Reinforcement Learning for Hierarchical Proof Generation in Lean 4

Fabian Gloeckle [*]
Meta FAIR
École des Ponts Paris

Alex Gu
Meta FAIR
MIT

Gabriel Synnaeve
Meta FAIR

Amaury Hayat
École des Ponts Paris
Korean Institute for Advanced Studies

Abstract

Scaling formal theorem proving to more complex problems requires more efficient inference methods than standard whole-proof generation, which struggles with longer proofs due to exponentially decreasing success rates with increasing proof length. In this work, we systematically study how online reinforcement learning can be coupled with a hierarchical (lemma-based) style of proof generation, that has recently gained popularity for inference-time methods. We show a fruitful interaction in two ways: reinforcement learning allows to train proof decomposition policies successfully, and hierarchical inference allows to overcome plateaus in reinforcement learning as its richer distribution favors exploration and generated lemmas can be understood as an online synthetic data generation technique. Overall, hierarchical inference trained with reinforcement learning produces strong numbers on evaluations, even at the 7B parameter scale, and outperforms standard whole-proof generation setups in terms of training and evaluation sample efficiency and scalability, making it a suitable technique for necessarily data-constrained formalization research efforts.

1 Introduction

With the availability of increasingly powerful pretrained large language models, machine learning for formal theorem proving has made rapid progress in the past years. At the same time, modern models' higher accuracy and stability over long generations made new paradigms applicable for proof generation. While in early works, *tree search* with line-level tactic generation models that interact with the environment state provided by interactive theorem provers dominated (e.g. [Polu and Sutskever, 2020, Han et al., 2021, Lample et al., 2022, Wu et al., 2024a, Gloeckle et al., 2024]), more recent works moved to *whole-proof generation* where the model simply outputs the entire proof in one go [Jiang et al., 2023, Xin et al., 2024, Ren et al., 2025, Dong and Ma, 2025, Wang et al., 2025] and internalizes the *world modeling* of predicting state transitions in the environment.

However, full-proof generation suffers from *poor scalability* in terms of proof complexity.

To address this shortcoming, *decomposition and hierarchical recursion* is a natural method to apply to such complex construction tasks: problems are broken down into individual parts which are solved individually before re-assembling the overall solution from them.

While past works frequently either focused on hierarchical approaches to neural theorem proving in an inference-only or synthetic data generation setup, or on reinforcement learning but with fairly standard whole-proof generation methods (see Section B), in this work we study how reinforcement learning can be applied to hierarchical proving, choosing the Lean 4 proof assistant as a well-studied and convenient testbed.

Our findings are as follows:

[*]Correspondence to fgloeckle@meta.com

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

- Hierarchical proof generation can easily be trained with reinforcement learning with minimal amounts of supervised finetuning data specialized for the respective regime. Unlabeled reinforcement learning dataset size likewise can be several orders of magnitudes below that of current state-of-the-art methods.
- Hierarchical proof generation performs favorably compared to standard whole-proof generation, showing higher sample efficiency and plateauing later in reinforcement learning runs. In particular, they exhibit a cumulative solve rate that keeps increasing long after classical whole proof generation methods based on Group Relative Policy Optimization (GRPO) training plateau.
- The resulting models show convincing numbers both in the low and high sample regimes.

2 Background

2.1 Saturation in Reinforcement Learning

Unlike in language model pretraining where performance is predictably governed by *scaling laws* [Kaplan et al., 2020, Hoffmann et al., 2022, OpenAI, 2023], reinforcement learning with language models rather appears to follow a "saturation model": with increasing RL compute budget, evaluation performance shows diminishing returns and even degrades later in training [Cui et al., 2025, Yue et al., 2025].

2.2 Techniques to Mitigate Saturation

To delay saturation effects in reinforcement learning, various techniques have been proposed, reviewed here with a focus on theorem proving in Lean:

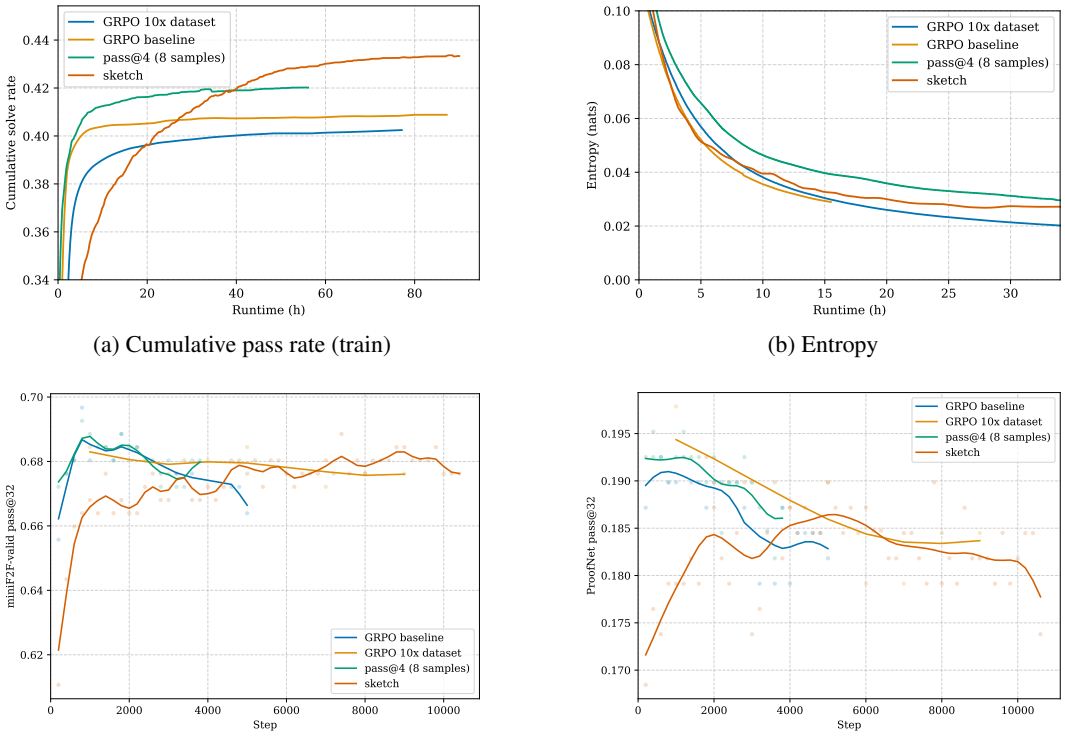
- **Expert iteration**, where data is collected from the current policy π_t, merged with data from previous iterations π_s, $s < t$, and used to finetune the next iteration π_{t+1} from the initial policy π_0 [Anthony et al., 2017]
- **Scaling training datasets** in order to reduce epoching and overfitting effects.
- **Pass@k training** and other alternative reward formulations that favor diversity by design [Chen et al., 2025b, Walder and Karkhanis, 2025, Tang et al., 2025].
- **Diverse contextual conditioning**, where the model is provided with various versions of additional context attached to each problem as a way to induce diversity and exploration (e.g. [Chen et al., 2025a] in the context of Lean).
- **Alternative inference schemes** such as agentic interactions with the environment, hierarchical decomposition and multi-agent inference or even tree search in the context of interactive theorem proving [Dong et al., 2024, Gehring et al., 2024, Lample et al., 2022, Gloeckle et al., 2024].

For theorem proving with language models returning a response in Lean, a training recipe has evolved that involves separate stages: pretraining; Lean-specific finetuning or distillation; expert iteration, and lastly online reinforcement learning [Lin et al., 2025, Ren et al., 2025]. In this work, we focus on mitigating saturation in the last stage: online reinforcement learning.

3 Method

3.1 Hierarchical Lemma-Based Proof Generation

Lemma-based Proving Hierarchical proof generation means decomposing the problem into subproblems which can be solved and evaluated independently. Concretely, we adopt a two-step design where first, a *decomposer* generates a *sketch* which provides a proof provided that the contained lemmas can be proved. The lemmas are then extracted and fed separately to a *prover* with multiple attempts per lemma. This design could be extended into a full recursive strategy where lemmas, too, can be broken down into subgoals further. But we do not employ such a scheme to minimize design choices for the reinforcement learning setup.



(a) Cumulative pass rate (train)

(b) Entropy

(c) miniF2F-valid evaluation

(d) ProofNet evaluation

Figure 1: **Reinforcement learning with hierarchical inference shifts plateaus. (a)** The fraction of training problems cumulatively solved over the course of training flattens off early with GRPO variants, but continues increasing with sketch-based proving, **(b)** which maintains a higher entropy similar to pass@k training. **(c, d)** Evaluation scores on miniF2F and ProofNet peak early in training for GRPO variants. With sketch-based proving, they start significantly lower but partially catch up later in training.

Reinforcement Learning For reinforcement learning, hierarchical generation comes with more choices than standard whole-proof generation. One could generate $k = 1$ proof attempts per lemma to form a trajectory, assign the overall proof success as reward and apply standard reinforcement learning algorithms, but as explained above that would not deviate from whole-proof generation except for the language model context resets for the individual parts. A natural option, instead, would be to conduct $k > 1$ proof attempts per lemma. In a sketch with n lemmas, this would result in k^n recombined trajectories, which, however, would no longer be independent and could break optimization due to their close correlation. Instead, we propose to **apply a (baselined) policy gradient algorithm at each node in the resulting tree** (concretely, GRPO without σ-normalization). See Appendix G for details.

Automatic Solvers Before sending a theorem statement to a prover model to solve, we attempt to close the goal with some of Lean's built-in automation. This serves two purposes: it speeds up inference since it reduces the required number of language model calls, and it filters out low-quality theorems and lemmas to allow the prover to focus on the most challenging problems. All reported numbers, both for whole-proof generation and sketch-based proving, include automation scheduled before language model-based proving. We detail the exact automation tactic used in Listing 1.

4 Experimental Setup: Toy Model for Saturation

Our experimental setup is designed to *investigate saturation in reinforcement learning* (Section 2). To enable a controlled study, we make two key decisions: **(1)** We opt for distilling two leading models within the 7B parameter class, deferring considerations of expert iteration. **(2)** We limit our reinforcement learning dataset to 6,000 samples – approximately 1,500 times smaller than the

Prover	Training budget (# Proofs)	Sample budget (#Attempts)	Sample budget (#LLM calls)	ProofNet *test*	MiniF2F *test*
Goedel-Prover-SFT 7B [Lin et al., 2025]	285M	32	–	15.6%	57.6%
		3200		–	62.7%
STP 7B [Dong and Ma, 2025] *without ProofNet-valid, miniF2F-valid*	240M	128	–	18.0%	57.2%
		3200	–	23.1%	61.1%
DeepSeek-Prover-V2 7B (CoT, Xin et al. [2024])	?	32	–	23.0%	75.6%
		128	–	25.4%	–
		1024	–	29.6%	79.9%
Ours 7B (sketch autoformalization)	8M attempts 75M lemmas	8	59 / 69	19.2%	55.7%
		32	237 / 278	20.4%	58.2%
		128	947 / 1111	21.4%	59.9%
		512	3787 / –	22.5%	–
		2048	15146 / –	23.9%	–

Table 1: **Results on miniF2F and ProofNet autoformalization, compared to various theorem proving methods.** Note that unlike other methods, our model has access to ground-truth and model-generated natural language solutions to evaluate proof autoformalization performance. Removing access to ground truth and only using model-generated natural language solutions does not significantly change the numbers. One sketch attempt corresponds to several language model calls depending on the number of lemmas in each sketch: we report the equivalent sample budget as $nk+1$ for ProofNet and miniF2F, respectively, in the central column. The training budget for GoedelProver is calculated from the number of expert iteration epochs, proofs per problem and dataset size. Note that for hierarchical proving, the *true cost* is somewhere between the count by number of LLM calls or lemmas (which does not factor in their shorter nature) and the number of attempts (which does not factor in overhead in LLM prefill or Lean execution setup).

expert iteration dataset used by DeepSeek-Prover-V1.5 [Xin et al., 2024]. This is both to model the data-constrained regime that naturally appears in several area such as specific area of research-level mathematics and to facilitate rapid experimental cycles.

See Appendix F.1 for details on our supervised finetuning setup, the model checkpoints used, and the specifics of our reinforcement learning data.

5 Results

Our experiments establish that our initial SFT checkpoint provides a performant starting point for analyzing reinforcement learning with Lean models (Section H), that GRPO indeed suffers from saturation that cannot be remedied with standard techniques (Section I.1), that sketch-based proving shows appealing initial results in terms of exploration and policy diversity (Section I.2) and results in competitive benchmark scores (Section 5).

We find that GRPO variants lateaus early in our RL setup. Sketch-based proving, on the other hand, continues exploring and maintains a higher entropy. See Figure 1 and Appendix I) for details.

Final Benchmark Results We evaluate the final sketch-based reinforcement learning checkpoint (step 10k) on proof **autoformalization** on miniF2F-test and ProofNet-test. Specifically, we generate 5 proofs per problem with Gemini 2.5 Pro and GPT-5 nano, and add a human-written ground truth proof from the dataset. For each attempt, we select one out of these 11 proofs per problem as input for the sketch-based autoformalizer, i.e. pass@k is computed over the mixed dataset of natural language proofs.

According to the results in Table 1, our proof autoformalizer performs on par with state-of-the-art theorem proving models despite using a short reinforcement learning over 10,000 steps based on only 6,000 training samples. We are not aware of other proof autoformalization models that produce a proof conditioned on a user-provided natural language proof which we could use as baselines.

6 Conclusion and Outlook

For scaling formal theorem proving to problems beyond mathematical competitions, test-time inference needs to be scaled by large amounts. While vanilla whole-proof generation suffers from exponential decay of proof rates with increasing proof complexity, lemma-based and iterative proof generation provide alternatives with better scaling behaviors.

In this work, we explored in a well-controlled setting how online reinforcement learning combined with hierarchical decomposition can improve performances and tackle the limitations of whole-proof generation methods. While not yet definitive, the evidence presented in this article suggests the strong potential of these hierarchical approaches and sketch-based proofs in formal mathematics.

Reproducibility Statement

Our approach, and in particular the adaptations to previous reinforcement learning approaches are detailed in Section 3. The data used to obtain the initial SFT checkpoint are detailed in Appendix F.2, while the data later used for training are specified in Section F.2. The evaluation datasets can be found at yangky11 and contributors [2025] for MiniF2F, with additionnal natural language solutions coming from [Research and contributors, 2022] and Azerbayev and contributors [2023] for ProofNet.

Acknowledgments

FG would like to thank all members of the FAIR CodeGen team, in particular Jonas Gehring, Taco Cohen and Kunhao Zheng, for advice and support in the reinforcement learning setup.

AH wishes to thank the PEPS Project JCJC 2025 from CNRS - INSMI and the Chair DESCARTES of Hi! Paris.

References

Leni Aniva, Chuyue Sun, Brando Miranda, Clark Barrett, and Sanmi Koyejo. Pantograph: A machine-to-machine interaction interface for advanced theorem proving, high level reasoning, and data extraction in lean 4. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 104–123. Springer, 2025.

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.

Hugh Leather Aram H. Markosyan, Gabriel Synnaeve. Leanuniverse: A library for consistent and scalable lean4 dataset management. 2025.

Zhangir Azerbayev and contributors. Proofnet: A benchmark for undergraduate-level formal mathematics. `https://github.com/zhangir-azerbayev/ProofNet`, 2023. Commit: 02b47219879da3a4fcb4d6ad5f2f29f384cf2de0, Accessed: 2025-09-25.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 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, 2025. *URL https://arxiv. org/abs/2507.23726*, 2025a.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@ k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*, 2025b.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Kefan Dong, Arvind Mahankali, and Tengyu Ma. Formal theorem proving by rewarding llms to decompose proofs hierarchically. *arXiv preprint arXiv:2411.01829*, 2024.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *Advances in Neural Information Processing Systems*, 36: 27092–27112, 2023.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

Fabian Gloeckle, Jannis Limperg, Gabriel Synnaeve, and Amaury Hayat. Abel: Sample efficient online reinforcement learning for neural theorem proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024.

Alex Gu, Bartosz Piotrowski, Fabian Gloeckle, Kaiyu Yang, and Aram H. Markosyan. Proofoptimizer: Training language models to simplify proofs without human demonstrations. *preprint*, 2025.

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.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35:26337–26349, 2022.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. `[https://huggingface.co/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)`, 2024.

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.

Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, et al. Combibench: Benchmarking llm capability for combinatorial mathematics. *arXiv preprint arXiv:2505.03171*, 2025.

OpenAI. Gpt-4 technical report, 2023.

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.

Facebook Research and contributors. minif2f: A dataset for formalizing mathematical problems and informal proofs. `https://github.com/facebookresearch/miniF2F`, 2022. Accessed: 2025-09-25.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. Optimizing language models for inference time objectives using reinforcement learning. *arXiv preprint arXiv:2503.19595*, 2025.

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. *Advances in Neural Information Processing Systems*, 37: 11545–11569, 2024.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024.

Christian Walder and Deep Karkhanis. Pass@ k policy optimization: Solving harder reinforcement learning problems. *arXiv preprint arXiv:2505.15201*, 2025.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, et al. Proving theorems recursively. *Advances in Neural Information Processing Systems*, 37:86720–86748, 2024.

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.

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*, 2024a.

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*, 2024b.

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.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis. *Advances in Neural Information Processing Systems*, 36:59304–59322, 2023.

yangky11 and contributors. minif2f-lean4: A lean 4 port of the minif2f dataset. `https://github.com/yangky11/miniF2F-lean4`, 2025. Commit: 88317cf19776c515d831fb3ad7afd6b85f4ff471, Accessed: 2025-09-25.

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. *Advances in Neural Information Processing Systems*, 37:105848–105863, 2024.

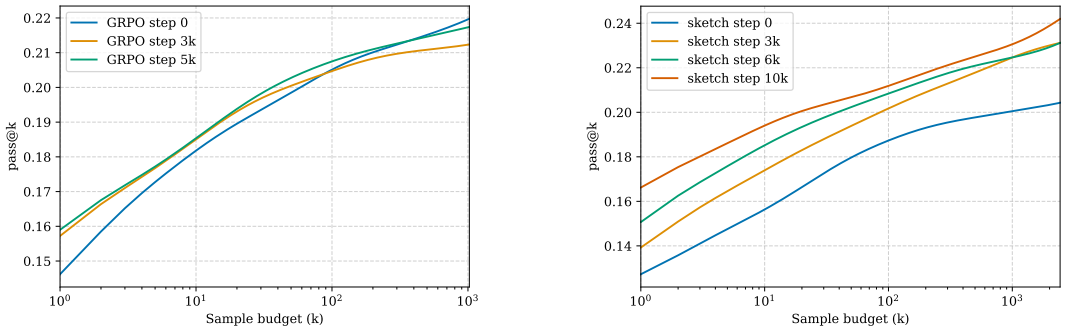
Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, et al. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL `https://arxiv.org/abs/2504.13837`.

Xueliang Zhao, Wenda Li, and Lingpeng Kong. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*, 2023.

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

A Inference Scaling Behavior



(a) Standard inference

(b) Lemma-based inference

Figure 2: **Inference scaling on ProofNet-test with standard and lemma-based inference.** GRPO training is unable to improve the model upon pass@1024 evaluations. Sketch-based RL starts at lower numbers and improves to up to a similar performance as standard training. Note that a sketch-based attempts consists of several language model calls, i.e. sample budgets are not comparable. Lemma-based inference uses generated natural language proofs. A similar scaling graph for miniF2F-test with sketch-based proving can be found in Figure 3.

B Related Work

DeepSeek-Prover-V2 [Xin et al., 2024] uses lemma-based proving for generating a warm-start dataset, but does not use lemma-based proving during reinforcement learning.

SeedProver [Chen et al., 2025a] alleges using reinforcement learning with both iterative and lemma-based inference schemes but does not disclose further details.

Draft-Sketch-Prove [Jiang et al., 2023] is foundational for proof autoformalization and uses sketches, but relies on classical automation for lemma proofs due to the unavailability of adequate proof generation models at the time.

Dong et al. [2024] apply reinforcement learning to lemma-based proving in Isabelle, arriving at marginal improvements over the baseline which may be linked to the intricacy of optimizing the proposed proof tree generation using a learned value function.

Zhao et al. [2023] rewrite Isabelle proofs in a lemma-based way using LLMs to generate a finetuning dataset.

Wang et al. [2024] uses a recursive lemma decomposition method, trained by supervised finetuning.

C Background: Token Crisis in LLM Training

With compute budgets projected to increase exponentially over the upcoming years, language model pretraining is predicted to hit the "token wall" when available organic data runs out compared to optimal scaling requirements [Xue et al., 2023, Gadre et al., 2023]. Beyond that point, synthetic data, post-training and reinforcement learning are expected to take an even more prominent role [Villalobos et al., 2024].

Language modeling for Lean can be seen as a case study for this situation: with organic tokens only ranging at the 100M scale [Aram H. Markosyan, 2025, Wu et al., 2024b], state of the art models already heavily rely on synthetic data and reinforcement learning [Lin et al., 2025, Ren et al., 2025, Dong and Ma, 2025] for additional data in the tens of billions of tokens scale, dwarfing the organic data by three orders of magnitude.

While our analysis will be focused on language models for Lean specifically, this study can also be understood as an early data point on best practices for *data-constrained* language modeling.

D Lean LSP Environment

We use a custom Lean environment that interacts with the Lean language server. This allows for fast interaction with Lean due to caching of imports. Lemma extraction is implemented using `extract_goal`.

E Automation Tactic

We use the following code for calling several automatic solvers built into Lean 4:

Listing 1: Automation tactic for filtering trivial theorems

```
repeat' {
  try rfl
  try tauto
  try assumption
  try norm_cast
  try norm_num
  try ring
  try { ring_nf at * }
  try { ring_nf! at *}
  try native_decide
  try omega
  try { simp_all }
  try { field_simp at * }
  try positivity
  try linarith
  try nlinarith
  try exact?
}
```

F Details on Supervised Finetuning and Data

F.1 Supervised Finetuning

As a starting point for reinforcement learning, we perform sequence distillation from two state-of-the-art 7B parameter models: STP [Dong and Ma, 2025], derived from DeepSeek-Prover-V1.5-SFT 7B [Xin et al., 2024], and GoedelProver [Lin et al., 2025], derived from DeepSeek-Prover-V1.5-RL 7B. We intentionally exclude distillation from larger models such as DeepSeek-Prover-V2 671B [Ren et al., 2025], Goedel-Prover-V2 32B [Lin et al., 2025], and their distilled variants, to maintain a controlled experimental environment.

Our finetuning data consists of: proofs from the above mentioned models, curated Lean 4 code in file-format and traced into state-tactic sequences, alongside natural language mathematical problem solving and statement autoformalization data (see Appendix F.2). Additionally, we include a small portion of proof sketching data obtained from rejection sampling to warm-start the new lemma-based proof style described above.

F.2 Data

We focus on the task of *proof autoformalization*, i.e. formal theorem proving conditioned on a proof given in natural language due to its high relevance for the community and its convenience for sketching. This setup allows us to focus on the Lean proficiency of the model without separately optimizing natural language mathematical reasoning skills simultaneously.

Concretely, we pair up natural language solutions from NuminaMath 1.5 [LI et al., 2024] to the problems in Goedel-Pset-v1, and use the resulting proof autoformalization dataset as our reinforcement

learning problem set. We evaluate on ProofNet [Azerbayev et al., 2023] and miniF2F [Zheng et al., 2021], for which we likewise generated natural language solutions.

Following Gu et al. [2025], we collect a finetuning dataset consisting of roughly 1B Lean tokens. The training data spans a diverse set of math and Lean-related tasks and includes the following:

- **Theorem proving**: conjectures and proofs from STP [Dong and Ma, 2025], and solution autoformalization data from the Goedel-Pset-v1-Solved dataset obtained by mapping Lean solutions with natural language solutions.
- **Natural language problem solving**: problems and solutions in natural language from the NuminaMath-1.5 dataset [LI et al., 2024].
- **Lean code**: Lean 4 code curated from GitHub, filtering with GPT-4o and manual heuristics to ensure only Lean 4 examples are included.
- **Auto-formalization**: mathematical statements in natural language paired up with their Lean counterparts, from the following high-quality datasets: CombiBench [Liu et al., 2025], Compfiles, FormalMATH [Yu et al., 2025], Goedel-Pset [Lin et al., 2025], Lean Workbook [Ying et al., 2024], miniF2F [Zheng et al., 2021], ProofNet [Azerbayev et al., 2023], and PutnamBench [Tsoukalas et al., 2024].
- **Tactic prediction**: Lean repositories traced at the tactic level into (state, tactic, next state) triples. We use data from LeanUniverse [Aram H. Markosyan, 2025] and extract additional data using the Pantograph [Aniva et al., 2025] tool from the STP dataset.

F.3 Reinforcement Learning with Limited Data

Reinforcement learning on finite datasets is limited by "plateaus", the maximal performance that can be extracted from the dataset using the reinforcement learning algorithm. In order to study the scalability of our hierarchical reinforcement learning method, we therefore adopt a two-stage approach of ablations conducted on 10 percent of the full dataset (6k samples), and then final runs using the full dataset (60k samples).

G Sketch RL

Concretely, applying a mean baseline at each node in the tree means the following. For the i-th proof attempt of lemma j, the reward is its proof success r_i^j and its advantage is

$$A_i^j = r_i^j - \frac{1}{k}\sum_{m=1}^{k} r_m^j.$$

For sketches, there are various options, detailed below. We opt for rewarding sketches with **overall proof success**:

$$R = \min_j \max_i r_i^j,$$

which depends on the number of lemma proof attempts k per lemma. In this work, we choose $k = 4$ in both training and evaluations.

Like for lemmas, we sample several sketch attempts per problem and apply a **mean baseline to R over N independent sketch attempts**, i.e. given $R_1, \ldots, R_N$, for $i = 1, \ldots, N$, we set the advantage

$$A_i = R_i - \frac{1}{N}\sum_{m=1}^{N} R_m.$$

Note that a single hierarchical proof generation attempt requires $nk+1$ language model calls, making $\mathrm{pass@}(nk+1)$ the compute-matched comparison.

G.1 Choices for Sketch Reward Functions

For rewarding a proof sketch with k lemma proof attempts to lemma j with successes r_i^j, $i = 1, \ldots, k$, there are various options, including the following:

- Reward as **overall proof success**: $R_o = \min_j \max_i r_i^j$, which depends on k.
- Reward as **average proof success**: $R_a = \prod_j \left(\frac{1}{k} \sum_i r_i^j \right)$, which is an unbiased estimator of $\mathbb{E}_{r_1^j \sim d}[\prod_j r_1^j]$, an average independent of k.
- Reward as a soft interpolation of overall and average proof success.

In this work, we exclusively use $R = R_o$ and $k = 4$ proof attempts per lemma, but we note that for scaling inference, larger values for k may turn out advantageous, necessitating a different reward function that does not saturate with $k \to \infty$.

H Results: Performant Initial Checkpoint

Evaluating the initial supervised finetuning checkpoint described in Section F.1, we obtain scores of 18.7% on ProofNet (valid and test) pass@32 with standard inference, of 16.8% pass@32 with sketching; of 63.5% pass@32 on miniF2F-valid with standard inference and of 50.8% pass@32 with sketching. In all of these evaluations, we couple the neural model with the Lean automation tactic described in Section 3.

These numbers come close to state-of-the-art numbers for 7B parameter models, supporting our claim that the SFT model can be used as a performant initial checkpoint to study saturation in the subsequent reinforcement learning stage.

I Results: Pleateaus and Sketch-based Proving

I.1 Saturation with GRPO variants

Standard reinforcement learning with GRPO plateaus early in our RL setup: as can be seen in Figure 1, the GRPO baseline quickly reaches a ceiling after which few additional problems from the training dataset are solved, while entropy decays. Evaluation scores peak early in training and then subsequently decline. This cannot be remedied with the standard mitigations described in Section 2.2: increasing the dataset size ten-fold and switching to pass@k training both show the same behavior on evaluations and similar saturation slopes in cumulative pass rate. Pass@k training maintains a higher entropy, but this does not translate to higher cumulative solve rates. (Note that cumulative solve rates for the larger dataset are not comparable with the ones on the default dataset, but the saturation behavior is the same.)

I.2 Sketch-Based Proving

For sketch-based proving, we obtain promising for its utility in theorem proving in Lean: According to Figure 1, sketch-based proving reaches **higher cumulative solve rates** than any of the baseline methods, attesting continued exploration and suggesting usefulness in large-scale reinforcment learning setups beyond the scale tested here. **Entropy is maintained higher** than for both of the GRPO runs, coming close to the entropy of pass@k training. In terms of evaluation results, performance starts lower since the initial model was hardly trained on this inference scheme, but mostly catches up with the baselines over the course of training.

Analyzing the scaling behavior at inference time across training steps in Figure 2, we find that standard inference is hardly optimized further over the course of training while sketch-based proving continues to improve from its low starting point.

J Autoformalization Inference Scaling on miniF2F-test

We provide the analogon to Figure 2 for miniF2F-test in Figure 3.

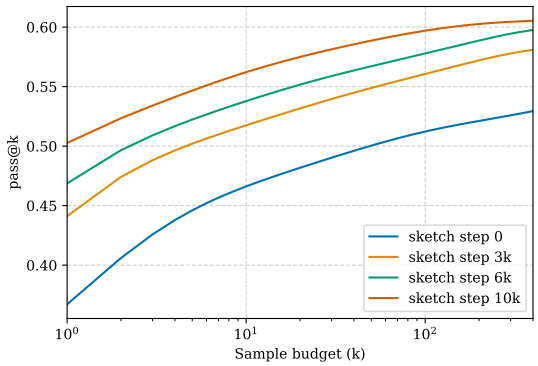


Figure 3: **Autoformalization inference scaling on miniF2F-test with sketch-based inference.** We use ground truth proofs and generated proofs, and pass@k is evaluated over k attempts that may stem from different proof sources.