Numina-Lean-Agent: An Open and General Agentic Reasoning System for Formal Mathematics

Junqi Liu [*1] **Zihao Zhou** [*2 3] **Zekai Zhu** [*4] **Marco Dos Santos** [5] **Weikun He** [1] **Jiawei Liu** [6] **Ran Wang** [6] **Yunzhou Xie** [7] **Junqiao Zhao** [4] **Qiufeng Wang** [3] **Lihong Zhi** [1] **Jia Li** [6] **Wenda Li** [8]

[*]Equal contribution
[1]Academy of Mathematics and Systems Science, University of Chinese Academy of Sciences
[2]University of Liverpool [3]Xi'an Jiaotong-Liverpool University [4]Tongji University
[5]University of Cambridge [6]Numina [7]Imperial College London [8]University of Edinburgh.

Abstract

Agentic systems have recently become the dominant paradigm for formal theorem proving, achieving strong performance by coordinating multiple models and tools. However, existing approaches often rely on task-specific pipelines and trained formal provers, limiting their flexibility and reproducibility. In this paper, we propose the paradigm that directly uses a general coding agent as a formal math reasoner. This paradigm is motivated by (1) A general coding agent provides a natural interface for diverse reasoning tasks beyond proving, (2) Performance can be improved by simply replacing the underlying base model, without training., and (3) MCP enables flexible extension and autonomous calling of specialized tools, avoiding complex design. Based on this paradigm, we introduce **Numina-Lean-Agent**, which combines Claude Code with Numina-Lean-MCP to enable autonomous interaction with Lean, retrieval of relevant theorems, informal proving and auxiliary reasoning tools. Using Claude-Opus-4.5 as the base model, Numina-Lean-Agent solves all problems in Putnam 2025 (12/12), matching the best closed-source system. Beyond benchmark evaluation, we further demonstrate its generality by interacting with mathematicians to successfully formalize the Brascamp–Lieb theorem. We release Numina-Lean-Agent and all solutions at https://github.com/project-numina/numina-lean-agent.

1. Introduction

Formal theorem proving aims to construct machine-verifiable proofs for mathematical theorems within rigorously defined logical systems, such as Lean (2015) and Isabelle (Paulson, 1994). Unlike informal mathematical reasoning, formal verification systems provide tools for automatically and soundly verifying the correctness of proofs. Consequently, these systems establish a foundation for developing reliable reasoning. Previous advances in neural theorem proving have focused on developing single-model formal provers. Early provers relied on tactic prediction combined with explicit search methods, such as Monte Carlo tree search, to explore the proof space. (Lample et al., 2022; Hubert et al., 2025). To mitigate the efficiency limitations of search-based methods, subsequent work explored whole proof generation to directly produce complete proofs (Xin et al., 2024). Subsequently, other efforts incorporated informal reasoning to guide tactic generation and proof construction (Wang et al., 2025; Ren et al., 2025). Despite notable progress, effectively capturing long-horizon, structured reasoning within formal systems remains a central challenge.

More recently, several systems have moved beyond single-model formal provers by introducing agentic workflows that enable provers to interact with formal theorem proving environments and other models. For example, HILBERT (Varambally et al., 2025) proposes an agentic framework that combines informal reasoners with formal provers to guide proof construction. In parallel, Seed-Prover 1.5 (Chen et al., 2025) trains a formal prover via large-scale agentic reinforcement learning, emphasizing repeated interaction with the Lean compiler and related tools. In addition, AxiomProver (Axiom Math Team, 2025), developed by Axiom Math, adopts an autonomous multi-agent ensemble architecture and has achieved a perfect

Correspondence to: Jia Li <jia@projectnumina.ai>, Wenda Li <wenda.li@ed.ac.uk>.

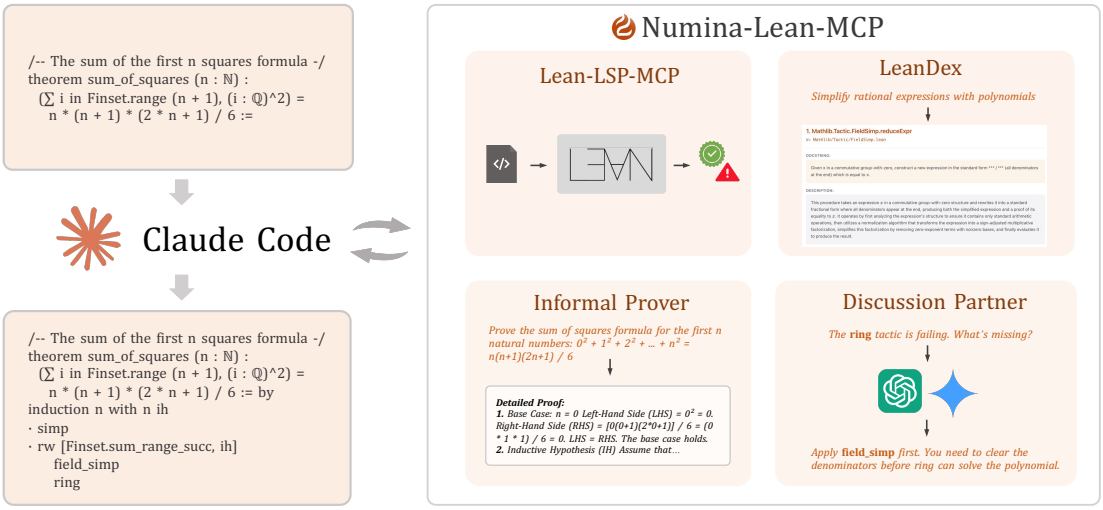

Figure 1. Overview of Numina-Lean-Agent, an agentic formal reasoning framework built on Claude Code and Numina-Lean-MCP. The agent autonomously selects and invokes specialized reasoning tools to handle diverse queries.

score on Putnam-2025. These systems highlight the growing effectiveness of agentic proving systems. Despite their strong performance, existing agentic proving systems exhibit several limitations: (1) They rely on task-specific reasoning pipelines that are explicitly designed and often coupled with extensively trained formal provers, which can limit their extensibility to new tools or domains. (2) Most systems are closed-source and provide limited implementation details, making it difficult for the broader community to reproduce and extend their work.

In this paper, we propose a paradigm for building a formal math reasoner based on a general coding agent, which is motivated by three reasons: (1) Coding agents provide a native interface for diverse proof engineering tasks beyond proving. (2) It allows for the flexible replacement of underlying base models to enhance reasoning capabilities without any training. (3). Integration of MCP enables the plug-and-play extension of specialized reasoning tools and the model can autonomously invoke them based on the specific query. Following this paradigm, we propose Numina-Lean-Agent, which consists of Claude Code and Numina-Lean-MCP for providing diverse reasoning tools. Specifically, Numina-Lean-MCP integrates several core components. It includes *Lean-LSP-MCP* (Dressler, 2025) [2] for agentic interaction with the Lean theorem prover. To provide relevant background knowledge, we develop *LeanDex* for semantic retrieval of related theorems and definitions from Lean libraries such as mathlib. An *Informal Prover* (Huang & Yang, 2025) is used to generate detailed informal proof solutions, while a *Discussion Partner* tool enables querying external language models to assist in reasoning and planning. Together, these components make Numina-Lean-Agent a general and powerful formal mathematical reasoning system.

Using Claude-Opus-4.5 (Anthropic, 2025) as the base model, Numina-Lean-Agent successfully solved all 12 problems in the Putnam 2025, achieving state-of-the-art performance. This result matches the closed-source system AxiomProver and surpasses Harmonic's Aristotle (Achim et al., 2025) by two problems. We report all solutions along with their computational costs and proof lengths in Section 3. Notably, on certain problems such as Problem-B1, Numina-Lean-Agent produced remarkably concise proofs than AxiomProver and Seed-Prover 1.5. Beyond standard automated proving, Numina-Lean-Agent serves as a general mathematical reasoning system, enabling mathematicians to engage in interactive "vibe proving". We demonstrate this paradigm by collaborating with human experts to formalize the Brascamp-Lieb theorem, with the details of the interactive process reported in Section 4.

2. Numina-Lean-Agent

2.1. Overview

As shown in Figure 1, Numina-Lean-Agent is an agentic formal theorem proving framework built upon Claude Code and Numina-Lean-MCP. Functioning as an autonomous agent, it can dynamically select and invoke the appropriate reasoning

[2] https://github.com/oOo0oOo/lean-lsp-mcp

tools within Numina-Lean-MCP to handle diverse queries and complete complex formal reasoning tasks.

2.2. Numina-Lean-MCP

Lean-LSP-MCP. Lean-LSP-MCP (Dressler, 2025) is a Model Context Protocol (MCP) server explicitly designed for the Lean theorem prover. Acting as a bridge between LLMs and the Lean kernel via the Language Server Protocol (LSP), it empowers models with the capability to deeply comprehend, analyze, and manipulate Lean projects. It significantly enhances the proving capabilities of models through a toolset organized into three distinct dimensions:

1. Semantic Awareness and Interaction: This dimension provides a suite of tools enabling agents to simulate the behavior of proficient Lean users. Ranging from `lean_file_outline` for grasping global structures, to `lean_goal` for precise goal querying, and `lean_diagnostic_messages` for obtaining authoritative verification results, these tools liberate models from guessing regarding proof states, enabling precise decision-making grounded in the authentic compilation environment.
2. Code Execution and Strategy Exploration: It supports the instant compilation of isolated code snippets via `lean_run_code` and utilizes `lean_multi_attempt` to allow parallel execution and evaluation of multiple strategies at a single proof node. This mechanism establishes a robust "trial-feedback-optimization" closed-loop for automated theorem proving.
3. Theorem Retrieval and Knowledge Enhancement: Multi-level search tools are integrated to effectively mitigate hallucinations. `lean_local_search` focuses on mining definitions within local lean projects and the standard library (stdlib), while `lean_loogle` facilitates searching the massive Mathlib repository via natural language or structured queries. This dual-retrieval mechanism ensures that every theorem cited by the model is factually existent and contextually appropriate.

LeanDex. We present a new theorem search tool for Lean that supports theorem retrieval under Lean v4.26.0. In contrast to existing tools, loogle imposes strict requirements on the format of search queries, while local search is mainly limited to searching within a local project. It is an agentic semantic search tool for Lean, capable of retrieving mathematical theorems and definitions across multiple packages, including mathlib and FLT. Given a natural language query, Leandex employs an intelligent agent to interpret and reason about the query, identifying the most relevant Lean objects. Built on top of LeanExplore, it extends the underlying semantic search framework with enhanced reasoning and retrieval capabilities, significantly improving both flexibility and coverage.

Informal Prover. We implement a lightweight Gemini IMO agent system (Huang & Yang, 2025) as the Informal Prover to generate detailed informal solutions. The system consists of two models: a Generator and a Verifier. The Generator is responsible for generating informal solutions, while the Verifier assesses the correctness of the generated solutions. These two models interact in an iterative refinement loop. When the Verifier identifies errors in the generated proof, it will provide feedback to the Generator. In the next iteration, the Generator refines its solution base on both the previous solution and the Verifier's feedback. This process continues until the Verifier accepts the solution as correct or a maximum number of iterations is reached, which we set to 20.

To improve the reliability of verification, the Verifier evaluates each candidate solution independently three times. A solution is accepted only if all three verification passes judge it to be correct. In our implementation, we use *Gemini-3-Pro-Preview* for both the Generator and the Verifier.

Discussion Partner. In scientific research, discussion is widely recognized as a highly effective cognitive tool. By exchanging diverse viewpoints and reasoning paths, researchers often overcome mental blind spots and spark new inspiration, thereby facilitating problem-solving. Inspired by this insight, we designed and implemented `discuss_partner`, a tool designed to assist the automated formalization process.

Specifically, this tool empowers Claude Code with the ability to 'seek assistance' during Lean formalization: when encountering obstacles—such as proof bottlenecks, dilemmas in strategy selection, or ambiguities in intermediate lemmas—the primary model can proactively initiate discussions with other LLMs. These collaborating models analyze the current state from distinct reasoning perspectives, offering candidate ideas or alternative proof paths to provide insightful feedback. Leveraging this multi-model collaborative mechanism, the system can explore the proof space more effectively, significantly enhancing both the robustness and the success rate of the formalization process.

3. Evaluation

3.1. Performance

We evaluated Numina-Lean-Agent on the Putnam 2025 benchmark and compared its performance with other existing provers. Notably, we used the formal statements provided by Seed-Prover 1.5. Moreover, all operations executed by our agent were strictly sequential, without any form of parallelization. Internet search access was disabled for all API calls for preventing the agent from retrieving solutions via online search. Under these settings, Numina-Lean-Agent achieved state-of-the-art performance, successfully solving 12 out of 12 problems on Putnam 2025. By default, each problem is allocated an approximate budget of $50. Due to their substantially higher difficulty and longer proof search trajectories, problem A5 is assigned a larger budget of approximately $1000, and problem B6 is allocated an increased budget of approximately $300. These values are intended to reflect relative computational effort rather than exact accounting.

Table 1. Performance comparison of Numina-Lean-Agent against other methods

PUTNAM2025	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
ARISTOTLE	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓
SEED-PROVER 1.5	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
AXIOM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUMINA-LEAN-AGENT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

We conducted a comparative experiment on two design paradigms for the `informal_prover` tool using the Putnam 2025 B4 problem. The first approach employs an iterative refinement strategy: generating an initial solution, verifying it, revising based on feedback, and repeating this cycle for n iterations (Huang & Yang, 2025). The second approach adopts an independent sampling strategy: generating n solutions independently and verifying each separately. The total number of calls remains identical across both approaches to ensure a fair comparison. Experimental results demonstrate that, under the same search configuration, the iterative refinement strategy significantly outperforms independent sampling. Specifically, independent sampling failed to complete the formal proof of B4 within 10 rounds, whereas iterative refinement successfully completed the proof in only 5 rounds, clearly demonstrating the advantage of feedback-driven iterative correction in improving proof efficiency. We also propose a new subagent-based approach to solve problem A5; the detailed methodology will be discussed in Section 3.2.

Table 2. Evaluation results on Putnam 2025 by Numina-Lean-Agent.

PUTNAM2025	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
W/O INFORMAL	✓	✓		✓					✓			
W INFORMAL	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
W SUBAGENT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

As shown in Table 3, we compare the per-problem solving time of Numina-Lean-Agent against other representative provers on Putnam 2025. Despite the fact that Numina-Lean-Agent operates without any parallel execution, it demonstrates notable efficiency advantages on a subset of problems, achieving shorter solving times than competing methods on several instances.

Table 3. Time spent comparison of Numina-Lean-Agent against other methods (Unit: minutes)

PUTNAM2025	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
ARISTOTLE	**30**	60	**30**	180	–	**60**	150	**25**	40	–	420	**180**
SEED-PROVER 1.5	60	**30**	120	240	–	240	540	360	**30**	120	240	**180**
AXIOM	110	180	165	**107**	**518**	259	270	65	43	**112**	254	494
NUMINA-LEAN-AGENT	97	**30**	44	169	2040	89	**55**	142	**30**	308	**88**	797

In Table 4, we further compare the proof length generated by different provers. For a fair comparison, we remove all comments and blank lines from the final Lean code and measure the resulting number of lines. The results show that,

compared with AxiomProver and Seed-Prover 1.5, Numina-Lean-Agent produces shorter proofs on a substantial number of problems; in particular, on A3, B1, and B5, the advantage is especially pronounced. We note that step-based provers have an inherent advantage in producing very short proofs, and therefore the proofs generated by Numina-Lean-Agent are generally longer than those produced by Aristotle. Nevertheless, when compared with other agentic provers under a similar setting, Numina-Lean-Agent consistently yields more concise formalizations on most problems, demonstrating its effectiveness in generating compact and efficient formal proofs.

Table 4. Code length comparison of Numina-Lean-Agent against other methods

PUTNAM2025	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
ARISTOTLE	45	195	103	291	–	123	223	108	70	–	291	280
SEED-PROVER 1.5	631	469	927	1095	–	881	849	1613	584	**628**	2499	2594
AXIOM	556	458	1089	825	**1878**	**468**	1179	**346**	302	993	1310	**862**
NUMINA-LEAN-AGENT	**365**	**401**	**422**	**605**	3263	835	**328**	690	**292**	648	**929**	1820

3.2. Putnam-2025-A5

For problem A5, we adopt a novel subagent mechanism that decomposes the proof into several subgoals and solves them independently, effectively mitigating the issue of excessively long contexts. Our empirical observations indicate that when the context becomes too long, the model's ability to follow instructions degrades significantly, making it difficult to focus on a single proof objective, which in turn hampers the resolution of critical subgoals. By introducing subagents and modularizing the proof task, we can substantially alleviate these issues and improve the overall proof efficiency.

The core of A5 is to prove that, among all permutations satisfying a certain property, alternating permutations occur in the largest number. In several previous experiments, the model repeatedly got stuck on this key lemma. We conjecture that this difficulty is caused by overly long contexts, and therefore adopt a subagent strategy that isolates this lemma from the overall proof and handles it separately. Concretely, the subagent first invokes GPT-5.2 to generate an informal hint, and then carries out the corresponding formalization guided by this hint. This process can be iterated until the lemma is successfully formalized.

4. Formalizing Brascamp Lieb with Numina-Lean-Agent

4.1. Blueprint Generation

Formalizing a complex theorem in Lean is a long-horizon task with dense dependencies. When Claude Code is asked to directly prove the final statement, it often commits to suboptimal formulations and gets trapped in local dead ends. We therefore introduce a **blueprint** as an explicit planning layer that decomposes the global goal into a sequence of verifiable subgoals.

A blueprint is a design-document-style artifact consisting of (i) required definitions and notation, (ii) a curated list of intermediate lemmas with suitable granularity, and (iii) the final theorem whose proof largely composes these lemmas. Dependencies are recorded explicitly (e.g., via `\uses{...}`), forming a DAG that determines proving order and reduces ambiguity during search.

Importantly, blueprint generation is *recursive* and tightly coupled to the formalization loop rather than a one-shot preprocessing step. As the agent attempts to discharge lemmas in Lean, compilation feedback and proof-state inspection may reveal that an informal step is incorrect, underspecified, or split at an unsuitable granularity. In such cases, the agent revisits and refines the blueprint (e.g., strengthening assumptions, rephrasing statements to match Lean interfaces, or inserting missing intermediate lemmas) and then continues formalization with the updated plan. To improve robustness, the agent can also invoke external discussion models (e.g., Gemini) to propose alternative decompositions or repairs when the current blueprint repeatedly leads to bottlenecks.

Overall, the blueprint plays the role of a high-level mathematical plan: a stronger mathematical reasoner is used to decompose a difficult statement into a sequence of small, checkable steps, while Claude Code focuses on turning these steps into machine-verifiable Lean proofs. Crucially, verification is not merely an endpoint—Lean feedback (failed typeclass search,

missing lemmas, mismatched interfaces, etc.) provides concrete signals that are fed back to revise the blueprint, yielding a closed-loop "plan–formalize–refine" workflow that stabilizes long-horizon formalization.

4.2. Human-AI Cooperation

We design a human–machine collaborative interaction framework for Numina-Lean-Agent, enabling human experts to work together with the agent by writing hints and modifying the Blueprint. One of the authors of this paper is a mathematician, and we conduct a collaborative case study based on his preprint **Effective Brascamp–Lieb Inequalities** (2025), published in November 2025. In this experiment, a mathematician, a Lean formalization expert, and Numina-Lean-Agent jointly cooperate to formalize the results of the paper.

Over a period of less than two weeks of intermittent collaboration, the two human experts and the agent completed the formalization of more than **8,000** lines of Lean code. During this process, the agent autonomously introduced approximately **70** new definitions, lemmas, and theorems, illustrating its ability to actively extend the formal library and participate in large-scale, sustained formalization efforts.

When formalising more involved arguments, the agent sometimes chose to further decompose the proof, introducing additional intermediate lemmas that were more fine-grained than those in the original blueprint. This behaviour appears to be a form of adaptive proof decomposition tailored to the demands of formal verification.

Moreover, compared with other specialized prover models, our agent is not restricted to theorem proving alone, but instead exhibits strong general-purpose reasoning capabilities. For a given formal statement whose correctness is not known in advance, traditional approaches typically can only attempt to prove both the original statement and its negation in parallel. In contrast, during our formalization of the Brascamp–Lieb inequalities, we observed that our agent is able to actively reason about the validity of the statement itself during the proof process. When it detects that a statement is incorrect, it can autonomously revise the statement accordingly. This ability to dynamically inspect and modify the problem formulation during formalization has not been present in previous provers. We present concrete examples of this behavior in the Appendix A.1.

4.3. Limitation

Sometimes, our system generates Lean code that is overly long or less well-structured. In practice, we mainly tasked the agent with two kinds of 'sorry's of different difficulty. When a 'sorry' involved only local reasoning within an already well-structured proof, the agent usually filled the gaps with high-quality code. However, when a 'sorry' corresponded to the complete proof of a lemma, the agent was generally able to achieve the goal, but the resulting code tended to be verbose and less concise than desired. This highlights a limitation of the system: while it can handle complex proof goals, the readability and structure of generated formalizations may degrade for larger or more intricate tasks.

Our system occasionally struggles with type-level issues, which can significantly slow down the proof process. For instance, in one case, the agent failed completely—not because of difficulties in the core mathematical argument, but due to a type conversion from Real to NNReal. Such type-level constraints are rarely made explicit in informal mathematics, so the agent had difficulty reconstructing the required structure on its own. After revising the proof workflow and handling type conversions in advance to make the formalization path more "type-friendly", the agent was able to complete the remaining proof successfully. This case highlights an inherent gap between informal and formal proofs and underscores the challenge that type-level requirements can pose for automated reasoning systems.

Moreover, despite their strong problem-solving capabilities in automated theorem proving, current agents still exhibit a clear gap between functional correctness and formal elegance. While agent-generated proofs often pass Lean's compiler checks, they are frequently perceived by experienced Mathlib contributors as overly result-oriented, relying on verbose and low-level tactic scripts. Compared to human-written Mathlib code, these proofs lack structured abstraction and idiomatic use of higher-level patterns, leaving substantial room for improvement in conciseness, readability, and conformity to Mathlib's community standards.

References

Achim, T., Best, A., Bietti, A., Der, K., Fédérico, M., Gukov, S., Halpern-Leistner, D., Henningsgard, K., Kudryashov, Y., Meiburg, A., Michelsen, M., Patterson, R., Rodriguez, E., Scharff, L., Shanker, V., Sicca, V., Sowrirajan, H., Swope, A., Tamas, M., Tenev, V., Thomm, J., Williams, H., and Wu, L. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Anthropic. Claude opus 4.5 system card. Technical report, November 2025. URL https://www.anthropic.com/claude-opus-4-5-system-card.

Axiom Math Team. From seeing why to checking everything. https://axiommath.ai/territory/from-seeing-why-to-checking-everything, 2025. Axiom Math blog post.

Bénard, T. and He, W. Effective brascamp-lieb inequalities. *arXiv preprint arXiv:2511.11091*, 2025.

Chen, J., Chen, W., Du, J., Hu, J., Jiang, Z., Jie, A., Jin, X., Jin, X., Li, C., Shi, W., et al. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *arXiv preprint arXiv:2512.17260*, 2025.

de Moura, L., Kong, S., Avigad, J., Van Doorn, F., and von Raumer, J. The Lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pp. 378–388. Springer, 2015.

Dressler, O. Lean LSP MCP: Tools for agentic interaction with the Lean theorem prover, 3 2025. URL https://github.com/oOo0oOo/lean-lsp-mcp.

Huang, Y. and Yang, L. F. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.

Hubert, T., Mehta, R., Sartran, L., Horváth, M. Z., Žužić, G., Wieser, E., Huang, A., Schrittwieser, J., Schroecker, Y., Masoom, H., Bertolli, O., Zahavy, T., Mandhane, A., Yung, J., Beloshapka, I., Ibarz, B., Veeriah, V., Yu, L., Nash, O., Lezeau, P., Mercuri, S., Sönne, C., Mehta, B., Davies, A., Zheng, D., Pedregosa, F., Li, Y., von Glehn, I., Rowland, M., Albanie, S., Velingker, A., Schmitt, S., Lockhart, E., Michalewski, H., Sonnerat, N., Hassabis, D., Kohli, P., and Silver, D. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025. URL https://www.nature.com/articles/s41586-025-09833-y.

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X. Hypertree proof search for neural theorem proving. *Advances in Neural Information Processing Systems*, 35:26337–26349, 2022.

Paulson, L. C. *Isabelle: A generic theorem prover*. Springer, 1994.

Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Varambally, S., Voice, T., Sun, Y., Chen, Z., Yu, R., and Ye, K. Hilbert: Recursively building formal proofs with informal reasoning. *arXiv preprint arXiv:2509.22819*, 2025.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., Lu, J., de Saxcé, H., Bailey, B., Song, C., Xiao, C., Zhang, D., Zhang, E., Pu, F., Zhu, H., Liu, J., Bayer, J., Michel, J., Yu, L., Dreyfus-Schmidt, L., Tunstall, L., Pagani, L., Machado, M., Bourigault, P., Wang, R., Polu, S., Barroyer, T., Li, W.-D., Niu, Y., Fleureau, Y., Hu, Y., Yu, Z., Wang, Z., Yang, Z., Liu, Z., and Li, J. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11354.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2024. URL https://arxiv.org/abs/2408.08152.

A. Appendix

A.1. Self-Correction of Formal Statements during Formalization.

```
/-- When n is empty (dimension 0), the upper bound holds trivially.
    This lemma handles the degenerate case where the base space has dimension 0.
    In this case, both the LHS and RHS simplify to specific values and the inequality
    holds.
-/
lemma upperBound_empty_case {J : Type*} [Fintype J]
    {n : Type*} [Fintype n] [DecidableEq n]
    {m : J → Type*} [(j : J) → Fintype (m j)]
    (α : J → NNReal) (β : NNReal) (hα : ∀ i, 0 < α i)
    (D : locRegDatum (EuclideanSpace ℝ n) (fun j ↦ EuclideanSpace ℝ (m j)))
    (hP : D.IsMetricPercep α β)
    (hS : ∀ j : J, (D.map j).EssentialRank (α j) = Fintype.card (m j))
    (hn : ¬Nonempty n)
    (hβ_empty : β = 0)  -- Added: when n is empty, β must be 0 for the inequality to hold
    (A : (j : J) → EuclideanSpace ℝ (m j) →ₗ[ℝ] EuclideanSpace ℝ (m j))
    (hA : ∀ j, (A j).IsPosDef ∧ A j ≤ D.reg j) :
    (loc_constant_g_of n m D.1 A (fun j => (hA j).1) : ENNReal) ≤
      ↑((NNReal.rpow (Fintype.card n) (D.Acuity / 2)) *
      ∏ j, NNReal.rpow (D.weight j) (- D.weight j * Fintype.card (m j) / 2) *
      ∏ j, NNReal.rpow (α j) (- D.weight j * Fintype.card (m j)) *
      NNReal.rpow ‖(D.loc + ∑ j, (D.weight j) · (D.map j).adjoint ∘ₗ (D.reg j) ∘ₗ
        (D.map j)).toContinuousLinearMap‖₊
        ((D.Acuity - Fintype.card n + β) / 2) *
      NNReal.rpow
        ‖(D.loc.equivOfIsUnitDet (by simp
    [D.pos_loc.2])).symm.toLinearMap.toContinuousLinearMap‖₊
        (β / 2)) := by
```