NanoProof: Open and Efficient Automated Theorem Proving in Lean 4

Anonymous Author(s)

Abstract

We introduce *NanoProof*, the first factorized execution-guided theorem prover that is end-to-end reproducible in Lean 4 using open-source resources. We release a dataset of structured proof trees and a tool for programmatic interaction with the Lean 4 verifier. Thanks to several algorithmic improvements, NanoProof achieves 50.8% on MiniF2F-Test using ≈10,000× less compute than AlphaProof, proving that competitive tree-search-based theorem proving is viable for the open-source community.

1 Introduction

Formal automated theorem provers (ATP) can be roughly classified along three axes: *execution-guided* vs. *execution-agnostic* (whether each step is conditioned on the true proof state), *factorized* vs. *linear* (whether independent sub-goals can be explored in parallel), and *any-time* vs. *one-shot* (whether performance scales with compute on a single problem). Pure text-to-text systems — linear, one-shot, and execution-agnostic — directly benefit from frontier LLM advances, as exemplified by DeepSeek-Prover-V2 [9] and Kimina-Prover [11]. Factorized any-time execution-guided provers — e.g. HyperTree [5] and AlphaProof [2] — tightly integrate search with the verifier and have demonstrated strong sample efficiency, but all such systems to date have closed code or data and require immense compute budgets, putting the paradigm out of reach for the open-source community.

We address this reproducibility barrier with *NanoProof*, an end-to-end open system contributing (i) a factorized any-time execution-guided prover with full reproduction tooling; (ii) `LeanTree`, 92,927 verified structured proof trees extracted from Mathlib [7], plus the extraction tool and a Python interface to Lean for distributed RL; (iii) several algorithmic improvements to AlphaProof-style search; and (iv) a score of 50.8% on MiniF2F-Test [13] using ≈10,000× less compute than AlphaProof.

2 Method

Search algorithm and network. Following AlphaProof [2], the prover maintains an AND-OR search tree over factorized proof states, expanded via an AlphaZero-style [10] selection, expansion, and backpropagation loop. The reward is set to $r = -d$, where d is the proof-tree depth, encouraging short and branching proofs. A single 1B-parameter Transformer (based on Karpathy's nanochat [3]) serves as both actor and critic via a shared-vocabulary head [8]: it autoregressively decodes tactic tokens and predicts the critic value in one forward pass via dedicated value-bin tokens. The network is base-trained on 10B math/code tokens, fine-tuned on 258k LeanTree proof steps, then enters a *self-improvement loop* using 96,967 autoformalized statements, alternating proof collection with gradient updates on a replay buffer.

Algorithmic improvements. (1) *Proof simplification.* Generated proofs are pruned by contracting non-leaf edges whenever the resulting sub-proof remains valid; this simplifies 59% of multi-tactic proofs in a representative run, improving replay-buffer quality and readability. (2) *Proof augmentation.* Buffer samples have hypotheses and goals randomly renamed and shuffled. (3) *Cycle detection.* A tactic is discarded if it produces a goal syntactically equivalent (up to renaming, shuffling, and whitespace) to one already on the path to root; this rejects 18% of otherwise valid tactics in a representative run. (4) *Diverse tactic sampling.* Autoregressive sampling tends to produce duplicate short tactics; we cap the occurrences of each first token, improving MiniF2F-Valid by 2.0% at 16 simulations.

LeanTree dataset. Lean 4 [1] treats proofs as linear sequences of tactics, but factorized execution-guided training requires *trees* where each node carries a single goal and each edge is a self-contained tactic application. Existing Lean tactic-state datasets [12] do not expose this structure and lack distance-to-proof-end labels. `LeanTree` builds proof trees in three stages: (i) match tactic applications to the goals they affect, using an augmented BetterParser from PaperProof [4]; (ii) simplify complex tactics — e.g. nested `by` blocks, `calc`, `cases`/`induction` branches, and merged `rw` chains — into self-contained steps; (iii) merge sibling goals sharing metavariables. Each tree is independently re-verified top-down, yielding 92,927 verified trees (90.5% of Mathlib tactic proofs). The tool also ships a Python interface over Lean REPL [6] with a dynamic pool of CPU-side environments suitable for distributed RL.

3 Experiments

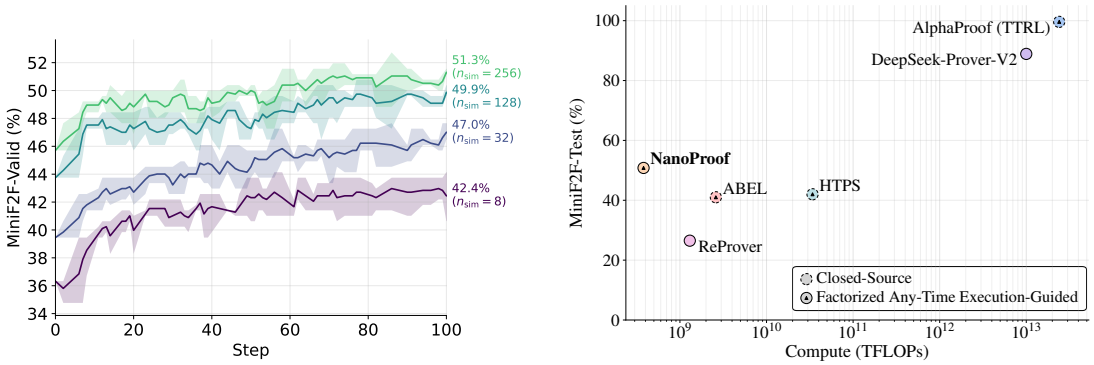

Figure 1: **Left:** NanoProof success rate on MiniF2F over RL training time and number of simulations n_{sim} (mean of three independent runs with min/max bands). **Right:** MiniF2F success rate vs. approximate total compute; NanoProof lies on the Pareto frontier and is the first open-source factorized any-time execution-guided prover in Lean 4.

After 100 RL steps (45 hours on 8 GPUs and 2 CPU nodes), NanoProof reaches a mean 51.3% on MiniF2F-Valid under 256 MCTS simulations (Figure 1, left); the best checkpoint, selected on the validation split, achieves **50.8% pass@16 on MiniF2F-Test** under 512 simulations. Figure 1 (right) places this on the Pareto frontier, reaching AlphaProof-comparable territory at ≈10,000× less compute. All code, data, and checkpoints are released, lowering the entry bar for tree-search-based neural theorem proving and inviting the AITP community to build on top.

References

[1] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28*, pages 625–635. Springer, 2021.

[2] T. Hubert, R. Mehta, L. Sartran, M. Z. Horváth, G. Žužić, E. Wieser, A. Huang, J. Schrittwieser, Y. Schroecker, H. Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025.

[3] A. Karpathy. nanochat: The best ChatGPT that $100 can buy. https://github.com/karpathy/nanochat, 2025.

[4] E. Karunus and A. Kovsharov. Paperproof: A new proof interface for Lean 4. https://github.com/Paper-Proof/paperproof, 2024.

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

[6] Lean Community. Lean 4 REPL. https://github.com/leanprover-community/repl, 2025.

[7] Mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pages 367–381, 2020.

[8] S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

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

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[11] H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

[12] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2023.

[13] K. Zheng, J. M. Han, and S. Polu. MiniF2F: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.