FormalML: A Benchmark for Evaluating Formal Subgoal Completion in Machine Learning Theory

Xiao-Wen Yang[1,2*], Zihao Zhang[1*], Jianuo Cao[1,3], Zhi Zhou[1], Zenan Li[4], Lan-Zhe Guo[1,3], Yuan Yao[1], Taolue Chen[5], Yu-Feng Li[1,2†], and Xiaoxing Ma[1†]
[1]State Key Laboratory of Novel Software Technology, Nanjing University
[2]School of Artificial Intelligence, Nanjing University
[3]School of Intelligence Science and Technology, Nanjing University
[4]Department of Computer Science, ETH Zurich, Switzerland
[5]School of Computing and Mathematical Sciences, Birkbeck, University of London, UK
[*] Equal Contribution [†] Corresponding Author
yangxw@lamda.nju.edu.cn, zihaozhang@smail.nju.edu.cn

Abstract

Large language models (LLMs) have recently demonstrated remarkable progress in formal theorem proving. Yet their ability to serve as practical assistants for mathematicians—filling in missing steps within complex proofs—remains underexplored. We identify this challenge as the task of *subgoal completion*, where an LLM must discharge short but nontrivial proof obligations left unresolved in a human-provided sketch. To study this problem, we introduce **FormalML**, a Lean 4 benchmark built from foundational theories of machine learning. Using a translation tactic that converts procedural proofs into declarative form, we extract 4,937 problems spanning optimization and probability inequalities, with varying levels of difficulty. FormalML is the first subgoal completion benchmark to combine premise retrieval and complex research-level contexts. Evaluation of state-of-the-art provers highlights persistent limitations in accuracy and efficiency, underscoring the need for more capable LLM-based theorem provers for effective subgoal completion.

1 Introduction

Recent advancements in large language models (LLMs) have showcased their impressive performance in formal theorem proving [Ren et al., 2025, Wang et al., 2025, Xin et al., 2025], particularly culminating in silver-medal level performance at the 2025 International Mathematical Olympiad [Chen et al., 2025]. Nevertheless, their effectiveness as copilots for mathematicians in tackling research-level problems [Yang et al., 2024] remains far from satisfactory. For example, the recent Equational Theories Project [Tao et al., 2024] reported that "LLMs did not provide useful suggestions beyond what the human participants could already propose" in most of the difficult cases.

In this paper, rather than attempting to generate a complete proof for such research-level problems, we investigate an intermediate milestone, which we refer to as *subgoal completion*. To elaborate, human experts specify the problem in natural language, formalize the theorem, and outline the high-level proof structure, while AI systems assist by filling in the missing technical details [Jiang et al., 2022]. Figure 1 illustrates this process using the proof of gradient descent convergence [Li et al., 2024a, 2025b]; The expert provides the informal statement and the main reasoning steps. At the natural language level, the proof appears complete; however, once formalized, it remains incomplete, as indicated by the `sorry` placeholders. This gap arises because the theorem prover cannot automatically bridge each reasoning step. Subgoal completion precisely aims to bridge this gap, aligning the informal and formal proofs.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

Theorem. If f is differentiable, and its gradient f' is l-Lipschitz, then the gradient descent with stepsize $a \in [0, \frac{l}{2}]$ at any point x satisfies

$$f\big(x - af'(x)\big) \leq f(x) - \frac{a}{2} \| f'(x) \|^2$$

Proof. Let's plug in the gradient descent update by letting $x' = x - a\nabla f(x)$. We then get:

$$f(x') \leq f(x) + \langle \nabla f(x), x' - x \rangle + \frac{1}{2} l \|x' - x\|_2^2$$

$$= f(x) + \langle \nabla f(x), x - a\nabla f(x) - x \rangle + \frac{1}{2} l \|x - t\nabla f(x) - x\|_2^2$$

$$\leq f(x) - a\|\nabla f(x)\|_2^2 + \frac{1}{2} l a^2 \|\nabla f(x)\|_2^2$$

$$= f(x) - \frac{a}{2}\|\nabla f(x)\|_2^2$$

```
/-- the bound for one step of the gradient method
   using the Lipschitz continuity of the gradient -/

theorem convex_lipschitz
(h₁ : ∀ x₁ : E, HasGradientAt f (f' x₁) x₁)
(h₂ : l > (0 : ℝ)) (ha₁ : l ≤ 1 / a) (ha₂ : a > 0)
(h₃ : LipschitzWith l f') :
∀ x : E, f (x - a • (f' x)) ≤ f x - a / 2 * ‖f' x‖ ^ 2 := by
intro x
calc
  _ ≤ f x + inner (f' x) (x - a • (f' x) - x)
          + l / 2 * ‖x - a • (f' x) - x‖ ^ 2 := by sorry
  _ = f x + ((l.1 / 2 * a * a -a) * ‖f' x‖ ^ 2) := by sorry
  _ ≤ f x + (- a / 2* ‖(f' x)‖ ^2) := by sorry
  _ = f x - a / 2 * ‖f' x‖ ^ 2 := by sorry
```

Figure 1: The informal and formal versions of the statement and proof regarding the sufficient decrease in gradient descent. Both the formal statement and proof can be accurately formalized from the natural language version. However, to verify each step in Lean 4, there are unproven holes (notated as `sorry`) that need to be further completed by existing automated theorem provers.

Despite its natural role as an interface between informal and formal reasoning, subgoal completion still lacks dedicated benchmarks. Existing evaluations are built around competition-style problems and emphasize full-proof generation, whereas our target lies in research-level theorems and the task of completing intermediate proofsteps. Moreover, while proof-generation benchmarks primarily measure the LLM performance by proving success rate, subgoal completion poses a different challenge: handling complex proof contexts efficiently without resorting to verbose or speculative reasoning chains. In practice, a useful prover must therefore strike a balance between accuracy and conciseness. To achieve these goals, we make the following contributions:

- **A Lean 4 translation tactic for subgoal extraction.** We propose a symbolic strategy for the dataset construction. Specifically, we implemented a new Lean 4 tactic that extracts subgoal problems from procedural-style proofs. By adapting the length of the proof segments during extraction, we further utilize this tactic to generate new problems with varying levels of proving difficulty.
- **The FormalML benchmark.** We narrow the scope to the foundational theory of machine learning (ML) and establish a new benchmark in this domain. Our motivation is twofold. First, AI agents are becoming increasingly central to automating scientific discovery [Romera-Paredes et al., 2024], particularly within machine learning research [Lu et al., 2024, Yamada et al., 2025, Gottweis et al., 2025]. Hence, automated theorem proving plays a key role in guaranteeing the soundness of derived results. Second, automation tools that verify theoretical correctness can substantially ease the burden on human reviewers [Xu et al., Pineau et al., 2021], especially given the rapidly growing volume of ML conference submissions. A well-designed benchmark would accelerate the development of such tools. To this end, building on two Lean 4 libraries—Optlib [Li et al., 2024a] and FoML [Sonoda et al., 2025], we establish a dataset of 4,937 subgoal completion problems. A comprehensive comparative analysis of benchmarks is presented in Table 1.
- **Systematic evaluation of LLM-based provers on FormalML**. We evaluate state-of-the-art LLM-based provers on FormalML, highlighting fundamental limitations in accuracy, token-efficiency, and premise retrieval. Our results show that while models such as DeepSeek-Prover-V2 [Ren et al., 2025] improve retrieval capabilities, overall performance drops sharply on higher-difficulty problems. Moreover, chain-of-thought prompting, though effective in natural language reasoning, fails to improve proof completion and often reduces efficiency in this context. Additionally, we find expert iteration to be a effective training approach on FormalML. These insights highlight the need for further development and refinement in LLM-based theorem provers to better support mathematicians in their work.

Table 1: Comparison of existing Lean 4 benchmarks.

Benchmark	# Problems	Type	Premise	Complex Context	Subgoal Completion
MiniF2F [Zheng et al., 2022]	244	Olympiad	✗	✗	✗
PutnamBench [Tsoukalas et al., 2024]	522	Olympiad	✗	✗	✗
ProofNet [Azerbayev et al., 2023]	186	Undergraduate (UG)	✗	✓	✗
FormalMATH [Yu et al., 2025]	5,560	Olympiad & UG	✗	✓	✗
LeanDojo [Yang et al., 2023]	2000	Mathlib	✓	✗	✗
MiniCTX [Hu et al., 2024]	762	UG	✓	✓	✗
FormalML (Ours)	4,937	UG	✓	✓	✓

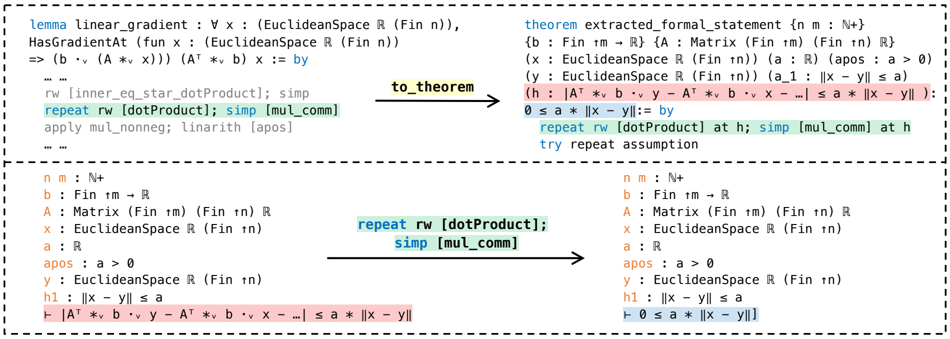

Figure 2: An example of the `to_theorem` tactic illustrates its functionality. When applied to the tactic `repeat rw [dotProduct]; simp [mul_comm]`, it captures pre- and post-execution proof states, abstracts their transition, and synthesizes a subgoal.

2 FormalML

2.1 Subgoal Extraction

With the growing availability of formalized libraries in machine learning theory, we aim to leverage these resources for dataset construction. A key challenge, however, lies in extracting subgoals from procedural-style proof scripts [Winograd, 1975, Wiedijk, 2012]. Unlike declarative-style proofs [Syme, 1997], which progress by iteratively introducing and proving intermediate subgoals, procedural-style proofs instead transform the proof state step by step. As a result, existing goal-extraction tactics such as Lean 4's `extract_goal` can only capture the overall goal of a proof state, but not the finer-grained intermediate subgoals corresponding to each reasoning step. This limitation is particularly pressing as most proofs in current libraries are written in the procedural style. For example, in the proof of the lemma `linear_gradient` (see Figure 2 and Appendix D), procedural scripts such as `rw` and `simp` span nine lines, yet declarative statements (e.g., `have`) are entirely absent.

Fortunately, in human-written Lean 4 proofs, we observe a consistent pattern: *although proofs are expressed procedurally as sequences of tactics, each proof line typically corresponds to a single reasoning step identified by the human author.* In practice, human experts tend to first complete the reasoning informally and subsequently encode it in tactics. Accordingly, a single line frequently chains multiple tactics until the intended reasoning step has been fully realized, at which point a new line is introduced. Consider the running example in Figure 2, the expert combines two tactics (i.e., `repeat rw [dotProduct]` and `simp [mul_comm]`) within a single line to ensure that the resulting proof state aligns with the underlying informal reasoning. This observation motivates us to analyze human-written proofs at the line level.

We implement a customized Lean 4 tactic `to_theorem`, which automatically encodes line-level, procedural proofsteps into a new subgoal, with the corresponding proofsteps serving as their proofs. As shown in Figure 2, the tactic operates by recording the proof states immediately before and after executing the procedural steps `repeat rw [dotProduct]; simp [mul_comm]`. It then inserts the prior state as hypotheses and the subsequent state as the goal, thereby deriving a new theorem. This theorem effectively isolates a subgoal of the original proof, which can then be discharged directly by replaying the original procedural tactics. The tactic `to_theorem` extracts the subgoal from a single line. We can naturally extend it to proof-segment extraction, where a sequence of tactics spanning multiple lines is first converted into the declarative format, and then its subgoal is extracted. For the running example `linear_gradient` in Figure 2, which totally comprises a nine-line proof, we can extract up to nine single-line subgoals; four non-overlapping two-line subgoals; and so forth.

2.2 Data Curation

We categorize machine learning theories into two main types: optimization theory and probability theory. The former typically analyzes the convergence of optimization algorithms (e.g., gradient descent), while the latter addresses error bounds of machine learning models (e.g., Hoeffding's inequality). To develop FormalML, we expand and extract data based on two projects: **Optlib** [Li et al., 2024a] and **FoML** [Sonoda et al., 2025]. For each project, we designate the top-level theorems (e.g., the convergence of gradient descent) for subgoal extraction, while preserving lower-level lemmas (e.g., Lipschitz continuity) as a local library for premise retrieval. We utilize the `to_theorem` tactic to extract the subgoals, and ultimately compile 4,937 theorems into FormalML.

Optlib [Li et al., 2024a, 2025a,b] is a project started in September 2023 that formalizes a wide range of optimization algorithms, including gradient descent (GD), subgradient method (SubGD), proximal gradient descent (PGD), Nesterov acceleration method (NAG), block coordinate descent (BCD), and alternating direction minimization method (ADMM). Building on Optlib, we extend and refine theorems related to these algorithms, and update some proofs to ensure compatibility. FoML [Sonoda et al., 2025] is a recent project initiated in March 2025, primarily focused on formalizing the generalization error bound using Rademacher complexity. It also includes some important probability inequalities, such as the expectation inequality, the bounded differences inequality, and McDiarmid's inequality. Furthermore, we formalize Hoeffding's lemma, the Bennett inequality, and the Bernstein inequality into the library to enhance its comprehensiveness. The statistic of our benchmark is shown in Appendix B.

2.3 Benchmark Challenges

We identify that the proofstep generation task remains particularly challenging for LLM-based theorem provers, attributable to the following three key reasons.

- **Complex proof context.** In practice, users often rely on automated theorem provers to close subgoals within lengthy and complex proofs. This requires LLMs to comprehend all accumulated hypotheses and intricate goals to generate accurate proofs. However, current provers pay little attention to complex contextual reasoning.
- **Premise retrieval.** Accurately retrieving relevant premises from both local and global libraries is crucial, as many proofs depend on interdependent lemmas [Hu et al., 2024]. Thus, effective retrieval is a prerequisite for practical subgoal completion. However, current LLM-based provers still perform inadequately in this regard Yang et al. [2023].
- **Overthinking and efficiency.** Unlike competition problems that demand long chains of reasoning, subgoal completion is often repetitive and involves relatively straightforward reasoning. LLMs trained for complex competitions may *overthink* such tasks, exploring unnecessary inference paths that hinder performance [Chiang and Lee, 2024, Sui et al., 2025]. Moreover, efficiency is crucial: it is impractical to expend substantial computational resources on proving subgoals that require only brief, direct proofs. Balancing reasoning depth with computational cost is thus a key challenge for practical subgoal completion.

3 Experiments

In this section, we conduct a comprehensive evaluation of state-of-the-art LLM-based theorem provers using our FormalML benchmark. The full experimental setup and results are shown in A. Our findings can be summarized as follows:

- Existing LLM-based theorem provers are inadequate as practical tools for assisting mathematicians in achieving subgoal completion *under low computational budgets*. Moreover, the performance of LLM-based theorem provers exhibits significant disparities across different specialized areas within FormalML.
- Existing LLM-based provers often struggle with premise retrieval in complex proofs, resulting in suboptimal performance. However, models like DeepSeek-Prover-V2 demonstrate superior retrieval capabilities, attributable to their strong general reasoning abilities.
- Current LLM-based theorem provers tend to exhibit significantly degraded performance when handling higher-difficulty problems in FormalML.
- Long-CoT demonstrates no significant improvement in the pass rates of the subgoal completion task while simultaneously incurring substantial computational overhead.
- It is demonstrated that expert iteration enhances performance on FormalML and shows potential for improving subgoal completion capabilities.

4 Related Work

LLM-based Theorem Proving. In recent years, the rapid development of LLMs has significantly advanced research on formal theorem proving [Li et al., 2024b]. Current mainstream approaches can be broadly divided into two categories. The first category employs tree-search strategies, including best-first search [Yang et al., 2023, Xin et al., 2025, Ying et al., 2024, Polu and Sutskever, 2020], Monte Carlo tree search [Xin et al., 2024, Wang et al., 2023, Kocsis and Szepesvári, 2006, Yang et al., 2025], and others [Lin et al., 2024]. Although these methods align with the Markov Decision Process (MDP) framework of theorem proving, their step-by-step tactic generation and frequent interaction with the Lean environment result in low search efficiency. The second category is whole-proof generation [Dong and Ma, 2025, Lin et al., 2025a, Zhang et al., 2025, Wang et al., 2025, Ren et al., 2025, Chen et al., 2025], where LLMs directly generate complete proofs, usually through multiple rollouts verified in Lean to identify the final proof. This approach has demonstrated notable advantages. For instance, DeepSeek-Prover-V2 [Ren et al., 2025] and Kimina-Preview [Wang et al., 2025] leverage natural language-aided Long-CoT reasoning to produce formal proofs, achieving breakthroughs in competition-level tasks. However, the performance of existing LLM-based theorem provers in supporting practical formal proving has not yet been fully evaluated.

Formal Theorem Proving Benchmarks. The current mainstream Lean formal theorem proving benchmarks can be categorized into three primary types based on problem characteristics. The first type focuses on Olympiad-level high school mathematics competition problems, exemplified by: the MiniF2F [Zheng et al., 2022] dataset, which comprises 244 challenging problems from competitions such as AMC, AIME, and IMO; and the PutnamBench benchmark [Tsoukalas et al., 2024], featuring problems derived from the Putnam Mathematical Competition. The second type targets undergraduate-level mathematical tasks, including ProofNet [Azerbayev et al., 2023] (containing analysis and algebra problems at the undergraduate level) and MinCTX [Hu et al., 2024] (theorems sourced from real-world Lean projects and textbooks). Additionally, hybrid benchmarks like FormalMATH [Yu et al., 2025] cover theorem proving across domains ranging from high school to undergraduate mathematics. Another category, exemplified by Leandojo [Yang et al., 2023], directly constructs datasets from the Mathlib library. Notably, existing benchmarks either operate within simple contexts or do not require premise retrieval, and all involve full proof generation tasks. In contrast, our proposed FormalML benchmark specializes in subgoal completion tasks, a design better aligned with practical applications where LLM-based theorem provers assist humans in practical theorem proving.

5 Conclusion

In this paper, we identify subgoal completion as a critical yet understudied task for helping mathematicians working on research-level problems. We introduce FormalML, the first benchmark specifically designed for this task in formal theorem proving. Using a Lean 4 translation tactic, we systematically extract 4,937 subgoal problems from foundational machine learning theory, covering optimization and probability. Evaluations of state-of-the-art LLM-based provers demonstrate persistent limitations in accuracy, efficiency, and retrieval within complex proof contexts. Moreover, expert iteration has proven to be a promising direction for advancing subgoal completion. We anticipate that FormalML will catalyze the development of more capable provers to support subgoal completion.

References

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

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

Cheng-Han Chiang and Hung-yi Lee. Over-reasoning and redundant calculation of large language models. *arXiv preprint arXiv:2401.11467*, 2024.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv e-prints*, pages arXiv–2502, 2025.

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

Jiewen Hu, Thomas Zhu, and Sean Welleck. minictx: Neural theorem proving with (long-) contexts. *arXiv preprint arXiv:2408.03350*, 2024.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *European Conference on Machine Learning*, volume 4212, pages 282–293, 2006.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Chenyi Li, Ziyu Wang, Wanyi He, Yuxuan Wu, Shengyang Xu, and Zaiwen Wen. Formalization of complexity analysis of the first-order algorithms for convex optimization. *arXiv preprint arXiv:2403.11437*, 2024a.

Chenyi Li, Zichen Wang, Yifan Bai, Yunxi Duan, Yuqing Gao, Pengfei Hao, and Zaiwen Wen. Formalization of algorithms for optimization with block structures. *arXiv preprint arXiv:2503.18806*, 2025a.

Chenyi Li, Shengyang Xu, Chumin Sun, Li Zhou, and Zaiwen Wen. Formalization of optimality conditions for smooth constrained optimization problems. *arXiv preprint arXiv:2503.18821*, 2025b.

Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. A survey on deep learning for theorem proving. In *First Conference on Language Modeling*, 2024b.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-STaR: Learning to Interleave Thinking and Proving. *arXiv preprint arXiv:2407.10040*, 2024.

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

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

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *Journal of machine learning research*, 22(164):1–20, 2021.

Stanislas Polu and Ilya Sutskever. Generative Language Modeling for Automated Theorem Proving. *arXiv preprint arXiv:2009.03393*, 2020.

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

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Marco Dos Santos, Haiming Wang, Hugues de Saxcé, Ran Wang, Mantas Baksys, Mert Unsal, Junqi Liu, Zhengying Liu, and Jia Li. Kimina lean server: Technical report. *arXiv preprint arXiv:2504.21230*, 2025.

Sho Sonoda, Kazumi Kasaura, Yuma Mizuno, Kei Tsukamoto, and Naoto Onda. Lean formalization of generalization error bound by rademacher complexity. *arXiv preprint arXiv:2503.19605*, 2025.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

Donald Syme. Declare: A prototype declarative proof system for higher order logic. Technical report, University of Cambridge, Computer Laboratory, 1997.

Terence Tao, Pietro Monticone, and Shreyas Srinivas. Equational theories project: Mapping out the relations between different equational theories of magmas. `https://teorth.github.io/equational_theories/`, 2024.

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. *arXiv preprint arXiv:2407.11214*, 2024.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, et al. DT-Solver: Automated Theorem Proving with Dynamic-Tree Sampling Guided by Proof-level Value Function. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2023.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Freek Wiedijk. A synthesis of the procedural and declarative styles of interactive theorem proving. *Logical Methods in Computer Science*, 8, 2012.

Terry Winograd. Frame representations and the declarative/procedural controversy. In *Representation and understanding*, pages 185–210. Elsevier, 1975.

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

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 2025.

Yixuan Even Xu, Fei Fang, Jakub Tomczak, Cheng Zhang, Zhenyu Sherry Xue, Ulrich Paquet, and Danielle Belgrave. Neurips 2024 experiment on improving the paper-reviewer assignment.

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

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2023.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*, 2024.

Xiao-Wen Yang, Zhi Zhou, Haiming Wang, Aoxue Li, Wen-Da Wei, Hui Jin, Zhenguo Li, and Yu-Feng Li. Carts: Advancing neural theorem proving with diversified tactic calibration and bias-resistant tree search. In *The Thirteenth International Conference on Learning Representations*, 2025.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, et al. InternLM-Math: Open Math Large Language Models Toward Verifiable Reasoning. *arXiv preprint arXiv:2402.06332*, 2024.

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

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover: Posttraining scaling in formal reasoning. *arXiv preprint arXiv:2504.06122*, 2025.

Lichen Zhang, Shuai Lu, and Nan Duan. Selene: Pioneering automated proof in software verification. *arXiv preprint arXiv:2401.07663*, 2024.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: A Cross-System Benchmark for Formal Olympiad-Level Mathematics. In *Proceedings of the International Conference on Learning Representations*, 2022.

Appendix

Table 2: Performance comparison of LLM-based theorem provers on FormalML. The best results are in bold, and the second-best are underlined.

Method	Model Size	Sample Budget	Pass@K (%)		
			Optim.	Prob.	All
Best-First Tree Search Methods					
Reprover [Yang et al., 2023]	229M	4×20	21.80	16.15	19.48
		8×50	27.82	20.44	24.79
BFS-Prover [Xin et al., 2025]	7B	4×20	24.45	19.31	22.26
		8×50	27.62	23.84	25.31
Whole-Proof Generation Methods					
Kimina-Prover-Preview-7B [Wang et al., 2025]	7B	1	17.99	11.77	15.43
		4	30.75	23.69	27.85
		16	42.31	38.13	40.59
		32	47.09	43.79	45.74
Goedel-Prover [Lin et al., 2025a]	7B	1	8.22	13.50	10.39
		4	21.32	28.97	24.47
		16	37.29	41.28	38.93
		32	44.34	46.11	45.07
Goedel-Prover-V2-8B [Lin et al., 2025b]	7B	1	20.85	19.31	20.21
		4	34.61	33.15	34.01
		16	45.13	43.35	44.40
		32	49.67	48.67	49.26
Leanabell-Prover [Zhang et al., 2025]	7B	1	24.11	24.09	24.10
		4	47.33	43.30	45.68
		16	58.00	50.84	55.05
		32	61.23	53.99	58.07
STP [Dong and Ma, 2025]	7B	1	28.45	24.83	26.96
		4	51.01	46.06	48.98
		16	62.13	<u>57.73</u>	60.32
		32	**65.19**	**60.39**	**63.21**
DeepSeek-Prover-V1.5 [Xin et al., 2024]	7B	1	24.29	17.29	21.41
		4	43.31	37.39	40.88
		16	57.48	52.56	55.46
		32	61.51	56.00	59.25
DeepSeek-Prover-V2 (noCoT) [Ren et al., 2025]	7B	1	17.16	16.40	16.85
		4	37.98	37.09	37.61
		16	59.79	52.76	56.90
		32	<u>65.08</u>	<u>57.73</u>	<u>62.06</u>
DeepSeek-Prover-V2 (CoT) [Ren et al., 2025]	7B	1	16.23	21.58	18.43
		4	25.73	32.56	28.54
		16	33.54	39.36	35.93
		32	37.39	42.51	39.50

A Full Experiments

A.1 Evaluating LLM-based Theorem Provers on FormalML

Theorem Provers We primarily focus on the two most effective types of LLM-based provers:

- **Best-First Tree-Search (BFS) Methods:** Each node in the search tree corresponds to a proof state, and a heuristic scoring function assigns priorities to nodes. The BFS algorithm is employed to explore the search space and derive the final proof. We evaluate two models in this framework: Reprover [Yang et al., 2023] and BFS-Prover [Xin et al., 2025].
- **Whole-Proof Generation Methods:** These types of models generate a complete formal proof in a single pass directly from the problem description, eliminating the need for additional state transitions or search procedures. We consider the following SOTA models: STP [Dong and Ma, 2025], Goedel-Prover [Lin et al., 2025a], Godel-Prover-V2 [Lin et al., 2025b], Leanabell-Prover [Zhang et al., 2025], Kimina-Prover-Preview-7B [Wang et al., 2025], DeepSeek-Prover-V1.5 [Xin et al., 2024] and DeepSeek-Prover-V2 [Ren et al., 2025].

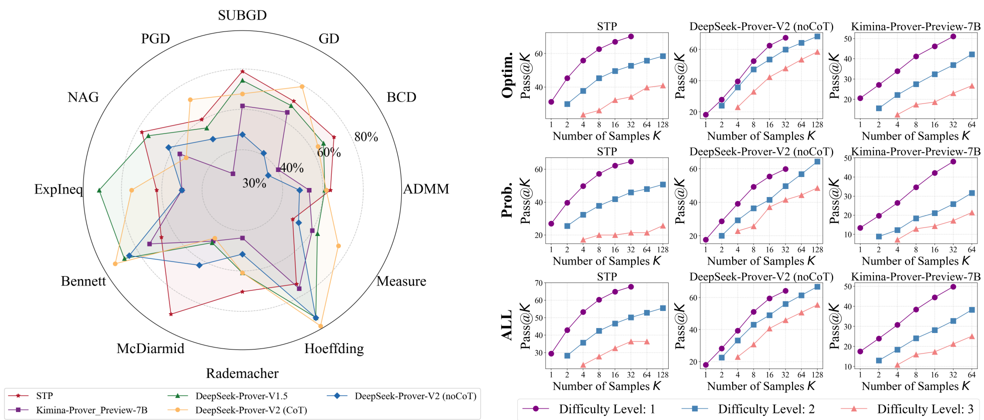

Figure 3: The left figure presents results of pass rate across various specific problem domains, while the right figure shows the performances under different difficulty levels.

Metrics & Evaluation We employ the Pass@K metric to evaluate theorem provers, where K represents the computational budget. This metric quantifies the proportion of problems for which at least one valid proof is discovered within the top K generated attempts. For BFS methods, $K = E \times T$, where E represents the number of tactics generated per expansion, and T denotes the number of expansion iterations. Unlike prior studies that predominantly focus on competition-level theorem proving—characterized by highly challenging problems and lengthy proofs, thereby requiring large values of K, this work concentrates on the research level subgoal completion. Consequently, our benchmark emphasizes low computational budgets, corresponding to small values of K ($\leq 8 \times 50$ for BFS methods and ≤ 32 for whole-proof generation methods).

Experimental Details For BFS methods, we implement and conduct interactive search based on the LeanDojo [Yang et al., 2023] framework. For whole-proof generation methods, we employed vLLM [Kwon et al., 2023] to generate proofs and evaluated their correctness using kimina-lean-server [Santos et al., 2025], which provides high-throughput Lean 4 verification. All experiments strictly adhered to the published parameter configurations and prompts for each model. All computations were performed on NVIDIA H800 GPUs.

Main Results In Table 2, we show the performances of current theorem provers on our FormalML benchmark. Our experiments demonstrate that current whole-proof generation models have preliminarily acquired the capability to solve FormalML problems when increasing sample budgets, STP attaining the highest Pass@32 score of 63.21%. However, their Pass@1 performances remains at a relatively low level (peaking at merely 26.96%), falling short of the practical requirements for assisting mathematicians in completing Lean formal proofs. Notably, the BFS tree search strategy, despite consuming more computational resources, does not yield significant performance improvements (remaining below 30%). Additionally, we observe that: 1) the performance ranking of models exhibits inconsistencies with the results from the miniF2F benchmark, suggesting that certain models exhibit varying degrees of overfitting to competition-level elementary theorem-proving tasks; 2) recent long-CoT provers (e.g. Kimina-Preview and Deepseek-Prover-V2) incorporating natural language assistance underperform on our FormalML benchmark (with pass@32 consistently below 50%), presenting a stark contrast to their strong performance on miniF2F.

Performance Distribution We present the distributional results of performance across various specific problem domains (e.g., the proof of Hoeffding's inequality). The experimental results presented in left panel of Figure 3 demonstrate that the performance of existing models varies across different specialized areas. For instance, in the probability split, STP achieves significantly superior results on Mcdiarmid, far surpassing other methods. On Hoeffding, Deepseek-Prover-V2 (noCoT) delivers the best performance.

A.2 Evaluating Retrieval Effectiveness on FormalML

While writing practical proof, humans often need to retrieve premises from local theorem libraries or Mathlib. The experiments are conducted on the retrieval subset of our benchmark, with the following specific design: for each proposition to be proved, we randomly sample candidate premises from a combined premise library comprising Mathlib and local premises, mixing in actually correct theorems

Table 3: Performances of retrieval and then proof. Relative improvements compared to $M = 0$ are shown in green (increase) or red (decrease).

Method	**M = 0**		**M = 10**		**M = 20**	
	Pass@16(%)	Pass@32(%)	Pass@16(%)	Pass@32(%)	Pass@16(%)	Pass@32(%)
STP	54.56	57.72	52.23 (-2.33)	56.43 (-1.29)	52.17 (-2.39)	56.04 (-1.68)
Goedel-Prover	32.97	39.04	34.58 (+1.61)	41.05 (+2.01)	33.35 (+0.38)	39.50 (+0.46)
Goedel-Prover-V2-8B	45.24	51.26	45.05 (-0.19)	49.90 (-1.36)	46.09 (+0.85)	50.36 (-0.90)
Leanabell-Prover	46.67	50.42	47.45 (+0.78)	50.87 (+0.45)	46.93 (+0.26)	50.29 (-0.13)
Kimina-Prover-Preview-7B	34.71	39.56	34.65 (-0.06)	39.69 (+0.13)	35.49 (+0.78)	39.75 (+0.19)
Deepseek-Prover-V1.5	49.45	52.94	46.86 (-2.59)	53.59 (+0.65)	42.79 (-6.66)	49.45 (-3.49)
Deepseek-Prover-V2 (noCoT)	53.46	58.37	58.82 (+5.36)	69.29 (+10.92)	57.40 (+3.94)	65.80 (+7.43)
Deepseek-Prover-V2 (CoT)	34.39	37.94	44.34 (+9.95)	47.71 (+9.77)	42.73 (+8.34)	46.74 (+8.80)

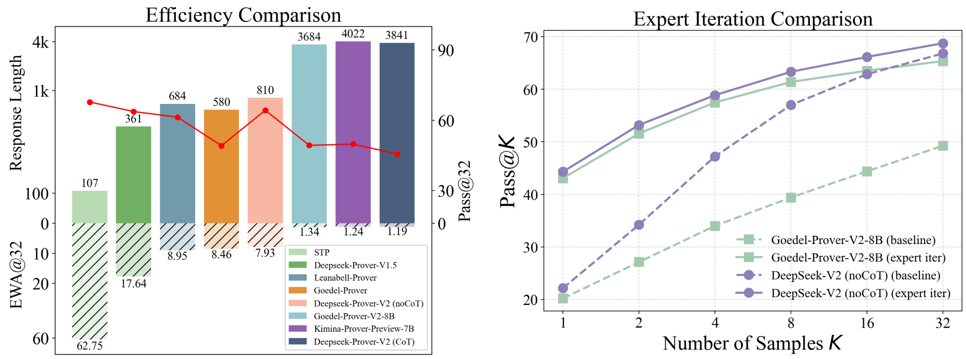

Figure 4: The left figure illustrates the efficiency comparison among current whole-proof generation provers, while the right shows performances before and after expert iteration.

to ultimately form a retrieval set containing M candidate theorems. All candidate theorems are presented in a structured format within the model's context (the complete prompt template is provided in Appendix G). Considering the current context window limitations of LLM-based provers, this study adopts two experimental configurations ($M = 10$ and $M = 20$) for comparative analysis. Our study intentionally omits a comparative evaluation of embedding-based retrieval models. This design choice stems from the consideration that using the same model for both retrieval and proof generation to enhance efficiency in the subgoal completion task.

Results are demonstrated in Table 3. We observe that most models exhibit improved performance when provided with candidate premises, yet their effectiveness declines as M increases (indicating higher difficulty). Although STP demonstrates strong performance across the full benchmark, it underperforms in premise retrieval. We attribute this to the absence of such data in its training set. We identify a several representative case in Appendix H.1. In contrast, DeepSeek-Prover-V2 achieves robust results under both CoT and noCoT settings, with a high performance gain of approximately 10% at $M = 10$. This indicates its strong retrieval capability, which we think stems from its training on extensive natural language reasoning data, thereby enhancing its general reasoning ability.

A.3 Evaluating FormalML of Different Difficulty Level

We evaluate the performance of existing whole-proof generation models at varying difficulty levels on FormalML. A larger K is used to assess problems of higher difficulty. As shown in right of Figure 3, experimental results indicate that model performance, both for the optimization and the probability split, decreases as difficulty increases. Notably, STP still achieves the highest results under difficulty levels 3 and 5, with Pass@128 scores of 55.5% and 33.36%, respectively. Results of other models are shown in Appendix E.2.

A.4 Evaluating Theorem Prover Efficiency on FormalML

In formal theorem proving, efficient provers are crucial for practical applications. Recent studies demonstrate that RL based long-CoT reasoning has been successfully integrated into systems such as DeepSeek-Prover-V2, Kimina-Prover and Goedel-Prover-V2. These models employ deep natural-language reasoning before generating formal proofs. While they exhibit strong performance on competition-level mathematical problems, our experiments reveal no significant advantage on the FormalML benchmark (See Table 2). Moreover, these models generally suffer from low reasoning efficiency. To address the trade-off between reasoning efficiency and accuracy in FormalML sce-

Table 4: Statistics of theorems in FormalML across various machine learning theories.

Optimization	GD	SubGD	PGD	NAG	BCD	ADMM	Other	Total
# Problems	211	331	388	528	433	1,016	0	2,907
Probability	Exp	Bennett	McDiarmid	Rademacher	Hoeffding	Measure	Other	Total
# Problems	100	136	642	615	52	315	170	2,030

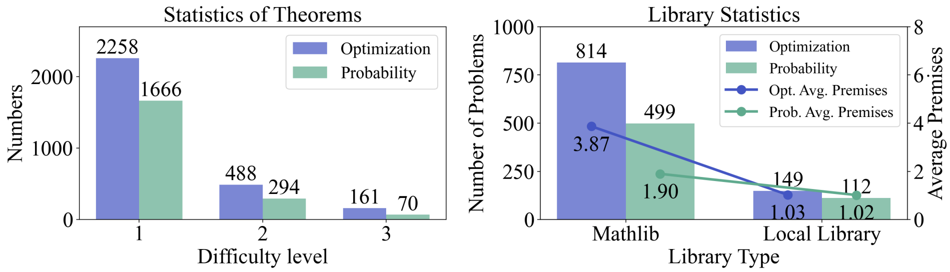

Figure 5: Statistics of FormalML theorems: (Left) distribution by proving difficulty; (Middle) retrieval sources and averages; (Right) average number of retrieved premises by difficulty.

narios, we use a novel evaluation metric—Efficiency-Weighted Accuracy (EWA@K)—defined as: EWA@K = Pass@$K \times \frac{100}{\text{Response Length}}$. This metric balances performance and output efficiency by normalizing the proof success rate against the response length. To the right of Figure 4, we compare the pass rates and efficiency of eight whole-proof generation models on FormalML. The results show that the three long-COT models generate the highest average number of response tokens, far exceeding other models. However, their performance is inferior to others, resulting in the lowest EWA@32 score. In contrast, STP achieves both the shortest output length and the highest pass rate on the FormalML benchmark, leading to the highest EWA@32 score.

A.5 Evaluating FormalML with Expert Iteration

We further explored the performance of expert iteration on FormalML. We extracted 92,815 problems from five repositories (mathlib, PrimeNumberTheoremAnd, PFR, PhysLean, and scilean) using the `to_theorem` tactic, of which 88,174 were used for expert iteration training. For each problem, we generated 8 candidate proofs and sampled the correct ones for training. We performed 1 round of expert iteration on DeepSeek-Prover-V2 (noCoT) and Goedel-Prover-V2-8B. Training details are shown in Appendix F. Results are shown in the right panel of Figure 4. The experimental results indicate a substantial improvement in performance after expert iteration, especially in Pass@1. This suggests that expert iteration holds potential for enhancing subgoal completion tasks.

B Benchmark Statistics

Format. Each theorem in FormalML is stored in JSON format (See Appendix C), including:

(1) Source location metadata (file path and line number coordinates for theorem extraction).
(2) The formal Lean 4 theorem statement.
(3) Required module imports and namespace declarations.
(4) The complete tactic sequence constituting the original proof.
(5) Retrieved proof-relevant premises from the mathematical context.

Summary. We establish FormalML by systematically expanding and extracting theorems from two Lean 4 libraries: Optlib and FoML. The resulting benchmark comprises 4,937 unique theorems, each derived from a single proof step or proof segment within the top-level theorems presented in these libraries. For the extracted theorems, we categorize them according to their corresponding top-level theorems, as illustrated in Table 4. The results indicate that the benchmark encompasses a diverse range of theories in machine learning. The number of extracted theorems associated with each top-level theorem depends on the length of the corresponding proof. On average, hundreds of theorems can be successfully extracted from each top-level theorem.

Proving difficulty. To enable a fine-grained evaluation of LLM-based theorem provers, we categorize theorems based on their proving difficulty. Specifically, since the proofs in FormalML are mostly procedural style, we can use the proof length as a metric of proving difficulty [Zhang et al., 2024]. We define three levels of proving difficulty, corresponding to proof lengths of 1, 3, and 5, respectively, and report the statistics of theorems across these levels in the left panel of Figure 5. In total, there are 3,924 theorems at difficulty level 1 and 1,013 theorems at higher difficulty levels.

Retrieval difficulty. We further examine the retrieval difficulty within FormalML. In total, 2,049 theorems require explicit premise retrieval. These theorems are divided into two categories based on the source of the retrieved premises: Mathlib and the local library. The middle panel of Figure 5 reports the number of theorems in each category as well as the average number of premises. Furthermore, the right panel of Figure 5 summarizes the average number of retrieved premises required at different difficulty levels, showing that more complex proofs generally demand greater retrieval effort.

C Data Format Specification

The FormalML dataset stores each theorem in a structured JSON format. Below is a detailed schema with field descriptions.

```
{
  "filename": string,
  "line": int,
  "tactic_state_before": string,
  "tactic": string,
  "tactic_state_after": string,
  "goal": string,
  "theorem_header": string,
  "formal_statement": string,
  "full_formal_statement": string,
  "retrieval": [
    {
      "library": string,
      "definition": string
    }
  ]
}
```

Based on the file `FoML/FoML/ForMathlib/Probability/Moments.lean` (Line 98), we further provide an illustrative example for some typical entries. In this line, the corresponding proof tactic is `apply aemeasurable_expt_hX`. By extracting the pre- and post-tactic states, we can formulate the corresponding subgoal as follows:

```
theorem extracted_formal_statement_27.{u_1}
{Ω : Type u_1}
{m : MeasurableSpace Ω}
{μ : Measure Ω}
[inst : IsFiniteMeasure μ]
(a b : ℝ)
{X : Ω → ℝ}
(hX : AEMeasurable X μ)
(h : ∀ᵐ (ω : Ω) ∂μ, X ω ∈ Set.Icc a b) :
let e := fun t ω => exp(t · Xω);
∀ (t' : ℝ), AEStronglyMeasurable (e t') μ := sorry
```

Regarding the retrieval entry, it records the library (e.g., `FoML`), and specifies the definition of the retrieved lemma. For example:

```
lemma aemeasurable_expt
{X : Ω → ℝ}
(t : ℝ)
(hX : AEMeasurable X μ) :
AEStronglyMeasurable (fun ω ↦ exp(t · X(ω))) μ
```

Table 5: Error type analysis for different theorem provers under pass@16 and pass@32.

Setting	Model	has_error	is_valid_with_sorry	is_valid_no_sorry
Pass@16: Optimization	DeepSeek-Prover-V1.5	74.80	0.02	25.16
	DeepSeek-Prover-V2 (CoT)	63.16	10.68	15.49
	Goedel-Prover	90.08	0.00	9.92
Pass@16: Probability	DeepSeek-Prover-V1.5	80.04	0.04	19.89
	DeepSeek-V2 (CoT)	61.94	7.65	22.77
	Goedel-Prover	85.20	0.00	14.80
Pass@32: Optimization	DeepSeek-Prover-V1.5	75.09	0.02	24.87
	DeepSeek-Prover-V2 (CoT)	62.98	10.76	15.49
	Goedel-Prover	90.11	0.00	9.89
Pass@32: Probability	DeepSeek-Prover-V1.5	79.88	0.04	20.04
	DeepSeek-V2 (CoT)	61.82	7.61	22.96
	Goedel-Prover	85.23	0.00	14.77

D The Full Proof of Lemma `linear_gradient`

```
private lemma linear_gradient :
∀ x : (EuclideanSpace ℝ (Fin n)),
HasGradientAt (fun x : (EuclideanSpace ℝ (Fin n)) =>
(b ●ᵥ (A *ᵥ x)))(Aᵀ *ᵥ b) x := by
  intro x
  rw [HasGradient_iff_Convergence_Point]
  intro a  apos
  use a ; use apos
  intro y _
  rw [dot_mul_eq_transpose_mul_dot,
    dot_mul_eq_transpose_mul_dot, ← dotProduct_sub]
  rw [EuclideanSpace.inner_eq_star_dotProduct]; simp
  repeat rw [dotProduct]; simp [mul_comm]
  apply mul_nonneg; linarith [apos]; apply norm_nonneg
```

E Additional Results

E.1 Error Analysis

This section presents a detailed breakdown of error types from additional experiments conducted on three different theorem provers: DeepSeek-Prover-V1.5, DeepSeek-Prover-V2 (CoT), and Goedel-Prover. The results are categorized into three main types: `has_error` (Lean code with execution errors), `is_valid_with_sorry` (code that verifies but contains the sorry tactic), and `is_valid_no_sorry` (correctly verified proofs). These proportions were calculated for both pass@16 and pass@32 metrics across two different problem domains: Optimization and Probability. The tables below provide a comprehensive overview of these results, allowing for a direct comparison of the performance characteristics of each prover.

The results indicate that long-CoT models such as DeepSeek-V2-CoT tend to output "sorry" more frequently, but demonstrate lower rates of Lean errors. Conversely, Goedel-Prover seldom outputs "sorry" tactic but exhibit higher Lean error frequencies.

E.2 Results Under Varying Difficulty Levels

We supplement the experimental results for all eight whole-generation models across different difficulty levels, as presented in Figure 6 and Figure 7. The results demonstrate a statistically significant trend where all models exhibit performance degradation as the difficulty level increases. Among all evaluated models, DeepSeek-Prover-V2 demonstrates the most modest performance decline.

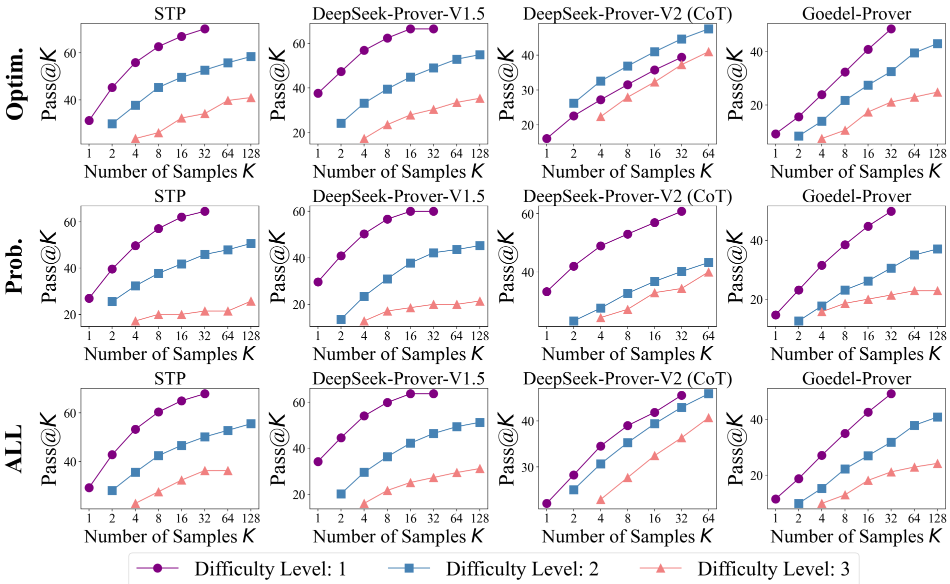

Figure 6: Results of pass rate under varying difficulty levels (Part 1).

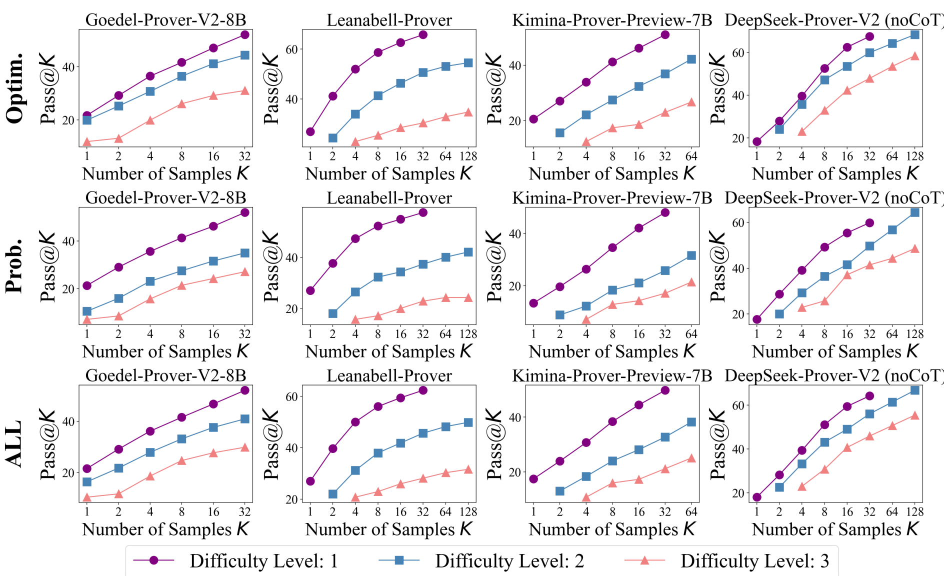

Figure 7: Results of pass rate under varying difficulty levels (Part 2).

F Training Details of Expert Iteration

Training used the AdamW optimizer with a learning rate of 1e-5, cosine learning rate schedule, warmup ratio of 0.1, batch size of 8 with gradient accumulation steps of 2 (effective batch size 16), and 3 epochs, with bf16 precision and DeepSpeed ZeRO for distributed optimization. All experiments were conducted using GPU acceleration, and evaluation was carried out on the FormalML dataset.

G Prompts for Retrieval

For premise retrieval, we designed the following prompt, using DeepSeek-Prover-V2 as an illustrative example.

The prompt for premise retrieval of DeepSeek-Prover-V2 (CoT)

You can use some of the following lemmas or theorems: {all_definitions}
Complete the following Lean 4 code:
{formal_statement}
Before producing the Lean 4 code to formally prove the given theorem, provide a detailed proof plan outlining the main proof steps and strategies.
The plan should highlight key ideas, intermediate lemmas, and proof structures that will guide the construction of the final formal proof.

H Case Study

H.1 Cases of Retrieval

We present a case of STP where the problem was initially solved without retrieval, but errors emerged after candidate premises were provided.

Problem

```
theorem extracted_formal_statement_162
{n m : ℕ+}
{A : Matrix (Fin ↑m) (Fin ↑n) ℝ}
{μ : ℝ}
{μpos : 0 < μ}
{Ane0 : A ≠ 0}
(x : EuclideanSpace ℝ (Fin ↑n))
(a : x ∈ Set.univ)
(ε : ℝ)
(εpos : ε > 0)
(y : EuclideanSpace ℝ (Fin ↑n))
(ydist : ‖x − y‖ ≤ ε / ↑↑n / μ)
(h_original_goal : |μ * Σ x, |y x| − μ * Σ x_1, |x x_1|| ≤ ε ) :
|μ * Σ x_1, (|y x_1| − |x x_1|)| ≤ ε := sorry
```

Additional Theorems

```
theorem norm_one_proximal
(lasso : h = fun y => μ · ‖y‖₁)
(x : EuclideanSpace ℝ (Fin n))
(xm : EuclideanSpace ℝ (Fin n))
(tpos : 0 < t)
(μpos : 0 < μ)
(minpoint : ∀ i : Fin n, xm i = Real.sign (x i) * (max (abs (x i) − t * μ) 0)) :
prox_prop (t · h) x xm
theorem map_map
(g : S →+* T)
(f : R →+* S) :
(s.map f).map g = s.map (g.comp f)
theorem inner_add_left
(x y z : F) :
<<x + y, z>> =<<x, z>> + <<y, z>>
theorem mul_pos
{a b : Ordinal}
(h₁ : 0 < a)
(h₂ : 0 < b) :
0 < a * b
lemma id :
(fst φ).1.comp (inl φ) (add_neg_cancel 1) +
(snd φ).comp (Cochain.ofHom (inr φ)) (add_zero 0) =
Cochain.ofHom (1 _)
theorem mul_sub
(a b c : Ordinal) :
a * (b − c) = a * b − a * c
theorem id :
(id V).NormNoninc
```

```
theorem smul_add
[Zero R]
[DistribSMul R V]
(x : HahnSeries Γ R)
(y z : HahnModule Γ' R V) :
x · (y + z) = x · y + x · z
theorem add_le_add_iff_right
{a b : Ordinal} :
∀ n : ℕ, a + n ≤ b + n ↔ a ≤ b
theorem add_right_cancel
{a b : Ordinal}
(n : ℕ) :
a + n = b + n ↔ a = b
theorem abs_abs
{x y : R}
(h : IsCoprime x y) :
IsCoprime |x| |y|
```

Correct solution without retrieval (STP)

```
import Mathlib
import Optlib

open Set Real Matrix Finset

open Set Real Matrix Finset NNReal
theorem extracted_formal_statement_162
{n m : ℕ+}
{A : Matrix (Fin ↑m) (Fin ↑n) ℝ}
{μ : ℝ}
{μpos : 0 < μ}
{Ane0 : A ≠ 0}
(x : EuclideanSpace ℝ (Fin ↑n))
(a : x ∈ Set.univ)
(ε : ℝ)
(εpos : ε > 0)
(y : EuclideanSpace ℝ (Fin ↑n)
(ydist : ‖x - y‖ ≤ ε / ↑↑n / μ)
(h_original_goal : |μ * Σ x, |y x| - μ * Σ x_1, |x x_1|| ≤ ε ) :
|μ * Σ x_1, (|y x_1| - |x x_1|)| ≤ ε := by
  cases' le_total (μ * Σ x_1, |y x_1|)
    (μ * Σ x_1, |x x_1|) with h h <;>
  simp_all [abs_of_nonneg, abs_of_nonpos, mul_add, mul_sub,
sub_add_eq_add_sub]
  <;> linarith
```

Incorrect solution with retrieval (STP)

```
import Mathlib
import Optlib

open Set Real Matrix Finset

open Set Real Matrix Finset NNReal

theorem extracted_formal_statement_162
{n m : ℕ+}
{A : Matrix (Fin ↑m) (Fin ↑n) ℝ}
{μ : ℝ}
{μpos : 0 < μ}
{Ane0 : A ≠ 0}
(x : EuclideanSpace ℝ (Fin ↑n))
```

```
(a : x ∈ Set.univ)
(ε : ℝ)
(εpos : ε > 0)
(y : EuclideanSpace ℝ (Fin ↑n))
(ydist : ‖x − y‖ ≤ ε / ↑↑n / μ)
(h_original_goal : |μ * Σ x, |y x| − μ * Σ x_1, |x x_1|| ≤ ε) :
|μ * Σ x_1, (|y x_1| − |x x_1|)| ≤ ε := by

  have h₀ : (ε / ↑↑n / μ) * μ ≤ ε := by
    have h₁ : 0 < (↑↑n : ℝ) := by exact_mod_cast
      (Nat.pos_of_ne_zero (fun h ↦ by simp_all))
    have h₂ : 0 < μ := by exact_mod_cast μpos
    field_simp [h₁, h₂]
    <;> rw [div_le_iff (by positivity)] <;> nlinarith

  have h₁ : |μ * Σ x_1, (|y x_1| − |x x_1|)| ≤
    μ * Σ x_1, |y x_1 − x x_1| := by
    have h₂ : ∀ x_1, |μ * (|y x_1| − |x x_1|)| =
    μ * |y x_1 − x x_1| := by
      intro x_1
      rw [abs_mul, abs_of_pos (by exact_mod_cast μpos)]
    simp_all [abs_sum]

  have h₂ : μ * Σ x_1, |y x_1 − x x_1| ≤ ε := by
    calc
      μ * Σ x_1, |y x_1 − x x_1| ≤ μ * Σ x_1, ‖x − y‖ := by
        exact mul_le_mul_of_nonneg_left (by
          exact Finset.sum_le_sum (fun i _ ↦ abs_le_norm
            (y i − x i))) (by positivity)
      _ = (μ * Σ x_1 : Fin ↑n, ‖x − y‖) := by
        simp_all [Finset.sum_const]
      _ = (↑↑n * ‖x − y‖) * μ := by ring
      _ ≤ (↑↑n * (ε / ↑↑n / μ)) * μ := by gcongr
      _ = ε := by field_simp [npos, μpos] <;> ring

  linarith
  <;> simp_all
  <;> linarith
```

H.2 Cases of Long-CoT

We present the output results of the Long-CoT model on our benchmark. The model exhibits redundant responses and suboptimal performance, which stems from issues such as hallucination and overthinking.

Problem

```
theorem extracted_formal_statement_1.{u_3, u_2, u_1}
{E₁ : Type u_1}
{E₂ : Type u_2}
{F : Type u_3}
[inst : NormedAddCommGroup E₁]
[inst_1 : InnerProductSpace ℝ E₁]
[inst_2 : FiniteDimensional ℝ E₁]
[inst_3 : NormedAddCommGroup E₂]
[inst_4 : InnerProductSpace ℝ E₂]
[inst_5 : FiniteDimensional ℝ E₂]
[inst_6 : NormedAddCommGroup F]
[inst_7 : InnerProductSpace ℝ F]
[inst_8 : FiniteDimensional ℝ F]
{admm : ADMM E₁ E₂ F}
{admm_kkt : Existance_of_kkt admm}
(n : ℕ+)
{|xx : E₁|}
(a : xx ∈ univ) {|yy : E₁|}
```

```
(a_1 : yy ∈ univ)
{|aa bb : ℝ|}
(a_2 : 0 ≤ aa)
(a_3 : 0 ≤ bb)
(abh : aa + bb = 1)
(h_original_goal :
aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
(OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
OptProblem.b E₁ E₂ =
aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
(aa + bb) · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
OptProblem.b E₁ E₂)) :
aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
(OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
OptProblem.b E₁ E₂ =
aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
1 · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
OptProblem.b E₁ E₂) := by
```

Model Response (Kimina-Prover-Preview-7B)

Solving the Formal Statement in Lean 4

I'm going to solve the problem of proving that
`aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
  OptProblem.b E₁ E₂ =
aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
  1 · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) – OptProblem.b E₁ E₂)`

given the assumptions in the problem.

Let's think about this algebraically first. We have:
– `aa + bb = 1` (from the hypothesis `abh`)
– `0 ≤ aa` and `0 ≤ bb` (from `a_2` and `a_3`)
– The equation to prove is of the form:
  `x + y + (1 · z) = x + y + z`, where `x = aa · (OptProblem.A₁ E₂) xx`, `y = bb · (OptProblem.A₁ E₂) yy`, and `z = (OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) – OptProblem.b E₁ E₂`

This is just algebraic manipulation. Since `1 · z = z` for any scalar `z`, the equation simplifies to showing that
`x + y + (aa + bb) · z = x + y + z` where `z = (OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natpred) – OptProblem.b E₁ E₂`

But since `aa + bb = 1`, we have:
`x + y + (aa + bb) · z = x + y + 1 · z = x + y + z`

So the left side of the equation becomes `x + y + z`, which is exactly the right side.

In Lean, we can prove this directly from the hypothesis `h_original_goal` by simplifying the expression. Let's complete the proof:

```
import Mathlib
import Optlib


open Set InnerProductSpace Topology Filter InnerProduct

open scoped Pointwise
theorem extracted_formal_statement_1.{u_3, u_2, u_1} {E₁ : Type u_1} {E₂ : Type u_2} {F : Type u_3}
  [inst : NormedAddCommGroup E₁] [inst_1 : InnerProductSpace ℝ E₁] [inst_2 : FiniteDimensional ℝ E₁]
```

[inst_3 : NormedAddCommGroup E_2] [inst_4 : InnerProductSpace ℝ E_2] [inst_5 : FiniteDimensional ℝ E_2]
[inst_6 : NormedAddCommGroup F] [inst_7 : InnerProductSpace ℝ F] [inst_8 : FiniteDimensional ℝ F]
{admm : ADMM E_1 E_2 F} {admm_kkt : Existence_of_kkt admm} (n : ℕ+) {|xx : E_1|} (a : xx ∈ univ) {|yy : E_1|}
(a_1 : yy ∈ univ) {|aa bb : ℝ|} (a_1 : 0 ≤ aa) (a_2 : 0 ≤ bb) (abh : aa + bb = 1)
(h_original_goal :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
(aa + bb) · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natpred) − OptProblem.b E_1 E_2)) :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natpred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natpred) − OptProblem.b E_1 E_2) := by

Now we have the hypothesis `h_original_goal` which states that:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2 =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (aa + bb) · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

We need to prove that:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2 =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + 1 · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

From algebra, we know that if `aa + bb = 1`, then `(aa + bb) · z = 1 · z` for any scalar `z`. So we can rewrite the right side of `h_original_goal` as:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (aa + bb) · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2) =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + 1 · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

And the left side is exactly what we need to prove. So we can rewrite the goal using the given hypothesis:

have h1 : aa + bb = 1 := abh

Now we can rewrite the goal using the equation in `h_original_goal` and the fact that `aa + bb = 1`:

From `h_original_goal`:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2 =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (aa + bb) · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

Substituting `h1` into the right side:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (aa + bb) · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2) =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + 1 · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

So the goal becomes proving:
`aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + (A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2 =
aa · (A_1 E_2) xx + bb · (A_1 E_2) yy + 1 · ((A_2 E_1) (x_2 E_1 F n.natpred) − b E_1 E_2)`

Which is exactly what `h_original_goal` states with the substitution of `h1`. Let's complete the proof:

linarith [h_original_goal, h1]

The `linarith` tactic is powerful enough to solve goals involving linear arithmetic. Here, it's using `h_original_goal` which relates the two sides of our goal via the equality given, and `h1` which states that `aa + bb = 1`, to rewrite the goal directly.

The key insight is that:
– Left side of goal: `aa · ($A_1$ $E_2$) xx + bb · ($A_1$ $E_2$) yy + ($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) − b $E_1$ $E_2$`

– Right side of goal after substitution: `aa · ($A_1$ $E_2$) xx + bb · ($A_1$ $E_2$) yy + 1 · (($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$)`

From `h_original_goal`, we have:
`left side = right side` where `right side = left side + (aa + bb) · (($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$) – 1 · (($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$)`

But since `aa + bb = 1`, we get:
`left side = left side + 1 · (($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$) – 1 · (($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$)`

Which simplifies to:
`left side = left side`

Therefore, `linarith` can directly solve the goal by substituting the given equality and the fact that `aa + bb = 1`.

The complete proof is remarkably concise because the algebraic manipulation is straightforward. The key insight is recognizing that the equation we need to prove is precisely the original goal with the substitution of `aa + bb = 1` into the term involving `($A_2$ $E_1$) ($x_2$ $E_1$ F n.natpred) – b $E_1$ $E_2$`.

```
import Mathlib
import Optlib


open Set InnerProductSpace Topology Filter InnerProduct

open scoped Pointwise
theorem extracted_formal_statement_1.{u_3, u_2, u_1} {E₁ : Type u_1} {E₂ : Type u_2} {F : Type u_3}
  [inst : NormedAddCommGroup E₁] [inst_1 : InnerProductSpace ℝ E₁] [inst_2 : FiniteDimensional ℝ E₁]
  [inst_3 : NormedAddCommGroup E₂] [inst_4 : InnerProductSpace ℝ E₂] [inst_5 : FiniteDimensional ℝ E₂]
  [inst_6 : NormedAddCommGroup F] [inst_7 : InnerProductSpace ℝ F] [inst_8 : FiniteDimensional ℝ F]
  {admm : ADMM E₁ E₂ F} {admm_kkt : Existence_of_kkt admm} (n : ℕ+) ⦃xx : E₁⦄ (a : xx ∈ univ) ⦃yy : E₁⦄
  (a_1 : yy ∈ univ) ⦃aa bb : ℝ⦄ (a_1 : 0 ≤ aa) (a_2 : 0 ≤ bb) (abh : aa + bb = 1)
  (h_original_goal :
    aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) –
        OptProblem.b E₁ E₂ =
      aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
        (aa + bb) · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natpred) – OptProblem.b E₁ E₂)) :
  aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natpred) –
      OptProblem.b E₁ E₂ =
    aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy +
      1 · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natpred) – OptProblem.b E₁ E₂) := by
  have h1 : aa + bb = 1 := abh
  linarith [h_original_goal, h1]
```

Model Response (DeepSeek-Prover-V2 (CoT))

–––

Detailed Proof and Analysis

Understanding the Problem

The given Lean theorem statement is somewhat peculiar because:
1. The hypotheses `a : xx ∈ univ` and `a_1 : yy ∈ univ` are trivial since everything is in the universe (`univ : Set $E_1$`). Similarly, `abh : aa + bb = 1` is the condition on `aa` and `bb`.
2. The hypothesis `h_original_goal` is exactly the conclusion we need to prove, under certain simplifying assumptions.
3. The goal is to simplify `aa · (OptProblem.$A_1$ $E_2$) xx + bb · (OptProblem.$A_1$ $E_2$) yy + (aa + bb) · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` to `aa · (OptProblem.$A_1$ $E_2$) xx + bb · (OptProblem.$A_1$ $E_2$) yy + 1 · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` using `abh : aa + bb = 1`.

Simplifying the Goal
The expression can be simplified as follows:
`(aa + bb) · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$) = 1 · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` because `aa + bb = 1` by `abh`.

Thus, the goal is to replace `(aa + bb) · . . .` with `1 · . . .`, which is true because `(aa + bb) · X = 1 · X` when `aa + bb = 1`.

The Proof
1. We know `aa + bb = 1` by `abh`.
2. Thus, `(aa + bb) · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$) = 1 · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` because `aa + bb = 1` and scalar multiplication is linear.
3. Substitute this back into the original equation to get the result.

Formal Proof Sketch
1. Replace `(aa + bb) · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` with `1 · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)` using `abh`.
2. The result is exactly the right−hand side of the goal, so we are done.

Abstract Plan
1. Use `abh : aa + bb = 1` to replace `(aa + bb)` with `1` in the scalar multiplication.
2. Simplify the LHS to the RHS:
  − `(aa + bb) · (X) = 1 · (X)`.
3. The rest follows trivially.

−−−

Step−by−Step Abstract Plan

1. **Simplify the Scalar Multiplication**:
  − Since `aa + bb = 1`, we have `(aa + bb) · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$) = 1 · ((OptProblem.$A_2$ $E_1$) (ADMM.$x_2$ $E_1$ F n.natPred) − OptProblem.b $E_1$ $E_2$)`.
  − This is a direct consequence of the linearity of scalar multiplication and the fact that `aa + bb = 1`.

2. **Substitute into the Hypothesis**:
  − The hypothesis `h_original_goal` is exactly the original equation with the LHS rewritten using the simplified scalar multiplication.

3. **Conclude the Goal**:
  − The RHS of `h_original_goal` is exactly the goal, so we are done.

Lean 4 `have` Statements

theorem extracted_formal_statement_1.{u_3, u_2, u_1} {E_1 : Type u_1} {E_2 : Type u_2} {F : Type u_3}
 [inst : NormedAddCommGroup E_1] [inst_1 : InnerProductSpace ℝ E_1] [inst_2 : FiniteDimensional ℝ E_1]
 [inst_3 : NormedAddCommGroup E_2] [inst_4 : InnerProductSpace ℝ E_2] [inst_5 : FiniteDimensional ℝ E_2]

[inst_6 : NormedAddCommGroup F] [inst_7 : InnerProductSpace ℝ F] [inst_8 : FiniteDimensional ℝ F]
{admm : ADMM E_1 E_2 F} {admm_kkt : Existance_of_kkt admm} (n : ℕ+) ⦃xx : E_1⦄ (a : xx ∈ univ) ⦃yy : E_1⦄
(a_1 : yy ∈ univ) ⦃aa bb : ℝ⦄ (a_2 : 0 ≤ aa) (a_3 : 0 ≤ bb) (abh : aa + bb = 1)
(h_original_goal :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
(aa + bb) · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2)) :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) := by
have h_main : (aa + bb) · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) = 1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) := by sorry
have h_final : aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2 = aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + 1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) := by sorry
sorry

Explanation
– `h_main` captures the scalar multiplication simplification using `aa + bb = 1`.
– `h_final` directly substitutes `h_main` into the original hypothesis `h_original_goal` to obtain the goal.
– The final goal is exactly `h_final`, so we use `sorry` to finish the sketch.

Complete Lean 4 Proof

theorem extracted_formal_statement_1.{u_3, u_2, u_1} {E_1 : Type u_1} {E_2 : Type u_2} {F : Type u_3}
[inst : NormedAddCommGroup E_1] [inst_1 : InnerProductSpace ℝ E_1] [inst_2 : FiniteDimensional ℝ E_1]
[inst_3 : NormedAddCommGroup E_2] [inst_4 : InnerProductSpace ℝ E_2] [inst_5 : FiniteDimensional ℝ E_2]
[inst_6 : NormedAddCommGroup F] [inst_7 : InnerProductSpace ℝ F] [inst_8 : FiniteDimensional ℝ F]
[inst_6 : NormedAddCommGroup F] [inst_7 : InnerProductSpace ℝ F] [inst_8 : FiniteDimensional ℝ F]
{admm : ADMM E_1 E_2 F} {admm_kkt : Existance_of_kkt admm} (n : ℕ+) ⦃xx : E_1⦄ (a : xx ∈ univ) ⦃yy : E_1⦄
(a_1 : yy ∈ univ) ⦃aa bb : ℝ⦄ (a_2 : 0 ≤ aa) (a_3 : 0 ≤ bb) (abh : aa + bb = 1)
(h_original_goal :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
(aa + bb) · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2)) :
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy + (OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) −
OptProblem.b E_1 E_2 =
aa · (OptProblem.A_1 E_2) xx + bb · (OptProblem.A_1 E_2) yy +
1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) := by
have h_main : (aa + bb) · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) = 1 · ((OptProblem.A_2 E_1) (ADMM.x_2 E_1 F n.natPred) − OptProblem.b E_1 E_2) := by
have h_1 : aa + bb = 1 := abh
rw [h_1]
<;> simp [add_smul]
<;> abel

```
have h_final : aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (OptProblem.A₂ E₁)
(ADMM.x₂ E₁ F n.natPred) − OptProblem.b E₁ E₂ = aa · (OptProblem.A₁ E₂) xx + bb ·
(OptProblem.A₁ E₂) yy + 1 · ((OptProblem.A₂ E₁) (ADMM.x₂ E₁ F n.natPred) − OptProblem.b E₁
E₂) := by
  calc
    aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (OptProblem.A₂ E₁) (ADMM.x₂
E₁ F n.natPred) − OptProblem.b E₁ E₂
      = aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + (aa + bb) · ((OptProblem.A₂
E₁) (ADMM.x₂ E₁ F n.natPred) − OptProblem.b E₁ E₂) := by
        rw [h_original_goal]
        <;> abel
      _ = aa · (OptProblem.A₁ E₂) xx + bb · (OptProblem.A₁ E₂) yy + 1 · ((OptProblem.A₂ E₁)
(ADMM.x₂ E₁ F n.natPred) − OptProblem.b E₁ E₂) := by
        rw [h_main]
        <;> simp [add_smul]
        <;> abel

exact h_final
```

I Limitations and Future Work

Our current benchmark primarily focuses on a subset of main domains related to machine learning theory. In the future, we aim to expand into more relevant fields. Additionally, we plan to formalize certain aspects of ML theory using Lean, with the goal of encouraging more researchers to explore the use of LLM-based methods for assisting in formal theorem proving.