Zeming Sun, Shurui Liu, Leheng Chen, Yutong Wang, Yuefeng Wang, Zichen Wang, Wanyi He, Peihao Wu, Liang Xiao, Ruochuan Liu, Bryan Dai, and Bin Dong. Automated conjecture resolution with formal verification, 2026. URL https://arxiv.org/abs/2604.03789.

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 26337–26349. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a8901c5e85fb8e1823bbf0f755053672-Paper-Conference.pdf.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-STar: Learning to interleave thinking and proving. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SOWZ59UyNc.

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. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j4C0nALrgK.

Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, Lihong Zhi, Jia Li, and Wenda Li. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. 2026. URL https://arxiv.org/abs/2601.14027.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6774–6786, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.528. URL https://aclanthology.org/2021.acl-long.528/.

Thang Luong and Edward Lockhart. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad. Google DeepMind Blog, July 2025. URL https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/. Accessed: 2026-05-02.

Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu Hoang Trinh, Quoc V Le, and Junehyuk Jung. Towards robust mathematical reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35418–35442, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1794. URL https://aclanthology.org/2025.emnlp-main.1794/.

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, page 367–381. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

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

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Nikil Ravi, Kexing Ying, Vasilii Nesterov, Rayan Krishnan, Elif Uskuplu, Bingyu Xia, Janitha Aswedige, and Langston Nashold. Formalproofbench: Can models write graduate level math proofs that are formally verified? In *ICLR 2026 Workshop: VerifAI-2: The Second Workshop on AI Verification in the Wild*, 2026. URL https://openreview.net/forum?id=1LdnVndCEG.

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. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL https://arxiv.org/abs/2504.21801.

Alexander K Taylor, Junyi Zhang, Ethan Ji, Vigyan Sahai, Haikang Deng, Yuanzhou Chen, Yifan Yuan, Di Wu, Jia-Chen Gu, Kai-Wei Chang, Nanyun Peng, Amit Sahai, and Wei Wang. Taobench: Do automated theorem prover llms generalize beyond mathlib?, 2026. URL https://arxiv.org/abs/2603.12744.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=V7HRrxXUhN.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 11545–11569. Curran Associates, Inc., 2024. doi: 10.52202/079017-0368. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1582eaf9e0cf349e1e5a6ee453100aa1-Paper-Datasets_and_Benchmarks_Track.pdf.

George Tsoukalas, Anton Kovsharov, Sergey Shirobokov, Anja Surina, Moritz Firsching, Gergely Bérczi, Francisco JR Ruiz, Arun Suggala, Adam Zsolt Wagner, Eric Wieser, et al. Advancing mathematics research with ai-driven formal proof search. *arXiv preprint arXiv:2605.22763*, 2026.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=ljAHonPrs1.

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

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in neural information processing systems*, 35:32353–32368, 2022.

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

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. BFS-prover: Scalable best-first tree search for LLM-based automatic theorem proving. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32588–32599, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1565. URL https://aclanthology.org/2025.acl-long.1565/.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21573–21612. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4441469427094f8873d0fecb0c4e1cee-Paper-Datasets_and_Benchmarks.pdf.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A Limitations

LEAP is an initial exploration of using general-purpose LLMs for formal mathematical proving with structured proof memory. Our goal is to study how a strong general model can benefit from an explicit AND-OR DAG that organizes goals, decompositions, and reusable lemmas.

There are several natural directions for extending this work. First, LEAP currently uses a simple depth-first search strategy over the DAG. This keeps the search procedure transparent and allows us to isolate the effect of DAG-based memoization, but more informed search policies could further improve efficiency. For example, future work could explore learned node selection, decomposition ranking, or LLM-guided search heuristics.

Second, the main experiments use Gemini-3.1-Pro. In brief single evaluations on the Lean-IMO-Bench Basic set, LEAP achieves 90.0% with Opus 4.5 and Gemini 3.1 Pro as a discussion partner, and 86.6% with Gemini 3.5 Flash, compared with 83.3% in the main setting. These results do not establish performance on the Advanced set or across repeated runs.

Finally, LEAP focuses on proof planning with general LLMs rather than exhaustive optimization of every ATP component. This leaves room for complementary improvements, such as stronger retrieval, tighter integration with specialized Lean provers, and more advanced proof-state prioritization. We view these directions as promising extensions of the framework.

B Related Work

The intersection of machine learning and formal mathematics has seen significant growth, driven by the development of powerful language models and the availability of formalized benchmarks.

Neural Theorem Proving Early work in neural theorem proving mainly utilize in-house symbolic engines, such as Metamath [Polu and Sutskever, 2020], MM0 [Carneiro, 2020] or some dedicated formal language for geometry problems [Lu et al., 2021]. Later work such as mathlib [mathlib Community, 2020], LeanDojo [Yang et al., 2023] and MiniF2F [Zheng et al., 2022] pioneered the use of LLMs for generative theorem proving in Lean. They serve as pillar that provide a rich library of known theorems, an interactive environment for step-level search and a descent-level of difficulty evaluation set. To manage the large search space, HyperTree Proof Search [Lample et al., 2022] and

related Monte Carlo tree search methods [Lin et al., 2025, Xin et al., 2025b] have been explored. While search-based methods operate at the tactic level, Baldur [First et al., 2023] and DeepSeek-prover-v1.5 [Xin et al., 2025a] explored whole-proof generation, attempting to produce a complete proof in a single step. Another promising direction is guiding formal proof search with informal proofs or sketches. The "draft, sketch, and prove" methodology [Jiang et al., 2023] demonstrated that using an informal proof as a blueprint can significantly guide and improve formal theorem proving. Our work, LEAP, builds on this intuition by utilizing general LLMs to generate informal blueprints and iteratively refine formal proofs based on compiler feedback, but without relying on specialized fine-tuned models for the formalization step.

Specialized Prover Models Recent breakthroughs have often relied on extensive fine-tuning of large models on formal mathematical corpora. Representative works include AlphaProof [Hubert et al., 2026], DeepSeek Prover V2 [Ren et al., 2025], Seed Prover [Chen et al., 2025b], Kimina Prover [Wang et al., 2025] and Goedel Prover V2 [Lin et al., 2026]. These models achieve state-of-the-art performance by scaling up training and search on formal systems. However, they require substantial computational resources for training and are highly specialized for specific formal languages. In contrast, LEAP demonstrates that general-purpose LLMs, when placed in a proper agentic environment, can achieve competitive performance without such specialized fine-tuning.

General-Purpose Agentic Provers COPRA [Thakur et al., 2024] uses a general-purpose LLM with feedback-driven tactic search; Numina-Lean-Agent [Liu et al., 2026] uses a flexible coding agent with Lean tools, and Archon [Ju et al., 2026] formalizes arguments with agentic proof search. LEAP instead studies an explicit, persistent AND–OR proof graph for online, goal-directed decomposition and lemma reuse.

Auto-Formalization Auto-formalization, the task of translating natural language mathematics into formal statements and proofs, is a critical bridge between informal and formal reasoning. Early work relied on neural machine translation techniques [Wu et al., 2022]. More recently, LLMs have been used to generate formal statements for training provers at scale, as seen in the auto-formalization pipeline of AlphaProof [Hubert et al., 2026]. LEAP utilizes the strong auto-formalization capabilities of general LLMs within its agentic harness to bridge the gap between informal blueprints and formal proofs.

Mathematical Reasoning with LLMs Large Language Models have shown impressive progress in solving natural-language mathematical problems, demonstrating strong capabilities in complex reasoning. Recent advancements, such as OpenAI o1 [Jaech et al., 2024] and DeepSeek R1 [Guo et al., 2025], have demonstrated the effectiveness of scaling reinforcement learning for complex mathematical tasks, achieving high scores on competitive benchmarks like AIME. However, direct evaluation of these models on formal theorem proving benchmarks often yields low solve rates, highlighting the gap between informal reasoning and formal verification. LEAP addresses this by leveraging the strong informal reasoning and instruction-following capabilities of general LLMs within an agentic harness, enabling them to interact with the Lean compiler and iteratively self-correct, thus bridging the formalization gap without specialized fine-tuning.

C Problem Statements

We present the LEAN statements of the open problems that we tested with LEAP.

Hamiltonian Decomposition of Directed Cayley Graphs The Hamiltonian decomposition problem for the directed Cayley graph $\Gamma_m = Cay(\mathbb{Z}_m^3, \{e_1, e_2, e_3\})$ asks whether its edges can be partitioned into three distinct directed Hamiltonian cycles. For the even-case construction ($m = 2h \geq 10$), the 3D routing dynamics of individual color classes can be analytically projected onto a 2D planar "round map" defined on a $\mathbb{Z}_m \times \mathbb{Z}_m$ grid. The formal statement below encodes the exact operational semantics for the Color 2 subgraph—including its parity-dependent structural defects, coordinate shifts, and piecewise transitions—and asserts that its round map forms a single, unbroken cycle of length m^2.

Lean Statement for the Hamiltonian Decomposition of Directed Cayley Graphs

```
import Mathlib

set_option autoImplicit false

variable (h : ℕ) (hh : 5 ≤ h)

abbrev Fiber2 (h : ℕ) := Fin (2 * h) × Fin (2 * h)

-- 1. Base Coordinate Definitions
def one2 : Fin (2 * h) := ⟨1, by omega⟩
def mMinusOne2 : Fin (2 * h) := ⟨2 * h - 1, by omega⟩
def mMinusTwo2 : Fin (2 * h) := ⟨2 * h - 2, by omega⟩

def succ2c (x : Fin (2 * h)) : Fin (2 * h) := x + one2 h hh
def pred2c (x : Fin (2 * h)) : Fin (2 * h) := x - one2 h hh

-- 2. Exceptional Set Logic (Defects)
def y2SwitchRow (x : Fin (2 * h)) : Prop :=
  x.val = h + 1 ∨ x.val = h + 2 ∨ x.val = h + 3

instance (x : Fin (2 * h)) : Decidable (y2SwitchRow h x) := by
  unfold y2SwitchRow
  infer_instance

def y2star (x : Fin (2 * h)) : Fin (2 * h) :=
  if y2SwitchRow h x then
    if h % 2 = 0 then mMinusTwo2 h hh else mMinusOne2 h hh
  else
    ⟨2 * h - 1 - x.val, by omega⟩

def A2 (x : Fin (2 * h)) : Fin (2 * h) :=
  succ2c h hh (y2star h hh x)

def activeB2 (x y : Fin (2 * h)) : Prop :=
  if h % 2 = 0 then
    (x.val = h + 1 ∧ y.val ≤ h - 1) ∨
      (x.val = h + 4 ∧ h - 3 ≤ y.val ∧ y.val ≤ 2 * h - 2)
  else
    (x.val = h + 1 ∧ 1 ≤ y.val ∧ y.val ≤ h - 1) ∨
      (x.val = h + 4 ∧ h - 3 ≤ y.val)

instance (x y : Fin (2 * h)) : Decidable (activeB2 h x y) := by
  unfold activeB2
  infer_instance

-- 3. The Round Map
def r2Map (p : Fiber2 h) : Fiber2 h :=
  let x := p.1
  let u := pred2c h hh p.2
  if u = A2 h hh x then
    (succ2c h hh x,
      if x.val = h + 1 ∨ x.val = h + 2 then u else pred2c h hh u)
  else if activeB2 h x u then
    (x, pred2c h hh u)
  else
    (x, u)

-- 4. The Self-Contained Goal
/-- The unrolled Hamiltonicity goal for the Color 2 round map. -/
theorem color2_singleCycle_unrolled (h6 : 6 ≤ h) :
    (∀ p : Fin (2 * h) × Fin (2 * h), (r2Map h hh)^[(2 * h) * (2 * h)] p = p) ∧
    (∀ (p : Fin (2 * h) × Fin (2 * h)) (k : ℕ), 0 < k → k < (2 * h) * (2 * h) → (
     r2Map h hh)^[k] p ≠ p) := by
  sorry
```

Erdős 457 Erdős Problem 457 is a number theory challenge concerning the prime divisors of consecutive integers. Specifically, it conjectures the existence of a real number $\varepsilon > 0$ such that for infinitely many integers n, every prime number $p \leq (2+\varepsilon)\log n$ divides the product of the $\lfloor \log n \rfloor$ consecutive integers starting from $n+1$. The Lean formalization below captures this exact asymptotic prime divisibility condition.

Lean Statement for Erdős Problem 457

```
import Mathlib

theorem erdos_457 : ∃ ε > (0 : ℝ),
    { (n : ℕ) | ∀ (p : ℕ), p ≤ (2 + ε) * Real.log n → p.Prime →
      p ∣ ∏ i ∈ Finset.Icc 1 ⌊Real.log n⌋₊, (n + i) }.Infinite := by
  sorry
```

D Proof Contexts and Artifacts

This section describes the formal and informal artifacts used by LEAP during proof planning. Formal artifacts correspond to Lean-level objects that are checked by the compiler or represented in the proof DAG, while informal artifacts correspond to natural-language planning objects used to guide direct proving and decomposition.

Formal context. A *proof goal* is a Lean theorem or lemma statement that remains to be proved. The original input theorem is the root proof goal, while lemma statements introduced by decomposition become subgoals in the proof DAG. A *formal proof* is a complete Lean proof of the current proof goal that does not rely on newly proposed unproven lemmas; once accepted by the Lean compiler, the corresponding goal is marked as resolved. A *proof sketch* is a Lean artifact that proves the current goal assuming a set of proposed lemmas. In LEAP, a proof sketch may contain `sorry` placeholders only for these explicitly proposed lemmas. Thus, a verified proof sketch defines a valid decomposition: once all referenced proposed lemmas are later proved, the current goal is also proved. We present examples of these context using Lean-IMO-Bench Problem 001 and 009 in the Basic Set. **All artifacts, except for the Proof Goal of the root problem, are created by LEAP automatically.)**

Example Proof Goal (Lean-IMO-Bench, Basic 001)

```
theorem PBBasic001 : {f : ℤ → ℤ | ∀ x y, f (2 * x) + 2 * f y = f (f (x + y))}
      = {0} ∪ {(fun x ↦ 2 * x + c)| (c : ℤ)} :=
by sorry
```

Example Formal Proof (Lean-IMO-Bench, Basic 001)

```
import Mathlib

theorem PBBasic001 : {f : ℤ → ℤ | ∀ x y, f (2 * x) + 2 * f y = f (f (x + y))}
      = {0} ∪ {(fun x ↦ 2 * x + c)| (c : ℤ)} :=
by
  ext f
  simp only [Set.mem_setOf_eq, Set.mem_union, Set.mem_singleton_iff, Set.mem_range]
  constructor
  · intro h
    have h1 : ∀ y, f (f y) = 2 * f y + f 0 := by
      -- [Proof details omitted for brevity]
    have h2 : ∀ x, f (2 * x) = 2 * f x - f 0 := by
      -- [Proof details omitted]
    have h3 : ∀ x y, f (x + y) = f x + f y - f 0 := by
      -- [Proof details omitted]

    have hc : ∃ c, c = f 0 := ⟨f 0, rfl⟩
    rcases hc with ⟨c, hc_eq⟩
    have hk : ∃ k, k = f 1 - c := ⟨f 1 - c, rfl⟩
    rcases hk with ⟨k, hk_eq⟩

    -- ... [Induction steps for h4 omitted] ...

    have h5 : ∀ x : ℤ, f x = k * x + c := by
      -- ... [Negative cases to prove linear form omitted] ...

    have eq_all : ∀ x y : ℤ, k * (2 * x) + c + 2 * (k * y + c) = k * (k * (x + y) + c)
      + c := by
      intro x y
      have h_orig := h x y
      simp only [h5] at h_orig
      exact h_orig

    have hk_eq : k * (k - 2) = 0 := by
      -- [Algebraic simplification using eq_all 1 0 and eq_all 0 0 omitted]
```

```
    have hk2 : k = 0 ∨ k = 2 := by
      cases mul_eq_zero.mp hk_eq with
      | inl h1 => left; exact h1
      | inr h2 => right; omega

    rcases hk2 with hk0 | hk2
    · left
      have eq00 := eq_all 0 0
      rw [hk0] at eq00
      have hc0 : c = 0 := by linarith [eq00]
      ext x
      simp only [Pi.zero_apply]
      have hfx := h5 x
      rw [hk0, hc0] at hfx
      omega
    · right
      use c
      ext x
      have hfx := h5 x
      rw [hk2] at hfx
      omega

  -- Prove that the derived candidates are indeed solutions mappings
  · rintro (rfl | ⟨c, rfl⟩)
    · intro x y
      simp only [Pi.zero_apply, mul_zero, add_zero]
    · intro x y
      dsimp only
      ring
```

Example Proof Sketch (Lean-IMO-Bench, Basic 006)

```
import Mathlib

open Polynomial

-- Shared Definitions from the file environment
def K (c : ℕ → ℤ) : ℤ := (c 1)^2 - (2 : ℤ) * (c 0) * (c 2)

def k_target (c : ℕ → ℤ) : ℕ := max 2 (Int.toNat (K c + (1 : ℤ)))

def esymm_one_target (s : Multiset ℝ) : ℕ → ℝ
  | 0 => 0
  | k + 1 => s.esymm k

def esymm_two_target (s : Multiset ℝ) : ℕ → ℝ
  | 0 => 0
  | 1 => 0
  | k + 2 => s.esymm k

-- Supporting Lemmas (with sorry)
lemma root_count_bound_implies_eq (c : ℕ → ℤ) (hc : c 0 ≠ 0) (k : ℕ)
  (h_not_less : ¬ (((∑ i ∈ Finset.Icc 0 k, monomial i (c i)).rootSet ℝ).ncard < k))
    :
  ((∑ i ∈ Finset.Icc 0 k, monomial i (c i)).rootSet ℝ).ncard = k ∧
  (∑ i ∈ Finset.Icc 0 k, monomial i (c i)).natDegree = k := by
  sorry

lemma k_le_K_of_eq (c : ℕ → ℤ) (hc : c 0 ≠ 0) (k : ℕ) (hk : (2 : ℕ) ≤ k)
  (h_eq : ((∑ i ∈ Finset.Icc 0 k, monomial i (c i)).rootSet ℝ).ncard = k)
  (h_deg : (∑ i ∈ Finset.Icc 0 k, monomial i (c i)).natDegree = k) :
  (k : ℤ) ≤ K c := by
```

```
  sorry

lemma k_target_ge_two (c : ℕ → ℤ) : (2 : ℕ) ≤ k_target c := by
  sorry

lemma k_target_gt_K (c : ℕ → ℤ) : K c < (k_target c : ℤ) := by
  sorry

-- Main Theorem
theorem PBBasic006 (c : ℕ → ℤ) (hc : c 0 ≠ 0) :
  ∃ k, ((∑ i ∈ Finset.Icc 0 k, monomial i (c i)).rootSet ℝ).ncard < k := by
  use k_target c
  by_contra h
  have h1 := root_count_bound_implies_eq c hc (k_target c) h
  have h2 := k_le_K_of_eq c hc (k_target c) (k_target_ge_two c) h1.1 h1.2
  have h3 := k_target_gt_K c
  linarith
```

Informal context. An *informal proof* is a natural-language plan for proving the current goal directly, without introducing a decomposition into new subgoals. It guides the generation of a complete formal proof. An *informal blueprint* is a higher-level natural-language plan that explains how the current goal can be reduced to useful supporting lemmas. Unlike an informal proof, an informal blueprint may introduce proposed lemmas that are not immediately proved and may include auxiliary lemmas for later proof steps. These proposed lemmas are then translated into formal subgoals and organized in the proof DAG. Examples are shown below:

Example Informal Proof (Lean-IMO-Bench, Basic 001)

Let $S = \{f : \mathbb{Z} \to \mathbb{Z} \mid \forall x, y \in \mathbb{Z}, f(2x) + 2f(y) = f(f(x+y))\}$ and $T = \{0\} \cup \{x \mapsto 2x + c \mid c \in \mathbb{Z}\}$. We want to show that $S = T$.

Part 1: Proof that $T \subseteq S$

Suppose $f \in T$. Then either f is the constant zero function ($f = 0$) or $f(x) = 2x + c$ for some $c \in \mathbb{Z}$.

- Case 1: If $f = 0$, then for any $x, y \in \mathbb{Z}$, the left-hand side is $f(2x) + 2f(y) = 0 + 2(0) = 0$. The right-hand side is $f(f(x+y)) = f(0) = 0$. The two sides match, so $f \in S$.
- Case 2: If $f(x) = 2x + c$ for some $c \in \mathbb{Z}$, then for any $x, y \in \mathbb{Z}$, the left-hand side is: $f(2x) + 2f(y) = (2(2x) + c) + 2(2y + c) = 4x + 4y + 3c$. The right-hand side is: $f(f(x+y)) = f(2(x+y) + c) = 2(2x + 2y + c) + c = 4x + 4y + 3c$. The two sides are equal, so $f \in S$.

This proves $T \subseteq S$.

Part 2: Proof that $S \subseteq T$

Let $f \in S$. Then for all $x, y \in \mathbb{Z}$, we have:
(1) $f(2x) + 2f(y) = f(f(x+y))$.

Substitute $x = 0$ into Eq. (1):
$f(0) + 2f(y) = f(f(y))$.
Let $c = f(0)$. Then for all $y \in \mathbb{Z}$, we have:
(2) $f(f(y)) = 2f(y) + c$.

Using Eq. (2) on the right-hand side of Eq. (1) (since it holds for any input, including $x + y$), we obtain:
(3) $f(2x) + 2f(y) = 2f(x+y) + c$.

Substitute $y = 0$ into Eq. (3):
$f(2x) + 2f(0) = 2f(x) + c$.
Since $f(0) = c$, this becomes $f(2x) + 2c = 2f(x) + c$, which yields:
(4) $f(2x) = 2f(x) - c$.

Now, substitute Eq. (4) back into the left-hand side of Eq. (3):
$(2f(x) - c) + 2f(y) = 2f(x + y) + c$.
Rearranging the terms gives:
$2f(x + y) = 2f(x) + 2f(y) - 2c$.
Because the codomain is $\mathbb{Z}$, we can divide both sides by 2 to obtain:
(5) $f(x + y) = f(x) + f(y) - c$.

Define a new function $g : \mathbb{Z} \to \mathbb{Z}$ by $g(x) = f(x) - c$. Then we can rewrite Eq. (5) as:
$g(x + y) + c = (g(x) + c) + (g(y) + c) - c$.
Simplifying this yields Cauchy's functional equation:
$g(x + y) = g(x) + g(y)$ for all $x, y \in \mathbb{Z}$.

Since g satisfies $g(x + y) = g(x) + g(y)$ over integers, it must be a linear function. Let $k = g(1) \in \mathbb{Z}$. By standard induction, we have $g(x) = kx$ for all integers $x \geq 0$ (with $g(0) = 0$). For negative integers, $0 = g(0) = g(x + (-x)) = g(x) + g(-x)$, meaning $g(x) = -g(-x) = -(-kx) = kx$. Thus $g(x) = kx$ for all $x \in \mathbb{Z}$.
As a result, we have $f(x) = kx + c$ for all $x \in \mathbb{Z}$.

To determine the permissible values of k and c, substitute $f(x) = kx + c$ back into the original Eq. (1):
Left-hand side: $f(2x) + 2f(y) = k(2x) + c + 2(ky + c) = 2kx + 2ky + 3c$.
Right-hand side: $f(f(x + y)) = k(f(x + y)) + c = k(k(x + y) + c) + c = k^2x + k^2y + (k + 1)c$.

For the equality $2kx + 2ky + 3c = k^2x + k^2y + (k + 1)c$ to hold for all $x, y \in \mathbb{Z}$, the corresponding coefficients must match.
Setting $x = 0$ and $y = 0$ yields:
(6) $3c = (k + 1)c$.
Setting $x = 1$ and $y = 0$ yields:
$2k + 3c = k^2 + (k + 1)c$.
Subtracting Eq. (6) from this gives $2k = k^2$, which simplifies to $k(k - 2) = 0$. Since $k \in \mathbb{Z}$, the only solutions are $k = 0$ or $k = 2$.

- Case A: $k = 0$. Substitute $k = 0$ into Eq. (6): $3c = c \implies 2c = 0 \implies c = 0$. Thus, $f(x) = 0x + 0 = 0$, meaning f is the zero function. Hence $f \in \{0\} \subseteq T$.
- Case B: $k = 2$. Substitute $k = 2$ into Eq. (6): $3c = 3c$, which is true for any $c \in \mathbb{Z}$. Thus, $f(x) = 2x + c$ for some $c \in \mathbb{Z}$. Hence $f \in \{x \mapsto 2x + c \mid c \in \mathbb{Z}\} \subseteq T$.

In all cases, any function $f \in S$ is also in T, proving $S \subseteq T$.
Since both subset inclusions have been established, $S = T$.

Example Informal Blueprint (Lean-IMO-Bench, Basic 006)

High-Level Mathematical Idea
The goal is to prove that $k \leq K(c)$, where $K(c) = c_1^2 - 2c_0c_2$, given that the polynomial $P(x) = \sum_{i=0}^{k} c_i x^i$ with integer coefficients has degree k and exactly k real roots.
The proof elegantly uses multiset symmetric polynomials and the AM-GM inequality, avoiding rational functions or polynomial derivatives:

1. **Polynomial Roots and Splitting**: Since P has natural degree k and k distinct real roots, it splits completely over $\mathbb{R}$, and the multiset of its roots S has cardinality k.
2. **Vieta's Formulas**: Vieta's formulas express the coefficients c_0, c_1, c_2 in terms of the elementary symmetric polynomials of S:
   $c_0 = c_k(-1)^k E_k$
   $c_1 = c_k(-1)^{k-1} E_{k-1}$
   $c_2 = c_k(-1)^{k-2} E_{k-2}$
   where $E_i = \text{esymm}_i(S)$.
3. **Multiset Identity**: We construct a new multiset Y by mapping each root $x \in S$ to the product of all other roots, $(S \setminus \{x\})$.`prod`. Purely multiset inductive identities show:
   $\sum Y = E_{k-1}$
   $\text{esymm}_2(Y) = E_k E_{k-2}$
   $\prod Y = (E_k)^{k-1}$
4. **Sum of Squares**: For the multiset $Z = c_k Y$, we evaluate the sum of its squares $W = \{z^2 \mid z \in Z\}$. By the relation $(\sum Z)^2 = \sum(Z^2) + 2\text{esymm}_2(Z)$, we can algebraically compute $\sum W = c_1^2 - 2c_0c_2 = K(c)$.

5. **Integer Product Bound**: The product of Z evaluates to $c_k((-1)^k c_0)^{k-1}$. Since P has degree k and $c_0 \neq 0$, both c_0 and c_k are non-zero integers. Thus, the product of Z is a non-zero integer, implying $\prod W = (\prod Z)^2 \geq 1$.
6. **AM-GM Inequality**: Applying the AM-GM inequality to the multiset W (which consists of k non-negative real numbers whose product is ≥ 1), we obtain $\sum W \geq k \implies K(c) \geq k$.

Required Global Definitions, Variables, or Structures
No new definitions, axioms, or structures are needed. We use purely standard Mathlib components (like `Multiset`, `Polynomial`, and `esymm`).

Smaller Lemmas (Subproblems)

```
lemma coeff_of_sum_Icc (c : ℕ → ℤ) (k : ℕ) (hk : (2 : ℕ)
≤ k) (i : ℕ) (hi : i ≤ k) :
    (∑ j ∈ Finset.Icc 0 k, Polynomial.monomial j (c
j)).coeff i = c i
```

Purpose: Simplifies polynomial coefficient extraction from the given sum format.

```
lemma card_roots_eq_of_ncard_rootSet {k : ℕ} {P : Polynomial
ℝ}
    (h_deg : P.natDegree = k)
    (h_ncard : (P.rootSet ℝ).ncard = k) :
    P.roots.card = k
```

Purpose: Shows that a polynomial with k distinct real roots and degree k has exactly k roots counting multiplicities.

```
lemma multiset_map_erase_prod_sum {R : Type*} [CommRing R] (s
: Multiset R) :
    (s.map (fun x => (s.erase x).prod)).sum = s.esymm (s.card
- 1)
```

Purpose: Multiset identity relating the sum of element-wise excluded products to E_{k-1}.

```
lemma multiset_map_erase_prod_esymm_two {R : Type*} [CommRing
R] (s : Multiset R) :
    (s.map (fun x => (s.erase x).prod)).esymm 2 = s.prod *
s.esymm (s.card - 2)
```

Purpose: Multiset identity relating the 2nd elementary symmetric polynomial of excluded products to $E_k E_{k-2}$.

```
lemma multiset_sum_sq_eq {R : Type*} [CommRing R] (s :
Multiset R) :
    (s.map (fun x => x^2)).sum = (s.sum)^2 - (2 : R) *
s.esymm 2
```

Purpose: Expresses the sum of squares of a multiset in terms of its sum and its 2nd elementary symmetric polynomial.

```
lemma multiset_map_erase_prod_prod {R : Type*} [CommRing R] (s
: Multiset R) :
    (s.map (fun x => (s.erase x).prod)).prod = s.prod ^
(s.card - 1)
```

Purpose: Computes the full product of the excluded products multiset.

```
lemma multiset_sum_ge_card_of_prod_ge_one (W : Multiset ℝ)
(hw : ∀ x ∈ W, 0 ≤ x) (hp : (1 : ℝ) ≤ W.prod) :
    (W.card : ℝ) ≤ W.sum
```

Purpose: The AM-GM inequality specialized for a multiset whose product is at least 1, proving that the sum is bounded below by its cardinality.

Proof Body Outline

1. Define P as the sum $\sum_{i \in \text{Finset.Icc } 0\, k}$ monomial $i(c_i)$ and P_R as P.map (algebraMap $\mathbb{Z}\, \mathbb{R}$).
2. Apply `coeff_of_sum_Icc` to assert P_R.coeff $i = (c_i : \mathbb{R})$ for $i \in \{0, 1, 2, k\}$.
3. Establish P_R.roots.card $= k$ using `card_roots_eq_of_ncard_rootSet` and the natural degree injectivity.
4. Establish that P_R.splits (RingHom.id $\mathbb{R}$) follows from `Polynomial.splits_iff_card_roots`.
5. Let $s = P_R$.roots. Invoke Vieta's formulas (`Polynomial.coeff_eq_esymm_roots_of_splits`) to express c_0, c_1, and c_2 in terms of s.esymm i.

6. Define multisets Y and Z matching the theoretical blueprint. Use `multiset_sum_sq_eq`, `multiset_map_erase_prod_sum`, and `multiset_map_erase_prod_esymm_two` to show that the sum of the squared elements of Z expands algebraically to exactly $(c_1^2 - 2c_0c_2 : \mathbb{R}) = (K(c) : \mathbb{R})$.
7. Use `multiset_map_erase_prod_prod` to find Z.prod $= c_k((-1)^k c_0)^{k-1}$.
8. Observe that since c_0 and c_k are non-zero integers, their algebraic combination Z.prod represents a non-zero integer, so its square (the product of $W = Z^2$) is ≥ 1.
9. Feed W to `multiset_sum_ge_card_of_prod_ge_one` to deduce that W.sum $\geq W$.card.
10. Using W.card $= k$ and W.sum $= (K(c) : \mathbb{R})$, deduce $(k : \mathbb{R}) \leq (K(c) : \mathbb{R})$. Use `norm_cast` to translate this back to $(k : \mathbb{Z}) \leq K(c)$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and intro reflect this paper's contribution and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Appendix Part A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This paper focuses on empirically exploring the capabilities of current general-purpose large language models for formal theorem proving. We therefore do not include theoretical results that would require additional assumptions beyond the scope of this study.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We include our proposed dataset details in Section 3, and experiment details in Section 4 and Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the detailed contexts generated by LEAP in Appendix C and will release the code for the agentic framework upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We provide the details of main experiments in Section 4 and ablation studies in Section 5.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: In formal theorem proving, outcomes are binary: a statement is either proved or not proved. To contextualize the performance gains, we provide a compute analysis in Section 4 and report the number of rollouts used in our experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform to code of ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential impacts and future direction in Section 6. This work is foundational research in formal mathematics and automated theorem proving. Because it deals strictly with abstract mathematical verification, we do not foresee any direct negative societal impacts arising from the intended use or reasonably foreseeable misuse of this framework, so a dedicated broader impacts section was not included.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite original work for code or dataset we use in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We disclose the details of Lean-IMO-Bench in Section 3.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We discuss the use of LLMs for the proposed agentic framework LEAP in Section 3 and Section 4.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.