Scaling up Multi-Turn Off-Policy RL and Multi-Agent Tree Search for LLM Step-Provers

Ran Xin[*†1], **Zeyu Zheng**[*1,2], **Yanchen Nie**[*1,3], **Kun Yuan**[3], **Xia Xiao**[†1]
[1]ByteDance Seed [2]Carnegie Mellon University [3]Peking University
{ran.xin, x.xiaxiao}@bytedance.com
*Equal contribution †Corresponding author
https://bfs-prover.github.io/V2/

Abstract

The integration of Large Language Models (LLMs) with automated theorem proving has shown immense promise, yet is constrained by challenges in scaling up both training-time reinforcement learning (RL) and inference-time compute. This paper introduces `BFS-Prover-V2`, a step-level theorem proving system designed to address this dual scaling problem. We present two primary innovations. The first is a novel multi-turn off-policy RL framework for continually improving the performance of the LLM step-prover at training time. This framework, inspired by the principles of AlphaZero, utilizes a multi-stage expert iteration pipeline featuring adaptive tactic-level data filtering and periodic retraining to surmount the performance plateaus that typically curtail long-term RL in LLM-based agents. The second innovation is a planner-enhanced multi-agent system that scales reasoning capabilities at inference time. This architecture employs a general reasoning model as a high-level planner to iteratively decompose complex theorems into a sequence of simpler subgoals. This hierarchical approach substantially reduces the search space, enabling a team of parallel prover agents to collaborate efficiently by leveraging a shared proof cache. We demonstrate that this dual approach to scaling yields SoTA results on established formal mathematics benchmarks. `BFS-Prover-V2` achieves 95.08% and 41.4% on the miniF2F and ProofNet test sets respectively. While demonstrated in the domain of formal mathematics, the RL and inference techniques presented in this work are of broader interest and may be applied to other domains requiring long-horizon multi-turn reasoning and complex search. Our models and code have been open-sourced at https://github.com/ByteDance-Seed/BFS-Prover-V2.

1 Introduction

Automated Theorem Proving (ATP), a subfield of mathematical logic and automated reasoning, represents one of the foundational ambitions of computer science [3]. The contemporary landscape of formal mathematics is increasingly dominated by interactive theorem provers (ITPs) or proof assistants. These systems, such as Coq, Isabelle, and Lean, require a human user to guide the proof process, but they automate significant deductive tasks and, most importantly, provide a machine-checkable guarantee of correctness [9]. Among these, the Lean4 programming language [25] has emerged as a particularly vibrant ecosystem. A key factor in its success is Mathlib [4], a vast and comprehensive, community-driven library of formalized mathematics. Spanning over a million lines of code, mathlib covers extensive areas of algebra, analysis, topology, and more, providing a rich foundation for both advanced mathematical research and the development of verified systems.

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

The rise of Lean4 has coincided with the explosion in the capabilities of LLMs [26, 8, 33], opening a new frontier in neuro-symbolic AI systems. The goal here is to integrate the intuitive yet powerful generation and search capabilities of LLMs with the absolute logical verification of formal systems. This research direction centers on a key feedback loop: an LLM proposes intuitive proof steps, the Lean compiler provides rigorous verification, and RL [40] uses that verification to continuously improve the LLM's reasoning abilities [55, 51, 28, 11, 15].

1.1 A Duality of Scaling Challenges in LLM Provers and Reasoning Agents

The development of high-performance LLM-based provers, or any other reasoning agents, is contingent upon solving two fundamental and deeply interconnected scaling challenges.

Training-time scaling. This refers to the techniques required to continuously enhance a model's foundational capabilities and tactical intuitions via training. A common and significant obstacle in applying RL to LLMs is the phenomenon of performance plateaus: after an initial phase of rapid improvement, models often stagnate, with their capabilities ceasing to grow despite continued training [24, 41, 58, 59, 10, 33, 51, 52]. Overcoming it requires carefully designed algorithms that can sustain improving, enabling the model to master increasingly harder problems.

Inference-time scaling. This addresses the method of using a trained model to solve theorems. Real-world mathematical problems often require deep reasoning, the formulation of intermediate lemmas, and the exploration of an exponentially large search space of possible tactics. A powerful base model, while necessary, is not sufficient. Without an effective search strategy, even a very competent model can be overwhelmed by the search complexity. The challenge is to design an inference system that allocates computational resources to the most promising avenues of exploration [2, 61, 7, 13, 6].

1.2 Our Contributions

This paper presents `BFS-Prover-V2`, a comprehensive training and inference system for neural theorem proving in Lean4 that introduces novel solutions to the above scaling challenges. The primary contributions of this work are as follows:

Novel RL Scaling Techniques at Training: We develop a distillation-free multi-stage expert-iteration framework [37, 1], a form of off-policy RL, tailored for the domain of formal theorem proving. To sustain learning and overcome performance plateaus, we introduce a suite of specialized techniques within the RL pipeline. These include an adaptive, perplexity-based data filtering strategy at the tactic level, which creates an automated curriculum for the agent, and a periodic retraining mechanism to escape local optima in the model parameter space and increase model scaling potential.

A Planner-Enhanced Multi-Agent Tree Search System at Inference: For inference-time scaling, we introduce a hierarchical reasoning architecture. A general-purpose reasoning model, termed the Planner, iteratively decomposes complex theorems/goals into a sequence of more manageable subgoals. These subgoals are then tackled by a group of parallel prover agents that share a common subgoal cache, dramatically decreasing search complexity of the system and enabling it to solve problems that are intractable for a monolithic prover.

State-of-the-Art Empirical Results: We validate the effectiveness and generalizability of our dual scaling approach on established benchmarks. `BFS-Prover-V2` achieves 95.08% on miniF2F test set, largely surpassing previous step-provers [49, 53] and performing on par with best whole-proof models [32, 22, 43]. On ProofNet test, it achieves 41.4%, setting a new state-of-the-art, showing robust generalization across distributions.

2 The BFS-Prover-V2 System

This section details the two core components of `BFS-Prover-V2`: (i) a training pipeline, grounded in a Markov Decision Process (MDP) [30] and scaled via adaptive filtering and periodic retraining; and (ii) an inference engine, which uses a planner-enhanced multi-agent search for hierarchical reasoning.

We formulate proof search in Lean4 tactic mode as a multi-turn interaction between an LLM agent and the Lean environment, modeled as a MDP. This approach, by design, trains a model that functions as a genuine Lean copilot, suggesting the next tactic step at any point in the proof process [56].

The core training loop of `BFS-Prover-V2` is an expert iteration pipeline, which may be viewed as a variant of the AlphaZero algorithm [1, 37]. This process, illustrated in the inner loop of Fig. 1, includes two major alternating phases. In the **proof generation phase**, the best checkpoint of the LLM prover (the expert) solves a large corpus of mathematical problems using the best-first tree search (BFS) [53]. We formalized approximately 3 million problems [50, 57, 4, 17, 21, 53] to serve as the training ground. Each successful proof constitutes a trajectory of (state, tactic) pairs. The data generated in the first phase is then used in **model refinement phase** to update the LLM model's policy, which then becomes the new "expert" for the next iteration.

2.1 Scaling up training: multi-stage expert iteration

A central challenge in scaling the expert iteration pipeline or RL in general is managing the vast quantity and variable quality of the synthetic data. Naively training on every successful tactic discovered quickly leads to diminishing returns and mode collapse [24, 40, 53]. To sustain improvement over many iterations, we introduce two key algorithmic innovations: an adaptive data filtering strategy and a periodic retraining process. The overall architecture of this pipeline is illustrated in Fig. 1.

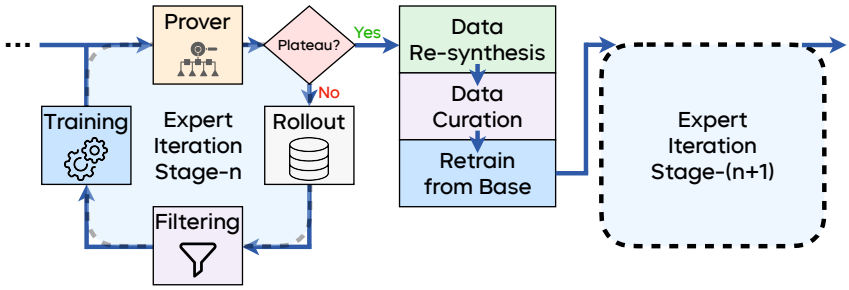

Figure 1: Overview of the training-time scaling up architecture.

Adaptive Tactic Filtering Instead of relying on coarse problem-level filtering [58, 41], we adopt a more fine-grained approach. This strategy is guided by the empirical observation that the perplexity of tactics generated by the LLM follows a roughly Gaussian distribution. The distribution, shown in Fig. 2, can be divided into three distinct regions, each with different implications for learning:

- **The Low-Perplexity Tail:** This region corresponds to tactics for which the model has very high confidence. These are typically simple steps, including which in the training batch offers no new learning signal. It can contribute to overfitting and a reduction in exploratory capacity.

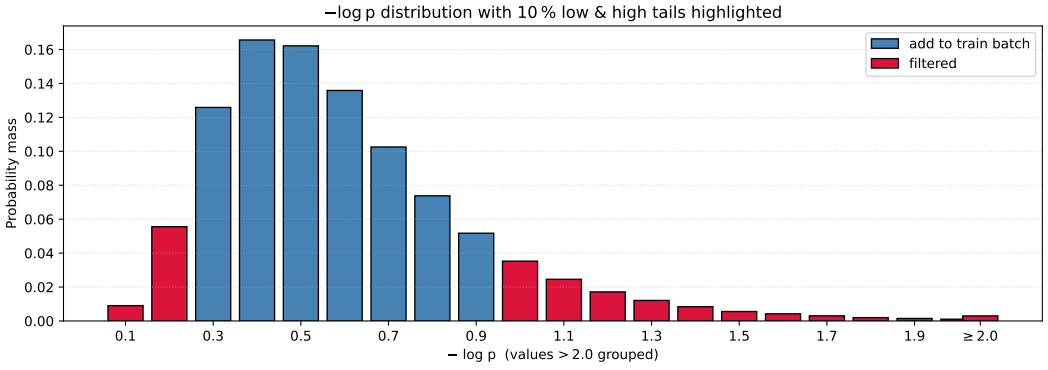

Figure 2: Tactic-Level Data Filtering Based on the Perplexity Distribution.

- **The High-Perplexity Tail:** This region represents tactics that the model finds highly surprising. Case studies reveal that these are often not brilliant reasoning but noisy or suboptimal choices. These data may teach the model to generate overly complex or irrelevant tactics.
- **The Central Distribution:** This region represent steps that are challenging for the model but still within its grasp—its zone of proximal development. By selectively training only on the data from

this central part of the distribution, we ensure that the model is constantly learning at the edge of its capabilities.

This filtering mechanism functions as a fully automated form of curriculum learning. It uses the model's own uncertainty as a dynamic signal of what constitutes valuable training data at its current stage of development. This ensures a smooth and stable evolution of the model's policy distribution.

Periodic Retraining: A "Soft Reset" to Escape Local Optima Even with adaptive filtering, after a few expert iterations, the model still plateaus because it gets trapped in a local optimum. To escape local optima, we introduce a multi-stage expert-iteration process to re-increase the model's entropy and reset its exploratory potential without losing too much competence. The procedure is as follows:

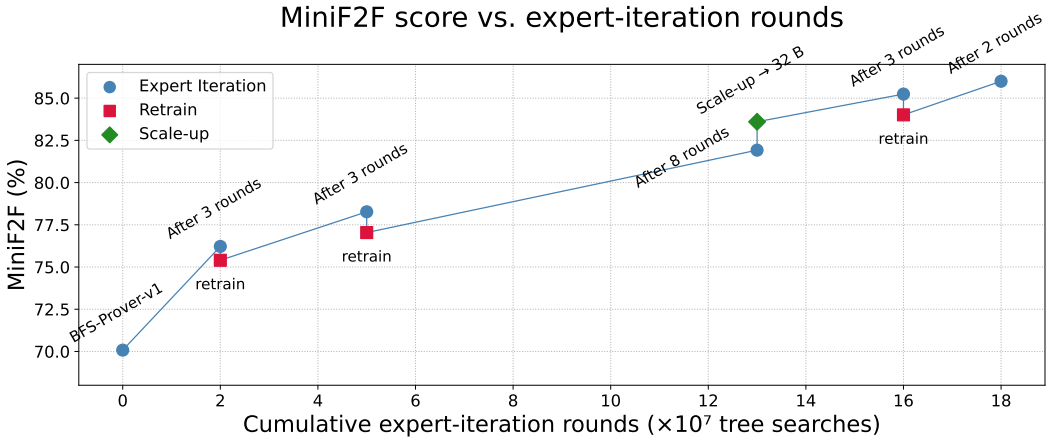

Figure 3: Sustained Performance Improvement through Expert Iteration and Periodic Retraining.

1. **Re-synthesis and De-noise:** The current best prover is used to re-solve the problems in all past iterations to find better proofs. This step effectively uses the expert model to de-noise and improve upon its own past work. The re-synthesized proofs are then filtered as described above.
2. **Retrain from a base Checkpoint:** The existing training data is completely replaced by the new dataset. A fresh model is then initialized and trained from scratch on this refined data.

The resulting model, as illustrated in Fig. 3, initially exhibits a temporary drop in performance. However, when it is reintroduced into the expert iteration loop, its increased capacity for exploration allows it to rapidly recover and then surpass the previous peak.

2.2 Scaling up Inference: Planner-Enhanced Multi-Agent Search

We introduce a hierarchical inference architecture shown in Fig. 4, which mirrors the workflow of a human mathematician, who might first sketch out a high-level idea of a proof by identifying key lemmas, and then fill in detailed deductions. Likewise, the **planner**, a reasoning LLM proposes a high-level plan that includes a series of intermediate subgoals. The **prover**, a specialized LLM trained as a tactic generator, receives one subgoal at a time and uses tree search to find a formal proof.

Planner-Guided Search At start, the planner is queried with the main theorem statement to propose a list of subgoals as Lean `have` statements. The prover system then addresses the subgoals sequentially via tree search. Once a subgoal is proven, it is treated as a known fact which can be used in all subsequent proofs. If the provers fail to find a proof for a subgoal within budget, the planner is re-queried with all subgoals that were successfully proven, in addition to the theorem statement to generate a revised plan that corrects or refines the original proof strategy.

Parallel Provers and Shared Subgoal Cache We deploy multiple parallel prover instances that jointly work on a single subgoal at a time to prevent wasting compute on subgoals that would be rendered invalid if an earlier step fails and triggers a replan. As soon as the first agent finds a proof, the

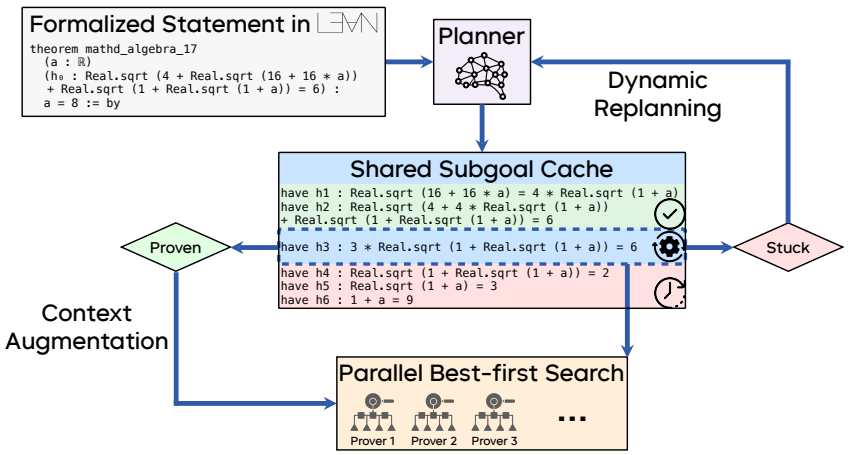

Figure 4: Overview of the planner-enhanced multi-agent tree search architecture.

subgoal cache signals all other provers to terminate their search, preventing redundant computation. The entire group of agents then proceeds to the next subgoal in the sequence.

3 Conclusion

This work's contributions are the design and implementation of a holistic system for scaling LLM-based step-provers. For training, our multi-stage expert iteration pipeline overcomes performance plateaus and enable sustained improvement over an extended period. For inference, we introduced a Planner-Prover paradigm. By using a planner to generate subgoals, we enable the system to tackle complex, multi-step theorems that are intractable for monolithic approaches. The SoTA results on the miniF2F and ProofNet benchmarks provide strong evidence for the efficacy of our approach.

Prover Method	budget	miniF2F-test	miniF2F-valid	ProofNet-test
Step-level provers				
InternLM2.5-StepProver-7B [49]	$256 \times 32 \times 600$	65.9%	69.6%	$\approx 27\%$
Hunyuan-Prover-7B [18]	$600 \times 8 \times 400$	68.4%	-	-
BFS-Prover-V1-7B [53]	$2048 \times 2 \times 600$	70.8%	-	-
	accumulative	73.0%	-	-
MPS-Prover-7B† [19]	$64 \times 4 \times 800 \times 8$	72.54%	-	-
	accumulative	75.8%	-	-
BFS-Prover-V2-7B (this work)	accumulative	82.4%	-	-
BFS-Prover-V2-32B (this work)	accumulative	86.1%	85.5%	41.4%
w/ Planner	accumulative	95.1%	95.5%	-
Whole-proof provers				
DeepSeek-Prover-V2-671B [32]	8192	88.9%	90.6%	37.1%
Kimina-Prover-72B† [43]	1024	87.7%	-	-
w/ TTRL search	accumulative	92.2%	-	-
Goedel-Prover-32B† [22]	8192	92.2%	-	-
w/ Self-correction	1024	92.6%	-	-
Delta-Prover† [61]	accumulative	95.9%	-	-
Seed-Prover† [7]	accumulative	99.6%	-	-

Table 1: Comparison between BFS-Prover-V2 and other theorem provers. † denotes concurrent work.

Acknowledgements

We would like to thank Kai Shen from ByteDance Seed for his insightful discussions throughout this project.

References

[1] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

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

[3] Wolfgang Bibel, Steffen Hölldobler, and Gerd Neugebauer. *Deduction: automated logic*. Academic Press London, 1993.

[4] Mark Blokpoel. mathlib: A scala package for readable, verifiable and sustainable simulations of formal theory. *Journal of Open Source Software*, 9(99):6049, 2024.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, Hui Xue, and Fan Yang. Reviving dsp for advanced theorem proving in the era of reasoning models. *arXiv preprint arXiv:2506.11487*, 2025.

[7] 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.

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

[9] Herman Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 34(1):3–25, 2009.

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

[11] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

[12] Yuhang He, Jihai Zhang, Jianzhu Bao, Fangquan Lin, Cheng Yang, Bing Qin, Ruifeng Xu, and Wotao Yin. Bc-prover: Backward chaining prover for formal theorem proving. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3059–3077, 2024.

[13] 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.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[15] Guillaume Lample, Timothée Lacroix, et al. Hypertree proof search for neural theorem proving. *Advances in Neural Information Processing Systems*, 35:26337–26349, 2022.

[16] Chenyi Li, Ziyu Wang, Wanyi He, Yuxuan Wu, Shengyang Xu, and Zaiwen Wen. Formalization of convergence rates of four first-order algorithms for convex optimization: C. li et al. *Journal of Automated Reasoning*, 69(4):28, 2025.

[17] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

[18] Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.

[19] Zhenwen Liang, Linfeng Song, Yang Li, Tao Yang, Feng Zhang, Haitao Mi, and Dong Yu. Mps-prover: Advancing stepwise theorem proving by multi-perspective search and data curation. *arXiv preprint arXiv:2505.10962*, 2025.

[20] Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.

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

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

[23] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[24] Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025.

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

[26] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[27] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[28] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, et al. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

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

[30] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[31] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[32] 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.

[33] ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

[34] Ziju Shen, Naohao Huang, Fanyi Yang, Yutong Wang, Guoxiong Gao, Tianyi Xu, Jiedong Jiang, Wanyi He, Pu Yang, Mengzhou Sun, et al. Real-prover: Retrieval augmented lean prover for mathematical reasoning. *arXiv preprint arXiv:2505.20613*, 2025.

[35] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[36] David Silver, Thomas Hubert, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[37] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[38] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1, 2025.

[39] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Towards large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

[40] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

[41] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[42] Ye Tian, Baolin Peng, et al. Toward self-improvement of llms via imagination, searching, and criticizing. *Advances in Neural Information Processing Systems*, 2023.

[43] 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.

[44] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*, 2023.

[45] Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and Tong Zhang. Ma-lot: Model-collaboration lean-based long chain-of-thought reasoning enhances formal theorem proving. *arXiv preprint arXiv:2503.03205*, 2025.

[46] Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*, 2024.

[47] Sean Welleck and Rahul Saha. Llmstep: Llm proofstep suggestions in lean. *arXiv preprint arXiv:2310.18457*, 2023.

[48] Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368, 2022.

[49] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

[50] Zijian Wu, Jiayu Wang, Dahua Lin, and Kai Chen. Lean-github: Compiling github lean repositories for a versatile lean prover. *arXiv preprint arXiv:2407.17227*, 2024.

[51] Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024.

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

[53] 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.

[54] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[55] 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.

[56] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[57] Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*, 2024.

[58] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[59] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

[60] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

[61] Yichi Zhou, Jianqiu Zhao, Yongxin Zhang, Bohan Wang, Siran Wang, Luoxin Chen, Jiahui Wang, Haowei Chen, Allan Jie, Xinbo Zhang, et al. Solving formal math problems by decomposition and iterative reflection. *arXiv preprint arXiv:2507.15225*, 2025.

A Practical Implementation and Benchmark Results

We now present practical implementation of the `BFS-Prover-V2` system and its benchmark results.

Model and Data: The LLM prover agent is built upon the Qwen2.5-Math-7B and Qwen2.5-32B models [54], which serve as the base for our policy optimization. The multi-stage expert iteration process was initialized with the checkpoint from `BFS-Prover-V1` [53]. To construct a large-scale training corpus, we autoformalized the NuminaMath-CoT and NuminaMath-1.5 datasets [17] using carefully designed prompts applied to general-purpose models, augmented with Lean4 compiler feedback. Combined with data provided by Goedel-Prover [21], this process produced approximately 3 million formal statements. Prompts used for autoformalization can be found in Section D.1. All experiments are conducted in Lean v4.10.0 with LeanDojo [56].

Training setup: We refine the policy LLM after each expert iteration round using one of two Supervised Fine-Tuning (SFT) strategies, chosen based on the outcome of the round. For rounds with a manageable data yield, we perform a continuous finetune from the current best checkpoint for a single epoch, using a conservative cosine learning rate decay from 5×10^{-6} and decaying to 1×10^{-7}. A more comprehensive retrain from the base model is triggered under two conditions: either if the round produces a very large volume of new data, or if model performance has stagnated. In the case of a performance plateau, this retraining is combined with an aggressive data curation step to create a new, refined dataset designed to break the local; see Section 2.1. This retraining process is conducted for 3 epochs with a higher learning rate decaying from 2×10^{-5} and decaying to 1×10^{-6}. Both strategies utilize a global batch size of 1024.

Inference configuration: Our inference process combines a low-level Prover with a high-level Planner, as detailed in Section 2.2. The Prover agents utilize a Best-First Search (BFS) algorithm, with an implementation that follows `BFS-Prover-V1` [53], where we set the sampling temperature to 1.3, the expansion width to 3, and a length normalization factor of 2.0. For the high-level strategic Planner, we employ Gemini2.5-pro, while other general-purpose reasoning models can achieve comparable performance if properly prompted. Prompts used for Planner can be found in Section D.2.

Benchmark results: We evaluated `BFS-Prover-V2` on two key benchmarks: miniF2F, a test of high-school competition math, and ProofNet, which challenges reasoning over a large undergraduate-level library. Our system sets a new state of the art for LLM step-provers, achieving 95.08% on the miniF2F test set (95.49% on validation) and 41.4% on the ProofNet test set. The near-saturation performance on miniF2F validates our iterative RL pipeline's ability to master a problem distribution. More importantly, the strong ProofNet result demonstrates successful generalization from the system's training corpus, which consists mainly of high-school competition problems, to the more complex, library-dependent undergraduate problems. See detailed comparison with other LLM provers in Table 1.

B Case Studies

B.1 Proof Conciseness and Tactic Proficiency

A primary advantage of our step-level proof approach over the whole-proof paradigm is a dramatic reduction in proof length, which arises from the interactive nature of our method. By engaging with the Lean environment step by step, our model captures and leverages fine-grained tactic state information. This iterative feedback loop significantly improves its ability to employ powerful, high-level tactics such as `simp`, `linarith`, and `nlinarith`, enabling it to solve complex goals in a single step rather than through lengthy, explicit derivations.

To illustrate this contrast, we present a comparative analysis of proofs for two problems, `aime_1984_p7` and `amc12_2000_p1`, generated by `BFS-Prover-V2` and `DeepSeek-Prover-V2`.

`aime_1984_p7`

For the problem `aime_1984_p7`, `BFS-Prover-V2` produces a remarkably concise proof, where a single tactic `simp [h₁, h₀]` effectively replaces over 2000 lines that appear in numerous `have` blocks within `DeepSeek-Prover-V2`'s proof.

BFS-Prover-V2 (step-level proof):

```
theorem aime_1984_p7
  (f : ℤ → ℤ)
  (h₀ : ∀ n, 1000 ≤ n → f n = n - 3)
  (h₁ : ∀ n, n < 1000 → f n = f (f (n + 5))) :
  f 84 = 997 := by
  simp [h₁, h₀]
```

DeepSeek-Prover-V2 (whole-proof):

```
theorem aime_1984_p7 (f : ℤ → ℤ) (h₀ : ∀ n, 1000 ≤ n → f n = n -
    3)
    (h₁ : ∀ n, n < 1000 → f n = f (f (n + 5))) : f 84 = 997 :=
   by
  have h₂ : f 1004 = 1001 := by
    have h₂₁ : f 1004 = 1004 - 3 := by
      apply h₀
      <;> norm_num
    rw [h₂₁]
    <;> norm_num

  have h₃ : f 1003 = 1000 := by
    have h₃₁ : f 1003 = 1003 - 3 := by
      apply h₀
      <;> norm_num
    rw [h₃₁]
    <;> norm_num

  have h₄ : f 1002 = 999 := by
    have h₄₁ : f 1002 = 1002 - 3 := by
      apply h₀
      <;> norm_num
    rw [h₄₁]
    <;> norm_num

  -- ... (approximately 2000 lines of similar calculations
    omitted) ...
```

```
  have h₁₉₃ : f 89 = 998 := by
    have h₁₉₃₁ : f 89 = f (f (89 + 5)) := by
      apply h₁
      <;> norm_num
    rw [h₁₉₃₁]
    have h₁₉₃₂ : f (89 + 5) = f 94 := by norm_num
    rw [h₁₉₃₂]
    rw [h₁₉₂]
    <;> norm_num
    <;> linarith

  have h₁₉₄ : f 84 = 997 := by
    have h₁₉₄₁ : f 84 = f (f (84 + 5)) := by
      apply h₁
      <;> norm_num
    rw [h₁₉₄₁]
    have h₁₉₄₂ : f (84 + 5) = f 89 := by norm_num
    rw [h₁₉₄₂]
    rw [h₁₉₃]
    <;> norm_num
    <;> linarith

  rw [h₁₉₄]
  <;> norm_num
```

amc12_2000_p1

For the problem amc12_2000_p1, BFS-Prover-V2 finishes the proof with a concise application of nlinarith, replacing the roughly 80 lines of explicit case analysis over the unknowns i, m, and o required in DeepSeek-Prover-V2's proof.

BFS-Prover-V2 (step-level proof):

```
theorem amc12_2000_p1
  (i m o : ℕ)
  (h₀ : i ≠ m ∧ m ≠ o ∧ o ≠ i)
  (h₁ : i*m*o = 2001) :
  i+m+o ≤ 671 := by
  obtain ⟨hi, hm, h₂⟩:= h₀
  obtain _ | _ | _ | i := i <;> obtain _ | _ | _ | m := m <;>
    obtain _ | _ | _ | o := o <;>try omega
  any_goals
    ring_nf at h₁ ⊢; nlinarith;
  ring_nf at * <;> try omega
  nlinarith [mul_nonneg (by positivity : 0 ≤ i) (by positivity :
     0 ≤ o), mul_nonneg (by positivity : 0 ≤ m) (by positivity :
    0 ≤ i), mul_nonneg (by positivity : 0 ≤ o) (by positivity : 0
     ≤ m)]
```

DeepSeek-Prover-V2 (whole-proof):

```
theorem amc12_2000_p1 (i m o : ℕ) (h₀ : i ≠ m ∧ m ≠ o ∧ o ≠ i) (
    h₁ : i * m * o = 2001) :
    i + m + o ≤ 671 := by
```

```
  -- ... (approximately 80 lines of properties related to
  unknowns i, m, and o omitted) ...

have h_main : i + m + o ≤ 671 := by
  have h₈ : i = 1 ∨ i = 3 ∨ i = 23 ∨ i = 29 ∨ i = 69 ∨ i = 87
  ∨ i = 667 ∨ i = 2001 := by
    have h₈₁ : i ∣ 2001 := h₂
    have h₈₂ : i ≤ 2001 := h₅
    have h₈₃ : i > 0 := h_pos_i
    have h₈₄ : i = 1 ∨ i = 3 ∨ i = 23 ∨ i = 29 ∨ i = 69 ∨ i =
  87 ∨ i = 667 ∨ i = 2001 := by
      have h₈₅ : i ∣ 2001 := h₈₁
      have h₈₆ : i ≤ 2001 := h₈₂
      interval_cases i <;> norm_num at h₈₅ ⊢ <;> omega
    exact h₈₄
  have h₉ : m = 1 ∨ m = 3 ∨ m = 23 ∨ m = 29 ∨ m = 69 ∨ m = 87
  ∨ m = 667 ∨ m = 2001 := by
    have h₉₁ : m ∣ 2001 := h₃
    have h₉₂ : m ≤ 2001 := h₆
    have h₉₃ : m > 0 := h_pos_m
    have h₉₄ : m = 1 ∨ m = 3 ∨ m = 23 ∨ m = 29 ∨ m = 69 ∨ m =
  87 ∨ m = 667 ∨ m = 2001 := by
      have h₉₅ : m ∣ 2001 := h₉₁
      have h₉₆ : m ≤ 2001 := h₉₂
      interval_cases m <;> norm_num at h₉₅ ⊢ <;> omega
    exact h₉₄
  have h₁₀ : o = 1 ∨ o = 3 ∨ o = 23 ∨ o = 29 ∨ o = 69 ∨ o = 87
   ∨ o = 667 ∨ o = 2001 := by
    have h₁₀₁ : o ∣ 2001 := h₄
    have h₁₀₂ : o ≤ 2001 := h₇
    have h₁₀₃ : o > 0 := h_pos_o
    have h₁₀₄ : o = 1 ∨ o = 3 ∨ o = 23 ∨ o = 29 ∨ o = 69 ∨ o =
   87 ∨ o = 667 ∨ o = 2001 := by
      have h₁₀₅ : o ∣ 2001 := h₁₀₁
      have h₁₀₆ : o ≤ 2001 := h₁₀₂
      interval_cases o <;> norm_num at h₁₀₅ ⊢ <;> omega
    exact h₁₀₄
  rcases h₈ with (rfl | rfl | rfl | rfl | rfl | rfl | rfl |
  rfl) <;>
  rcases h₉ with (rfl | rfl | rfl | rfl | rfl | rfl | rfl |
  rfl) <;>
  rcases h₁₀ with (rfl | rfl | rfl | rfl | rfl | rfl | rfl |
  rfl) <;>
  norm_num [mul_assoc, mul_comm, mul_left_comm] at h₁ h₀ ⊢ <;>
  (try omega) <;>
  (try
    {
      norm_num at h₀ ⊢ <;>
      (try omega) <;>
      (try
        {
          ring_nf at h₁ ⊢ <;>
          omega
        })
    }) <;>
  (try
    {
      norm_num at h₀ ⊢ <;>
      (try omega) <;>
      (try
        {
          ring_nf at h₁ ⊢ <;>
```

```
            omega
          })
      }) <;>
    (try
      {
        norm_num at h₀ ⊢ <;>
        (try omega) <;>
        (try
          {
            ring_nf at h₁ ⊢ <;>
            omega
          })
      }) <;>
    (try
      {
        norm_num at h₀ ⊢ <;>
        (try omega) <;>
        (try
          {
            ring_nf at h₁ ⊢ <;>
            omega
          })
      })
    <;>
    (try omega)
    <;>
    (try
      {
        ring_nf at h₁ ⊢ <;>
        omega
      })

  exact h_main
```

B.2 Novel Proof Strategies

Another significant advantage of our step-level proof approach is its ability to discover novel proof strategies that whole-proof or human-proof methods typically would not consider. By exploring the proof space progressively, our system can identify non-obvious connections and construct solutions that are both elegant and insightful.

We illustrate this capability by examining the problems `imo_1963_p5` and `algebra_amgm_sum1toneqn_prod1tonleq1`, each of which highlights a distinct advantage of our approach.

`imo_1963_p5`

For the problem `imo_1963_p5`, our model, DeepSeek-Prover-V2, and Compfiles dataset provide step-level proof, whole-proof, and human-proof versions, respectively. Notably, both whole-proof and human-proof approaches employ similar strategies: multiplying both sides of the equation by $2 \cdot \sin(\pi/7)$, then applying sum-to-product trigonometric identities for simplification. In contrast, `BFS-Prover-V2` develops an entirely different approach: first transforming the left side of the equation into a polynomial in $\cos(\pi/7)$ using double and triple angle formulas, then proving that $\cos(\pi/7)$ satisfies the corresponding polynomial equation.

BFS-Prover-V2 (step-level proof):

```
theorem imo_1963_p5 :
  Real.cos (π / 7) - Real.cos (2 * π / 7) + Real.cos (3 * π / 7)
    = 1 / 2 := by
  have x : Real.pi / 7 = Real.pi / 7 * 1 := by ring
  have h : 3 * Real.pi / 7 = Real.pi - 4 * Real.pi / 7 := by
    ring
  rw [h, cos_sub] <;> norm_num
  have h2 := cos_two_mul (Real.pi / 7)
  have h3 := cos_three_mul (π / 7)
  rw [show 4 * Real.pi / 7 = Real.pi - 3 * Real.pi / 7 by ring,
    cos_sub]
  simp [h2, h3, cos_two_mul, sin_pi, cos_pi]
  ring_nf at h2 h3 ⊢
  norm_num [h2, h3, cos_pi_div_two]
  ring_nf
    <;> have h4 := cos_pi
    <;> simp [h4]
  ring_nf at * <;> norm_num
  rw [← sub_eq_zero]
  nth_rewrite 1 [← sub_eq_zero]
  ring_nf
  apply eq_of_sub_eq_zero
  let y := cos (Real.pi * (1 / 7))
  have:= cos_three_mul (Real.pi * (1 / 7))
  ring_nf at *
  apply eq_of_sub_eq_zero
  clear this h3 h2
  apply eq_of_sub_eq_zero
  have := cos_three_mul (Real.pi * (1 / 7))
  field_simp [mul_assoc] at *
  on_goal 1 => ring
  replace : Real.pi * (1 / 7 : ℝ) = Real.pi / 7 := by ring
  try rw [this]; norm_num
  have h5 := cos_three_mul (Real.pi / 7)
  have : 3 * (Real.pi / 7) = Real.pi - 4 * (Real.pi / 7) := by
    ring
  simp [this, cos_pi] at h5
  let z := cos (Real.pi / 7)
  rcases lt_trichotomy 0 z with hz | hz | hz
  any_goals simp_all [show cos (4 * (Real.pi / 7)) = cos (2 * (2
    * (Real.pi / 7))) by ring,
   cos_two_mul]
  any_goals nlinarith [cos_sq_add_sin_sq (Real.pi / 7), Real.
    sin_pi_div_two_sub, pow_two_nonneg (cos (Real.pi / 7) - 1),
    pow_two_nonneg (cos (Real.pi / 7) + 1)]
  contrapose hz
  refine not_lt.2 ?_
  apply cos_nonneg_of_mem_Icc <;> constructor <;> linarith [
    pi_pos]
```

DeepSeek-Prover-V2 (whole-proof):

```
theorem imo_1963_p5 : Real.cos (Real.pi / 7) - Real.cos (2 *
    Real.pi / 7) + Real.cos (3 * Real.pi / 7) = 1 / 2 := by
  have h1 : 2 * Real.sin (Real.pi / 7) * (Real.cos (Real.pi / 7)
    - Real.cos (2 * Real.pi / 7) + Real.cos (3 * Real.pi / 7)) =
    Real.sin (Real.pi / 7) := by
    have h1₁ : 2 * Real.sin (Real.pi / 7) * Real.cos (Real.pi /
    7) = Real.sin (2 * Real.pi / 7) := by
```

```
      have h1₁₁ : Real.sin (2 * Real.pi / 7) = 2 * Real.sin (
    Real.pi / 7) * Real.cos (Real.pi / 7) := by
        have h1₁₂ : Real.sin (2 * Real.pi / 7) = Real.sin (2 * (
    Real.pi / 7)) := by ring
        rw [h1₁₂]
        have h1₁₃ : Real.sin (2 * (Real.pi / 7)) = 2 * Real.sin
    (Real.pi / 7) * Real.cos (Real.pi / 7) := by
          rw [Real.sin_two_mul]
          <;> ring
        rw [h1₁₃]
        <;> ring
      linarith
    have h1₂ : 2 * Real.sin (Real.pi / 7) * Real.cos (2 * Real.
    pi / 7) = Real.sin (3 * Real.pi / 7) - Real.sin (Real.pi / 7)
     := by

  -- ... (approximately 20 lines of calculations omitted) ...

    have h1₃ : 2 * Real.sin (Real.pi / 7) * Real.cos (3 * Real.
    pi / 7) = Real.sin (4 * Real.pi / 7) - Real.sin (2 * Real.pi
    / 7) := by

  -- ... (approximately 20 lines of similar calculations omitted
    ) ...

    have h1₄ : Real.sin (4 * Real.pi / 7) = Real.sin (3 * Real.
    pi / 7) := by

  -- ... (approximately 20 lines of similar calculations omitted
    ) ...

  have h2 : Real.sin (Real.pi / 7) > 0 := by
    apply Real.sin_pos_of_pos_of_lt_pi
    · linarith [Real.pi_pos, Real.pi_gt_three]
    · linarith [Real.pi_pos, Real.pi_gt_three]

  have h3 : Real.cos (Real.pi / 7) - Real.cos (2 * Real.pi / 7)
    + Real.cos (3 * Real.pi / 7) = 1 / 2 := by
    have h3₁ : 2 * Real.sin (Real.pi / 7) > 0 := by linarith
    have h3₂ : Real.cos (Real.pi / 7) - Real.cos (2 * Real.pi /
    7) + Real.cos (3 * Real.pi / 7) = 1 / 2 := by
      apply mul_left_cancel₀ (show (2 * Real.sin (Real.pi / 7) :
     ℝ) ≠ 0 by linarith)
      nlinarith [Real.sin_le_one (Real.pi / 7), Real.sin_le_one
    (2 * Real.pi / 7), Real.sin_le_one (3 * Real.pi / 7),
        Real.sin_le_one (4 * Real.pi / 7), Real.sin_le_one (Real
    .pi / 7)]
    exact h3₂

  apply h3
```

Compfiles dataset (human-proof):

```
theorem imo1963_p5 :
    Real.cos (π/7) - Real.cos (2*π/7) + Real.cos (3*π/7) = 1/2
    := by
  rw [show (2*π/7) = π - (5*π/7) by linarith]
  rw [Real.cos_pi_sub]
  simp only [sub_neg_eq_add]
  have h : 2 * Real.sin (π / 7) ≠ 0 := by
```

```
    simp only [ne_eq, mul_eq_zero, OfNat.ofNat_ne_zero, false_or
    ]
    apply ne_of_gt
    apply Real.sin_pos_of_pos_of_lt_pi
    simp only [Nat.ofNat_pos, div_pos_iff_of_pos_right, Real.
    pi_pos]
    trans 1
    · rw [div_lt_one (by linarith only)]
      linarith only [Real.pi_le_four]
    · linarith only [Real.pi_gt_three]
  apply (mul_right_inj' h).mp
  rw [left_distrib, left_distrib]
  have prod_sum : ∀ (x y : ℝ),
      2 * Real.sin x * Real.cos y = Real.sin (x + y) - Real.sin
    (y - x) := by
    intro x y
    rw [Real.sin_add, Real.sin_sub]
    linarith only
  rw [prod_sum, prod_sum, prod_sum]
  rw [show (π / 7 + π / 7)     = 2 * π / 7 by linarith only]
  rw [show (π / 7 - π / 7)     = 0         by linarith only]
  rw [show (π / 7 + 5 * π / 7) = 6 * π / 7 by linarith only]
  rw [show (5 * π / 7 - π / 7) = 4 * π / 7 by linarith only]
  rw [show (π / 7 + 3 * π / 7) = 4 * π / 7 by linarith only]
  rw [show (3 * π / 7 - π / 7) = 2 * π / 7 by linarith only]
  rw [Real.sin_zero]
  ring_nf
  rw [← Real.sin_pi_sub]
  rw [show (π - π * (6 / 7)) = π / 7 by linarith]
  congr
  linarith
```

algebra_amgm_sum1toneqn_prod1tonleq1

For the problem algebra_amgm_sum1toneqn_prod1tonleq1, the whole-proof model DeepSeek-Prover-V2 adopts a standard, first-principles approach: it proceeds by manually handling cases ($n = 0$, some $a_i = 0$, all $a_i > 0$), and then takes the logarithm of the product and then applies the well-known inequality $\ln(x) \leq x - 1$ to each term, resulting in a verbose proof. In contrast, BFS-Prover-V2 recognizes the problem as a special case of the Arithmetic Mean-Geometric Mean (AM-GM) inequality. It directly invokes the corresponding theorem from Mathlib, Real.geom_mean_le_arith_mean, demonstrating an ability to leverage high-level library theorems for a more insightful and efficient proof.

BFS-Prover-V2 (step-level proof):

```
theorem algebra_amgm_sum1toneqn_prod1tonleq1
  (a : ℕ → NNReal)
  (n : ℕ)
  (h₀ : ∑ x in Finset.range n, a x = n) :
  ∏ x in Finset.range n, a x ≤ 1 := by
  have g := h₀
  revert h₀
  intro amgm
  let S := Finset.range n
  by_cases h1 : n = 0
  simp[h1]
  have hn  : 0 < n := by omega
```

```
  let μ := (fun (x : ℕ) => (a x : ℝ))
  let w : ℕ → ℝ := fun _ => 1
  have w_nonneg : ∀ i, 0 ≤ w i
  simp [w, zero_le_one]
  have w_pos  : 0 < ∏ i in Finset.range n, w i
  all_goals aesop (add simp w)
  have w_nonneg : ∀ x ∈ S, 0 ≤ w x
  any_goals simp [w]
  have w_pos  : 0 < ∏ x ∈ S, w x
  simp [w]
  have amgm_real  : (∏ x ∈ S, μ x ^ (w x : ℝ)) ^ ((∑ x ∈ S, (w x
    : ℝ))⁻¹) ≤ (∑ x ∈ S, (w x * μ x : ℝ)) / (∑ x ∈ S, (w x : ℝ))
  apply Real.geom_mean_le_arith_mean
  exact w_nonneg
  simp [w, S, hn]
  field_simp[μ]
  simp [μ, w, S] at amgm_real
  norm_cast at amgm_real
  simp[ amgm] at amgm_real
  by_cases H : ∏ a_1 ∈ Finset.range n, a a_1 = 0
  simp [H]
  simp [← NNReal.coe_le_coe, amgm] at amgm_real
  rw  [div_self] at amgm_real
  apply le_of_not_lt
  all_goals aesop (add simp NNReal)
  rw[←NNReal.coe_one] at amgm_real
  norm_cast at amgm_real
  contrapose! amgm_real
  erw [ Real.one_lt_rpow_iff]
  left
  all_goals aesop (add simp [Finset.range])
```

DeepSeek-Prover-V2 (whole-proof):

```
theorem algebra_amgm_sum1toneqn_prod1tonleq1 (a : ℕ → NNReal) (n
    : ℕ)
    (h₀ : (∑ x in Finset.range n, a x) = n) : (∏ x in Finset.
   range n, a x) ≤ 1 := by
  have h₁ : (∏ x in Finset.range n, (a x : ℝ)) ≤ 1 := by
    have h₂ : (∑ x in Finset.range n, (a x : ℝ)) = n := by
      exact_mod_cast h₀
    by_cases h₃ : n = 0
      simp [h₃]
      have h₄ : 0 < n := Nat.pos_of_ne_zero h₃
      by_cases h₅ : ∃ i, i ∈ Finset.range n ∧ (a i : ℝ) = 0
        obtain ⟨i, hi, h₆⟩ := h₅
        have h₇ : (∏ x in Finset.range n, (a x : ℝ)) = 0 := by
          have h₈ : i ∈ Finset.range n := hi
          have h₉ : (a i : ℝ) = 0 := h₆
          have h₁₀ : (∏ x in Finset.range n, (a x : ℝ)) = 0 :=
   by
            calc
              (∏ x in Finset.range n, (a x : ℝ)) = ∏ x in Finset
   .range n, (a x : ℝ) := rfl
              _ = 0 := by
                apply Finset.prod_eq_zero h₈
                simp [h₉]
          exact h₁₀
        linarith
        have h₅' : ∀ i, i ∈ Finset.range n → (a i : ℝ) > 0 := by
          intro i hi
```

```
        have h₆ : ¬∃ i, i ∈ Finset.range n ∧ (a i : ℝ) = 0 :=
  h₅
        have h₇ : (a i : ℝ) ≠ 0 := by
          intro h
          have h₈ : ∃ i, i ∈ Finset.range n ∧ (a i : ℝ) = 0 :=
   ⟨i, hi, by simp_all⟩
          contradiction
        have h₈ : (a i : ℝ) > 0 := by
          have h₉ : (a i : ℝ) ≥ 0 := by exact_mod_cast (a i).
  prop
          contrapose! h₇
          linarith
        exact h₈
      have h₆ : (∏ x in Finset.range n, (a x : ℝ)) ≤ 1 := by
        have h₇ : ∑ x in Finset.range n, Real.log ((a x : ℝ))
  ≤ 0 := by
          have h₈ : ∑ x in Finset.range n, Real.log ((a x : ℝ)
  ) ≤ ∑ x in Finset.range n, ((a x : ℝ) - 1) := by
            apply Finset.sum_le_sum
            intro i hi
            have h₉ : (a i : ℝ) > 0 := h₅' i hi
            have h₁₀ : Real.log ((a i : ℝ)) ≤ (a i : ℝ) - 1 :=
   by
              have h₁₁ : Real.log ((a i : ℝ)) ≤ (a i : ℝ) - 1
  := by
                linarith [Real.log_le_sub_one_of_pos h₉]
              exact h₁₁
            exact h₁₀

  -- ... (approximately 40 lines of calculations omitted) ...

  have h₂ : (∏ x in Finset.range n, a x) ≤ 1 := by
    have h₃ : (∏ x in Finset.range n, a x : ℝ) ≤ 1 := by
      exact h₁
    have h₄ : (∏ x in Finset.range n, a x : ℝ) = (∏ x in Finset.
    range n, a x : ℝ) := rfl
    have h₅ : (∏ x in Finset.range n, a x : ℝ) = (∏ x in Finset.
    range n, (a x : ℝ)) := by simp
    have h₆ : (∏ x in Finset.range n, a x : ℝ) ≤ 1 := by simpa [
    h₅] using h₁
    have h₇ : (∏ x in Finset.range n, a x : NNReal) ≤ 1 := by
      norm_cast at h₆ ⊢
      <;> simp_all [Finset.prod_range_succ]
      <;> norm_num
      <;> linarith
    simpa using h₇
  exact h₂
```

Despite the advantages of step-level proof in significantly reducing proof length and discovering novel proof strategies, step-level proof has one notable limitation: poor readability. The interactive nature of step-level proof generation often results in proofs that are more challenging for humans to follow and understand compared to the more structured and explanatory whole-proof approaches. This trade-off between conciseness and readability represents a crucial consideration when evaluating the practical utility of different proof generation paradigms.

C Illustration of Planner-Prover Paradigm with an IMO Problem

To demonstrate the effectiveness of our Planner-Prover paradigm, we present an analysis of the solution process for a challenging IMO problem: `imo_1969_p2`.

In the following proof, the statements `h_coeffs_polar`, `h_y_rewritten_with_polar`, and `h_y_collapsed_to_single_cos` represent the dynamic replanning phase, while all other `have` statements belong to the initial planning phase. Unlike in conventional whole-proof methods, `have` statements in our framework are presented without the `:= by` clause. This example highlights the crucial role of dynamic replanning in our system. Without dynamic replanning, the prover gets stuck at `h_y_is_sinusoid`, failing to complete the proof even after 7,200 attempts. With dynamic replanning, however, the system successfully completes the proof in just 800 attempts. The dynamic replanning process breaks down complex steps into smaller, more manageable subgoals, which enables the prover to bypass critical bottlenecks more efficiently.

`imo_1969_p2` **- Part 1**

```
theorem imo_1969_p2
  (m n : ℝ)
  (k : ℕ)
  (a : ℕ → ℝ)
  (y : ℝ → ℝ)
  (h₀ : 0 < k)
  (h₁ : ∀ x, y x = ∑ i in Finset.range k, ((Real.cos (a i + x))
   / (2^i)))
  (h₂ : y m = 0)
  (h₃ : y n = 0) : ∃ t : ℤ, m - n = t * Real.pi := by
  have h_cos_add : ∀ i x, Real.cos (a i + x) = Real.cos (a i) *
   Real.cos x - Real.sin (a i) * Real.sin x
  simp [cos_add, add_right_inj]

  have h_y_sum_expanded : ∀ (x : ℝ), y x = ∑ i in (Finset.range
   k : Finset ℕ), (Real.cos (a i) * Real.cos x - Real.sin (a i)
   * Real.sin x) / ((2 : ℕ) ^ i : ℝ)
  simp [h₁, h_cos_add ]

  have h_y_sum_split : ∀ (x : ℝ), y x = (∑ i in (Finset.range k
   : Finset ℕ), Real.cos (a i) * Real.cos x / ((2 : ℕ) ^ i : ℝ))
    - (∑ i in (Finset.range k : Finset ℕ), Real.sin (a i) * Real
   .sin x / ((2 : ℕ) ^ i : ℝ))
  intro z <;> simp_rw [h_y_sum_expanded]
  simp [sub_div, Finset.sum_sub_distrib]

  have h_y_expand : ∀ (x : ℝ), y x = (∑ i in (Finset.range k :
   Finset ℕ), Real.cos (a i) / ((2 : ℕ) ^ i : ℝ)) * Real.cos x -
    (∑ i in (Finset.range k : Finset ℕ), Real.sin (a i) / ((2 :
   ℕ) ^ i : ℝ)) * Real.sin x
  intro x_exp
  simp only [Finset.sum_mul, h_y_sum_split]
  congr <;> symm <;> field_simp <;> ring

  have h_k_ge_one : 1 ≤ k
  apply Nat.succ_le_of_lt    <;> exact h₀

  have h_complex_repr : ((∑ i in (Finset.range k : Finset ℕ),
   Real.cos (a i) / ((2 : ℕ) ^ i : ℝ), ∑ i in (Finset.range k :
   Finset ℕ), Real.sin (a i) / ((2 : ℕ) ^ i : ℝ)⟩ : ℂ) = ∑ i in
   (Finset.range k : Finset ℕ), Complex.exp (↑(a i) * Complex.I)
    / ↑(((2 : ℕ) ^ i) : ℝ)
  simp [Complex.exp_mul_I, div_eq_inv_mul, Complex.ext_iff]
```

```
  simp [Complex.cos_ofReal_re, Complex.sin_ofReal_re] <;>
    field_simp <;> norm_cast
  constructor <;> apply Finset.sum_congr <;> aesop
  field_simp [_root_.pow_add, show (4 : ℝ) = 2 ^ 2 by norm_num]
    <;> ring
  norm_num [mul_comm _ 2, pow_mul]
  rewrite [show (4 : ℝ) ^ x = (2 * 2 : ℝ) ^ x by ring, mul_pow]
    <;> field_simp
    <;> ring

  have h_sum_split : ∑ i in (Finset.range k : Finset ℕ),
    Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ) ^ i) : ℝ)) =
    Complex.exp (↑(a 0) * Complex.I) + ∑ i in (Finset.Icc 1 (k-1)
     : Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ) ^
     i) : ℝ)
  have h_range_split  : Finset.range k = insert 0 (Finset.Icc 1
    (k - 1))
  ext x  <;>  simp  [Nat.lt_succ_iff]
  rcases x with (_|_|x) <;> omega
  rw [h_range_split, Finset.sum_insert]
  norm_num [pow_zero, eq_self_iff_true]
  simp [Nat.le_zero]

  have h_abs_head : Complex.abs (Complex.exp (↑(a 0) * Complex.I
    )) = 1
  simp [Complex.abs_exp, eq_self_iff_true]

  have h_tail_geom_sum_val : ∑ i in (Finset.Icc 1 (k - 1) :
    Finset ℕ), 1 / ((2 : ℕ) ^ i : ℝ)) = 1 - 1 / (2 : ℝ) ^ (k - 1)
  have h_tight  : (1 : ℝ) ≤ k
  norm_cast at * <;>
  linarith
  clear h_tight h_sum_split h_complex_repr h_y_expand
    h_y_sum_split h_y_sum_expanded h_cos_add h₂ h₃ h₁ h₀
  induction' k <;> simp [Finset.sum_Icc_succ_top, *]
  induction' ⟨ℕ⟩ <;> simp_all [Finset.sum_Icc_succ_top, pow_succ
    ]
  ring
  <;>ring_nf

  have h_abs_tail_le : Complex.abs ∑ i in (Finset.Icc 1 (k-1) :
     Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ) ^ i
    ) : ℝ)) ≤ 1 - 1 / (2 : ℝ) ^ (k - 1)
  rw [← h_tail_geom_sum_val]
  apply (Complex.abs.sum_le _ _).trans_eq
  apply Finset.sum_congr rfl
  intro i _
  simp [Complex.abs_exp_ofReal_mul_I, Nat.cast_pow, Nat.
    cast_ofNat]

  have h_abs_tail_lt_one : Complex.abs (∑ i in (Finset.Icc 1 (k
    -1) : Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ
    ) ^ i) : ℝ)) < 1
  refine lt_of_le_of_lt h_abs_tail_le ?_
  refine sub_lt_self _ (by positivity)

  have h_abs_ge_by_rev_triangle : Complex.abs (∑ i in (Finset.
    range k : Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2
     : ℕ) ^ i) : ℝ)) ≥ 1 - Complex.abs (∑ i in (Finset.Icc 1 (k
    -1) : Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ
    ) ^ i) : ℝ))
  rw [h_sum_split]
```

```
  rw [← h_abs_head]
  apply Complex.abs.le_add
```

imo_1969_p2 - **Part 2**

```
  have h_abs_ge_final : Complex.abs (∑ i in (Finset.range k :
    Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ) ^ i)
     : ℝ)) ≥ 1 / (2 : ℝ) ^ (k-1)
  refine' _root_.trans h_abs_ge_by_rev_triangle _
  linarith [h_abs_tail_le]

  have h_abs_gt_zero : 0 < Complex.abs (∑ i in (Finset.range k :
     Finset ℕ), Complex.exp (↑(a i) * Complex.I) / ↑(((2 : ℕ) ^ i
    ) : ℝ))
  linarith [pow_two_nonneg ((k - 1 : ℕ) : ℝ) ]

  have h_complex_val_ne_zero : (⟨∑ i in (Finset.range k : Finset
     ℕ), Real.cos (a i) / ((2 : ℕ) ^ i : ℝ), ∑ i in (Finset.range
     k : Finset ℕ), Real.sin (a i) / ((2 : ℕ) ^ i : ℝ)⟩ : ℂ) ≠ 0
  focus all_goals (norm_num; aesop)
```

```
  have h_coeffs_polar : ∃ (R b : ℝ), 0 < R ∧ (∑ i in (
    Finset.range k : Finset ℕ), Real.cos (a i) / ((2 : ℕ) ^
     i : ℝ)) = R * Real.cos b ∧ (∑ i in (Finset.range k :
    Finset ℕ), Real.sin (a i) / ((2 : ℕ) ^ i : ℝ)) = R *
    Real.sin b
  set x := ∑ i ∈ Finset.range k, cos (a i) / ((2 : ℝ) ^ i)
  use Complex.abs (∑ i ∈ Finset.range k, Complex.exp (↑(a i
    ) * Complex.I) / ↑(↑2 ^ i))
  let y : ℝ := ∑ i ∈ Finset.range k, sin (a i) / 2^i
  have h := Complex.abs_mul_cos_add_sin_mul_I (∑ i in
    Finset.range k, Complex.exp ((a i : ℝ) * Complex.I) /
    (2 : ℂ) ^ i)
  use  Complex.arg (∑ i in Finset.range k, Complex.exp (↑(a
     i) * Complex.I) / (2:ℝ) ^ i)
  simp_all [Complex.ext_iff]
```

```
  have h_y_rewritten_with_polar : ∃ (R a : ℝ), 0 < R ∧ ∀ x,
     y x = R * Real.cos a * Real.cos x - R * Real.sin a *
    Real.sin x
  obtain ⟨R, phi, hR_pos, h_cos_eq1, h_sin_eq1⟩ :=
    h_coeffs_polar
  use R, phi <;> simp_all[Complex.exp_mul_I, Complex.abs]
```

```
  have h_y_collapsed_to_single_cos : ∃ (R  a : ℝ), 0 < R ∧
    ∀ x, y x = R * Real.cos (x + a)
  rcases h_y_rewritten_with_polar with ⟨R, a', h_R_pos,
    h_y_⟩
  use R, a', h_R_pos <;> intros <;> simp [h_y_, cos_add]
    <;> ring
```

```
have h_y_is_sinusoid : ∃ (R a : ℝ), 0 < R ∧ (∀ x, y x = R *
  Real.cos (x - a))
obtain ⟨ R, a, _, hy  ⟩ := h_y_collapsed_to_single_cos
use R, -a <;> aesop

have h_roots_exist : ∃ (R a : ℝ), 0 < R ∧ y m = R * Real.cos (
  m - a) ∧ y n = R * Real.cos (n - a)
rcases h_y_is_sinusoid with ⟨R, a, h_R_pos, h_y_R_a⟩
exact ⟨ R, a, h_R_pos,
  by simp [h_y_R_a], by simp [h_y_R_a] ⟩

have h_cos_zero : ∃ (R a : ℝ), 0 < R ∧ Real.cos (m - a) = 0 ∧
  Real.cos (n - a) = 0
rcases h_roots_exist with ⟨R, a, h_rPos,  h_mEq,  h_nEq⟩
exact
  ⟨R, a, h_rPos,
    by have := h₂; have := h₃; field_simp [h₁] at * <;>
  nlinarith,
    by have := h₃; have := h₂; field_simp [h₁] at * <;>
  nlinarith⟩

have h_roots_in_pi_half_multiples : ∃ (a : ℝ) (t₁ t₂ : ℤ), m -
   a = (2 * (t₁ : ℝ) + 1) * Real.pi / 2 ∧ n - a = (2 * (t₂ : ℝ)
   + 1) * Real.pi / 2
rcases h_cos_zero with ⟨R, a, _, h_m_cos_zero, h_n_cos_zero⟩
rw [ cos_eq_zero_iff] at h_m_cos_zero  h_n_cos_zero
exact ⟨ a,  ↑( Classical.choose h_m_cos_zero ) , ↑( Classical.
  choose h_n_cos_zero ) , by convert h_m_cos_zero.choose_spec ,
   by convert  h_n_cos_zero.choose_spec ⟩

have h_m_minus_n_form : ∃ t₁ t₂ : ℤ, m - n = ((2 * (t₁ : ℝ) +
  1) * Real.pi / 2) - ((2 * (t₂ : ℝ) + 1) * Real.pi / 2)
obtain ⟨z, t₁, t₂, h_z_root_m, h_z_root_n⟩ :=
  h_roots_in_pi_half_multiples
refine ⟨t₁ , t₂,?_⟩<;>
linarith

have h_m_minus_n_simplified : ∃ t₁ t₂ : ℤ, m - n = (↑(t₁ - t₂)
   : ℝ) * Real.pi
rcases h_m_minus_n_form with ⟨t₁, t₂, h_form⟩  <;>
  exists t₁  <;> exists t₂  <;>  field_simp at h_form ⊢  <;>
  linarith

obtain ⟨t₁, t₂,h_m_sub_n_t₁_t₂⟩ := h_m_minus_n_simplified  <;>
    use t₁ - t₂  <;>  linarith [h_m_sub_n_t₁_t₂]
```

D Prompts Used in This Work

D.1 Prompts for Autoformalization

Our autoformalization pipeline operates in two stages to ensure syntactic correctness. First, an `Initial Formalization Prompt` (shown below) translates a natural language problem into a Lean 4 theorem statement. If the generated code fails to compile, an `Error Feedback Prompt` is then deployed to revise the statement, using the verbatim error message from the Lean compiler as direct feedback for revision.

`Prompt for Initial Formalization`

You are an expert in math proof and the theorem prover: Lean. Given a math problem that contains the question and all conditions, and its corresponding solution that contains solution steps and the correct answer, generate a mathematically equivalent proof problem and rewrite it in the Lean 4 statement. You should follow the following procedures.

a): Identify all questions and conditions in the given problem.

b): Identify all solution steps and the correct answers in the given solution.

c): With the questions and conditions in a) and correct answers in b), translate the (question, conditions, correct answer) tuple to a mathematically equivalent proof problem that proves question == answer given conditions.

d): Rewrite the math proof problem in c) to a Lean 4 statement. Note that you should write the statement only, no proof is required. This also means you do not need to consider the solution steps either.

The first priority is to ensure the generated Lean code can be built successfully. Consider using the following tips.

- Use a broader import, e.g., `import Mathlib`, to bring in the entirety of the necessary library, and remove specific import of submodules, e.g., `import Mathlib.LinearAlgebra.BasicReal3`, accordingly.
- Add `noncomputable` before `def` only when necessary.
- Use by instead of `begin end`.
- Add `sorry` to skip the proof.

You should strictly follow the below criteria to guarantee the lean statement is equivalent to the mathematical problem.

- Each definition used in Lean 4 statement should only directly appear in the conditions problem in a).
- Each definition should NOT come from and assume any knowledge directly from the solution step in b).
- Each condition in a) should be used as a definition in Lean 4.
- For any implications appearing in the conclusions of the original problem, extract their antecedents and declare them as explicit assumptions before the colon, leaving only the consequent in the conclusion after the colon.
- For equations, structure the theorem in the form 'conditions : conclusions' where conditions include variable definitions and domains, and conclusions are the solutions to the equation, avoiding implication or equivalence symbols.

Below are examples to illustrate the process:
Example 1 (Number Theory):
Lean 4 statement:

```
theorem nt3_problem (n p : ℕ) (hn : n > 1) (hp : Nat.Prime p)
  (h1 : n ∣ (p - 1)) (h2 : p ∣ (n^6 - 1)) :
  ∃ k : ℕ, (p - n = k^2) ∨ (p + n = k^2) := by
  sorry
```

problem:
NT3. Let $n > 1$ be a positive integer and p a prime number such that $n \mid (p-1)$ and $p \mid (n^6 - 1)$. Prove that at least one of the numbers $p - n$ and $p + n$ is a perfect square.
Example 2 (Number Theory):
Lean 4 statement:

```
theorem nt4_problem (x y : ℕ)
  (hx : x > 0) (hy : y > 0)
  (h1 : ∃ m : ℕ, 3 * x + 4 * y = m^2)
  (h2 : ∃ n : ℕ, 4 * x + 3 * y = n^2) :
  7 ∣ x ∧ 7 ∣ y := by
  sorry
```

problem:
NT4. If the positive integers x and y are such that both $3x + 4y$ and $4x + 3y$ are perfect squares, prove that both x and y are multiples of 7.
Example 3 (Algebra):
Lean 4 statement:

```
theorem sum_not_zero (a b c d : ℝ)
  (h1 : a * b * c - d = 1)
  (h2 : b * c * d - a = 2)
  (h3 : c * d * a - b = 3)
  (h4 : d * a * b - c = -6) :
  a + b + c + d ≠ 0 := by
  sorry
```

problem:
The real numbers a, b, c, d satisfy simultaneously the equations $abc - d = 1, bcd - a = 2, cda - b = 3, dab - c = -6$. Prove that $a + b + c + d \neq 0$.
Example 4 (Inequality):
Lean 4 statement:

```
theorem inequality_proof (a b c : ℝ)
  (ha : a > 0) (hb : b > 0) (hc : c > 0) :
  8 / ((a + b)^2 + 4*a*b*c) +
  8 / ((b + c)^2 + 4*a*b*c) +
  8 / ((c + a)^2 + 4*a*b*c) +
  a^2 + b^2 + c^2 ≥
  8 / (a + 3) + 8 / (b + 3) + 8 / (c + 3) := by
  sorry
```

problem:
The real numbers a, b, c, d satisfy simultaneously the equations $abc - d = 1, bcd - a = 2, cda - b = 3, dab - c = -6$. Prove that $a + b + c + d \neq 0$.
Now, use the same process for the following problem and solution:
{**problem**}
{**solution**}

Prompt for Error Feedback

You are an expert in math proof and the theorem prover: Lean. You are given the following math problem that contains the question and all conditions, and its corresponding solution that contains solution steps and the correct answer.
{**problem**}
{**solution**}

A mathematically equivalent proof problem that proves question == answer given conditions is generated and rewritten in the Lean 4 statement, as shown below:
{**Lean 4 statement**}
However, this lean code got error with `lake build`, and here is the error message:
{**error message**}
Please modify the lean code to ensure it can be built successfully with `lake build`. Here is a few tips that might help:

- Use a broader import, e.g., `import Mathlib`, to bring in the entirety of the necessary library, and remove specific import of submodules, e.g., `import Mathlib.LinearAlgebra.BasicReal3`, accordingly.
- Add `noncomputable` before `def` only when necessary.
- Use by instead of `begin end`.
- Add `sorry` to skip the proof.

D.2 Prompts for Planner

`Prompt for Initial Planning`

You are an expert assistant specializing in Math Olympiads and the Lean 4 theorem prover. Your primary goal is to generate **syntactically perfect, type-checkable** Lean 4 intermediate steps for a given theorem. Strictly adhere to the following rules. ANY violation will be considered an error. While ensuring correctness, generate as many intermediate steps as possible.
Task
Given the following theorem statement in Lean 4, your job is to **plan the complete proof** by analyzing the theorem statement and generating a coherent sequence of `have` statements. These statements should form a clear chain of reasoning that bridges the theorem's assumptions to its final claim, breaking down complex arguments into simpler components.
Mandatory Rules
You must comply with every rule in this section. Failure to adhere to any single rule will result in an incorrect output.

1. **Critical Rule: Explicitly Specify Set/Finset Types**
   This is the most common and fatal point of error. You must explicitly declare the type for any `Set` or `Finset` literal. This rule is non-negotiable.

   ```
   - Correct: ({ {-1, 0, 1}} : Set ℤ)
   - Incorrect: { {-1, 0, 1}}
   ```

2. **Omit the Proof**: Never provide the proof. Only state the `have` statement itself.
3. **Valid Lean 4 Code**: The entire output block must be type-checkable in a Lean 4.10.0 environment.
4. **Use Existing Names**: Use the exact, existing lemma and definition names from `mathlib`. Do not invent names.
5. **No Undeclared Variables**: Do not introduce any variables or constants not declared in the original theorem statement.
6. **Explicit Multiplication**: Multiplication must always use the `*` symbol.

   ```
   - Correct: a * x
   - Incorrect: ax
   ```

7. **No Chained Inequalities**: Never use chained inequalities. They must be split using logical AND $\wedge$.

```
- Correct: a <= x ∧ x <= b
- Incorrect: a <= x <= b
```

8. **Correct Logarithm Function**: `Real.log` is only for the natural logarithm. For logarithms with a specified base, you must use `Real.logb`.

```
- Correct: Real.logb (2 : ℝ) 8
- Incorrect: Real.log (2 : ℝ) 8
```

9. **Factorial Notation**: In Lean, factorials must be written as `(n)!` or `Nat.factorial n`, not `n!`.

```
- Correct: (n)! or Nat.factorial n
- Incorrect: n!
```

10. **Numeric Types Must Be Explicitly Annotated**: To avoid type ambiguity in Lean, any expression involving numeric operations must have at least one number's type specified.

```
- For division: (1 : ℝ) / 2 = 0.5, but (1 : ℤ) / 2 = 0.
- For subtraction: (1 : ℤ) - 2 = -1, but (1 : ℕ) - 2 = 0.
- Correct: (a : ℝ) / b, a / (b : ℝ), (n : ℤ) - m
- Incorrect: a / b, n - m
```

11. **Interval Notation**: Do not use `Icc`, `Ioo`, `Ico`, `Ioc`, etc., to represent intervals. Only use inequalities.

```
- Correct: a <= x ∧ x <= b
- Incorrect: Icc a b
```

12. **Complex Numbers**: Use `Complex.I` for the imaginary unit and `Complex.abs` for the modulus/absolute value of a complex number.
13. **Avoid Common Inequality Theorems**: Avoid using common inequality theorems like Holder's or Jensen's. For inequality problems, try to ensure each proof step only requires basic simplification.
14. **Proving Equivalences**: When handling equivalences $\leftrightarrow$ (iff), produce `have` statements as implications.

```
- Left-to-right: assume LHS, conclude RHS.
- Right-to-left: assume RHS, conclude LHS.
```

15. **Real.pi Notation**: Always use `Real.pi`, not π.
16. **Final Check**: Before providing the plan, perform a final review to ensure you have scrupulously followed all the rules above, especially the critical rule regarding `Set`/`Finset`.

Examples:
Below are examples to illustrate the process and input/output format.
Input:

```
theorem singapore2019_r1_p7 (x : ℝ) (hx : Real.tan x = 5) :
(6 + Real.sin (2 * x)) / (1 + Real.cos (2 * x)) = 83 := by
```

Output:

```
have h₁ : Real.sin x = 5 * Real.cos x
have h₂ : Real.sin x ^ 2 = 25 * Real.cos x ^ 2
have h₃ : 26 * Real.cos x ^ 2 = 1
have hsin2x_val : Real.sin (2 * x) = (5 : ℝ) / (13 : ℝ)
have hcos2x_val : Real.cos (2 * x) = -(12 : ℝ) / (13 : ℝ)
```

Input:

```
theorem problem4
(g : ℕ → ℝ)
(h : ∀ k : ℕ, 5 ≤ k → k ≤ 124 → g k = (Real.logb (k : ℝ) ((7 : ℝ
    ) ^ (k ^ 2 - 1))) / (Real.logb ((k + 1 : ℝ)) ((7 : ℝ) ^ (k ^
    2 - 4)))) :
(∏ k in Finset.Icc (5 : ℕ) 124, g k) = (41 : ℝ) / 7 := by
```

Output:

```
have h_prod_split : (∏ k in (Finset.Icc 5 124 : Finset ℕ), g k)
    = (∏ k in (Finset.Icc 5 124 : Finset ℕ), ((k ^ 2 - 1) / (k ^
    2 - 4 : ℝ))) * (∏ k in (Finset.Icc 5 124 : Finset ℕ), (Real.
    logb (k : ℝ) (7 : ℝ) / Real.logb ((k + 1 : ℝ)) (7 : ℝ)))
have h_telescope_part1 : (∏ k in (Finset.Icc 5 124 : Finset ℕ),
    ((k ^ 2 - 1) / (k ^ 2 - 4 : ℝ))) = (41 : ℝ) / 21
have h_telescope_part2 : (∏ k in (Finset.Icc 5 124 : Finset ℕ),
    (Real.logb (k : ℝ) (7 : ℝ) / Real.logb ((k + 1 : ℝ)) (7 : ℝ))
    ) = 3
have h_final_product : (41 / 21 : ℝ) * 3 = (41 : ℝ) / 7
```

Input:

```
theorem amc12b_variant_p13
(S : Finset ℝ)
(h₀ : ∀ (x : ℝ), x ∈ S ↔ 0 < x ∧ x ≤ 2 * Real.pi ∧ 2 - 4 * Real.
    sin x + 3 * Real.cos (3 * x) = 0) :
S.card = 4 := by
```

Output:

```
have h_interval1 : ∃ x, 0 ≤ x ∧ x < Real.pi / 2 ∧ (2 - 4 * Real.
    sin x + 3 * Real.cos (3 * x) = 0)
have h_interval2 : ∃ x, Real.pi / 2 ≤ x ∧ x < 3 * Real.pi / 4 ∧
    (2 - 4 * Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_interval3 : ∃ x, 3 * Real.pi / 4 ≤ x ∧ x < Real.pi ∧ (2 -
     4 * Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_interval4 : ∃ x, Real.pi ≤ x ∧ x < 2 * Real.pi ∧ (2 - 4 *
     Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_card_eq_4 : S.card = 4
```

You must follow all the instructions and mandatory rules above. After deep consideratioin, output the complete plan in Lean for the input below.
{**theorem**}

Prompt for Dynamic Replanning

You are an expert assistant specializing in Math Olympiads and the Lean 4 theorem prover. Your primary goal is to generate **syntactically perfect, type-checkable** Lean 4 intermediate steps for a given theorem. Strictly adhere to the following rules. ANY violation will be considered an error. While ensuring correctness, generate as many intermediate steps as possible.

Task

Given the following theorem statement, along with already proven subgoals and a currently stuck subgoal, your job is to **replan the remaining proof** by analyzing the stuck subgoal and generating a coherent sequence of `have` statements, either by correcting the stuck subgoal or decomposing the stuck subgoal into smaller, logically consistent steps leading toward the theorem's conclusion.

The plan must have all proven subgoals in their original order and position. Only insert new `have` statements **immediately after** them.
These new statements should form a clear chain of reasoning that bridges the current progress to the final theorem, breaking down complex reasoning into simpler components.
Mandatory Rules
You must comply with every rule in this section. Failure to adhere to any single rule will result in an incorrect output.
(The rules are identical to those in the `Prompt for Initial Planning`, with the three additional rules shown below introduced before **Final Check**.)

16. **Insert After Proven Steps**: All new auxiliary `have` statements must be inserted **immediately after** the provided proven subgoals. The proven subgoals' wording, order, and placement are **immutable**. They must remain exactly as given, with no edits, insertions, or extensions inside them.
17. **Provide Complete Plan**: Output the **entire updated plan**, preserving the proven subgoals **exactly as given** and appending new `have` statements after them in the correct order. Do **not** output only the new ones. The plan must contain multiple new `have` statements with sufficient intermediate steps to meaningfully connect to the theorem's conclusion.
18. **Ensure Logical Continuity**: Each new step must be logically sound. Avoid repetition of the stuck subgoal. If the stuck subgoal is restated, it must not appear immediately after the last proven subgoal. New intermediate steps must be introduced in between.

Examples:
Below are examples to illustrate the process and input/output format.
Input:
Theorem

```
theorem singapore2019_r1_p7 (x : ℝ) (hx : Real.tan x = 5) :
(6 + Real.sin (2 * x)) / (1 + Real.cos (2 * x)) = 83 := by
```

Proven Subgoals

```
-- (None)
```

Stuck Subgoal

```
have h_sin4x_is_2sin2xcos2x : Real.sin (4 * x) = 2 * Real.sin (2
    * x) * Real.cos (2 * x)
```

Output:

```
have h₁ : Real.sin x = 5 * Real.cos x
have h₂ : Real.sin x ^ 2 = 25 * Real.cos x ^ 2
have h₃ : 26 * Real.cos x ^ 2 = 1
have hsin2x_val : Real.sin (2 * x) = (5 : ℝ) / (13 : ℝ)
have hcos2x_val : Real.cos (2 * x) = -(12 : ℝ) / (13 : ℝ)
```

Input:
Theorem

```
theorem amc12b_variant_p13
(S : Finset ℝ)
(h₀ : ∀ (x : ℝ), x ∈ S ↔ 0 < x ∧ x ≤ 2 * Real.pi ∧ 2 - 4 * Real.
    sin x + 3 * Real.cos (3 * x) = 0) :
S.card = 4 := by
```

Proven Subgoals

```
have h_interval1 : ∃ x, 0 ≤ x ∧ x < Real.pi / 2 ∧ (2 - 4 * Real.
    sin x + 3 * Real.cos (3 * x) = 0)
```

```
have h_interval2 : ∃ x, Real.pi / 2 ≤ x ∧ x < 3 * Real.pi / 4 ∧
    (2 - 4 * Real.sin x + 3 * Real.cos (3 * x) = 0)
```

Stuck Subgoal

```
have h_card_eq_4 : S.card = 4
```

Output:

```
have h_interval1 : ∃ x, 0 ≤ x ∧ x < Real.pi / 2 ∧ (2 - 4 * Real.
    sin x + 3 * Real.cos (3 * x) = 0)
have h_interval2 : ∃ x, Real.pi / 2 ≤ x ∧ x < 3 * Real.pi / 4 ∧
    (2 - 4 * Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_interval3 : ∃ x, 3 * Real.pi / 4 ≤ x ∧ x < Real.pi ∧ (2 -
     4 * Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_interval4 : ∃ x, Real.pi ≤ x ∧ x < 2 * Real.pi ∧ (2 - 4 *
     Real.sin x + 3 * Real.cos (3 * x) = 0)
have h_card_eq_4 : S.card = 4
```

Input:
Theorem:

```
theorem trig_identity_4x (x : ℝ) :
Real.sin (4 * x) = 4 * Real.sin x * Real.cos x * (1 - 2 * Real.
    sin x ^ 2) := by
```

Proven Subgoals:

```
have h_sin4x_is_2sin2xcos2x : Real.sin (4 * x) = 2 * Real.sin (2
     * x) * Real.cos (2 * x)
```

Stuck Subgoal:

```
have h_identity : Real.sin (2 * x) * Real.cos (2 * x) = 2 * Real
    .sin x * Real.cos x * (1 - 2 * Real.sin x ^ 2)
```

Output:

```
have h_sin4x_is_2sin2xcos2x : Real.sin (4 * x) = 2 * Real.sin (2
     * x) * Real.cos (2 * x)
have h_sin2x : Real.sin (2 * x) = 2 * Real.sin x * Real.cos x
have h_cos2x_in_terms_of_sin_cos : Real.cos (2 * x) = Real.cos x
     ^ 2 - Real.sin x ^ 2
have h_cos2x_in_terms_of_sin : Real.cos (2 * x) = 1 - 2 * Real.
    sin x ^ 2
have h_final_identity : 2 * Real.sin (2 * x) * Real.cos (2 * x)
    = 4 * Real.sin x * Real.cos x * (1 - 2 * Real.sin x ^ 2)
```

You must follow all the instructions and mandatory rules above. After deep consideratioin, output the complete refined plan in Lean for the input below.
Input:
Theorem
{**theorem**}
Proven Subgoals
{**proven_subgoals**}
Stuck Subgoal
{**stuck_subgoal**}