# Mining Machine-Generated Lean Proofs: Advice for Users, Developers, and Testers

Ilija Ivanov, Jiayi Wu, Gavin Zhao, Zekai Li,
Stephen H. Bach, and Robert Y. Lewis*

Brown University, Providence, RI, U.S.A.

**Introduction.** In recent years the proof assistant community has been introduced to a torrent of LLM-based *autoformalization* tools, meant broadly to encompass all systems whose outputs include formal proofs. These tools vary in their interfaces and architectures. Some expect to be given the formal *statement* of a theorem and to output a putative *proof* [9, 10, 3, 18, 19, 13]. Others may take as input an informal theorem statement [18, 13] or an informal proof sketch [6, 2, 16], or output multiple declarations or files [1, 5, 4, 11]. Some of these tools generate full proofs at once [18, 9, 15], while others are based on tree search methods [17, 8, 20, 21]; some have an active line-by-line verifier in the loop [7], while others depend on natural language chain-of-thought for test-time reasoning.

Given this proliferation of forms, directly comparing autoformalization tools in terms of proving ability can be an exercise in frustration. We find it more productive to compare along more *qualitative* dimensions, looking for insights that inform those interested in developing, using, and evaluating such tools.

All of our experiments are conducted in Lean [12] with Mathlib [14]. We evaluate six publicly available prover models: DeepSeek-Prover V2 671B [13], Goedel-Prover V2 32B and 8B [10], BFS-Prover V2 32B [21], DeepSeek-Prover V1.5-RL [19], and Kimina-Prover-Distill 8B [15]. We use FormalMATH-Lite [22] as the benchmark suite: each problem supplies a Lean theorem statement, and the prover is asked to complete the proof. We then extract candidate Lean code from the model output and run it through Lean to separate accepted proofs from failures, keeping both. (The developers of FormalMATH Benchmark shared with us the output of DeepSeek Prover V2 671B.) When a model provider supplies a recommended prompt, we prefer that setting; otherwise, we use a simple completion-style prompt.

Since we are interested in a qualitative analysis of proof attempts and failure cases rather than performance analysis, we do not attempt to normalize resources across models.

**Tactic Frequency.** We extracted and counted tactic usage in both successful and failed proof attempts across all six models. The resulting distributions are model-dependent: different architectures and training strategies produce significantly different tactic usage patterns. DeepSeek Prover V2 uses `have` much more than other models. It aligns with its recursive training pipeline, which decomposes theorems into sequences of subgoals (`have` statements with `sorry` placeholders) and then solves them recursively [13]. We also notice that DeepSeek frequently uses `try`, which makes sense as a result of RL because `try` allows for more tolerance in proof errors.

The resulting distributions are also highly benchmark dependent. The data that we received for DeepSeek Prover V2 671B is run on the full FormalMATH benchmark, which contains a substantially higher fraction of competition-math problems than the FormalMATH-Lite benchmark. As a result, tactics such as `norm_num` account for a much larger share of tactic calls.

*This research was supported by the Renaissance Philanthropy AI for Math Fund.

We also note differences in tactic frequencies between *correct* and *incorrect* proofs generated by the same model, suggesting that certain models are "bad at" using certain tactics. For instance, while DeepSeek's training leans it heavily toward the "conjecturing" method, `have` shows up even more disproportionately in incorrect proofs.

Among the most common $n$-grams of tactic sequences are various permutations of normalization tactics like `ring_nf`, `field_simp`, and `norm_num`. This pattern and others in the $n$-grams suggest possible *extensions* to the tactic language.

Of course, adding new tactics poses difficulties to pretrained models. Notably absent from most provers' output was `grind`, a powerful tool introduced in Lean 4.22.0. Mere prompting does not seem to be effective at teaching a new tactic to a model; more sophisticated methods are needed.

**Error Message Analysis.** We analyzed error message frequencies in generated proofs that failed to type check. The most common error messages across all models are tied to unexpected tokens, unknown identifiers, unsolved goals, and `linarith` misuse.

Many "unexpected token" error messages come from the model using syntax that is not imported or in scope. Lean's flexible scoped syntax is very powerful, but it is impossible to fully qualify inline in a proof. This tells us that it is important for the model to be able to modify the frontend context (e.g. by opening namespaces or adding imports).

Models frequently misapplied "high-power" tactics like `linarith`. (We see this also in successful proofs, where these tactics are used in lieu of more appropriate options.) We suspect that models learn to use these tools as cudgels and expect them to work more often than they do. We suggest exploring whether providing models with examples of *incorrect* applications of tactics and subsequently corrected versions could help alleviate this issue.

**Evaluation Difficulties.** Most if not all models are trained against a specific Lean version. For example, DeepSeek Prover V2 was trained against v4.9. This causes a lot of trouble because models are not explicitly aware of Lean versions, and end users often work on the latest version of Mathlib. Empirically, while evaluating DeepSeek Prover V2's (trained against v4.9) performance on the latest Lean version (v4.29.1), we've found that models trained against an older Lean version commonly fail on two types of errors: "unknown identifier," meaning changed Mathlib4 definitions/theorems/APIs, and state confusion due to under- or overestimating blackbox-ish tactics like `simp`, `aesop`, or `linarith`. The latter is usually due to tactic implementation changes and/or annotation changes that remove or add a theorem to a tactic's "internal knowledge." "No goals to be solved" is a very common message that appears.

These errors are generally not hard to fix, since the logic of the proof still holds. In particular the latter kind of error is a sign that the proof has gotten "easier." We recognize that training a Lean version-aware model could be very difficult; we think that specialized tooling (language model based or symbolic) for doing Lean/Mathlib version migrations might be useful to bridge this gap. Simple postprocessing on proof output is likely to resolve many of these errors.

Some models are sensitive to the exact prompt format used at evaluation time. This includes both the natural-language instruction and whether the model's chat template is applied. These choices are easy to treat as harmless pipeline details, but they can change the measured success rate enough to affect conclusions about a model. Model providers should publish the exact prompt and chat-template policy used for their evaluations and (if applicable) fine-tuning. Benchmark maintainers should either adopt those settings or explicitly report deviations, so that prompt-format differences do not become hidden sources of inconsistent evaluation.

# References


[1] Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025.

[2] Rafael Cabral, Tuan Manh Do, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Xin Shen. Proofflow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025.

[3] 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.

[4] Guoxiong Gao, Bin Wu, Zeming Sun, Jiedong Jiang, Wanyi He, Zichen Wang, Yutong Wang, Peihao Wu, and Bin Dong. Archon: An agentic system for formalizing research-level mathematics. https://github.com/frenzymath/Archon, 2026. Software repository and technical blog post. Accessed: 2026-05-10.

[5] Fabian Gloeckle, Ahmad Rammal, Charles Arnal, Remi Munos, Vivien Cabannes, Gabriel Synnaeve, and Amaury Hayat. Automatic textbook formalization, 2026.

[6] Prithwish Jana, Kaan Kale, Ahmet Ege Tanriverdi, Cruise Song, Sriram Vishwanath, and Vijay Ganesh. Proofbridge: Auto-formalization of natural language proofs in lean via joint embeddings. *arXiv preprint arXiv:2510.15681*, 2025.

[7] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[8] 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.

[9] 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.

[10] 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.

[11] Math, Inc. Opengauss: An open source autoformalization harness for lean. https://github.com/math-inc/OpenGauss, 2026. Software repository. Accessed: 2026-05-10.

[12] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.

[13] 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.

[14] The mathlib community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 367–381, 2020.

[15] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu,

Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025.

[16] Ziyu Wang, Bowen Yang, Chenyi Li, Yuan Zhang, Shihao Zhou, Bin Dong, and Zaiwen Wen. Translating informal proofs into formal proofs using a chain of states. *arXiv preprint arXiv:2512.10317*, 2025.

[17] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Zheng Yuan, Wenwei Zhang, Dahua Lin, and Kai Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via critic-guided search. *arXiv preprint arXiv:2410.15700*, 2024.

[18] 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.

[19] 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. In *International Conference on Learning Representations*, volume 2025, pages 72274–72303, 2025.

[20] Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32588–32599, 2025.

[21] Ran Xin, Zeyu Zheng, Yanchen Nie, Kun Yuan, and Xia Xiao. Scaling up multi-turn off-policy rl and multi-agent tree search for llm step-provers. *arXiv preprint arXiv:2509.06493*, 2025.

[22] Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. Formalmath: Benchmarking formal mathematical reasoning of large language models, 2025.