From Prompts to Protocols: `lean4-skills` for AI-Assisted Lean Formalization

Cameron Freer

Massachusetts Institute of Technology, Cambridge, MA, USA
freer@mit.edu

Abstract

The `lean4-skills` package provides a reusable protocol layer for AI-assisted Lean development, implemented as a coding agent skill that aims to make agents easier to steer on large formalizations. This protocol defines the operational rules by which a human mathematician, an LLM coding agent, Lean, and Mathlib interact: when to search the library, inspect goals, attempt proofs, review, replan, checkpoint, or return control to the user. The main failures of coding agents in this setting are often operational rather than purely mathematical: agents reprove library lemmas, miss diagnostics, change target statements, add brittle adapters, or pursue stalled paths. The package equips a coding agent with countermeasures (library search, Lean feedback, bounded attempts, coherent checkpoints, review for drift, and explicit stop/replan decisions) alongside reference material on Lean and Mathlib conventions and a guide to the Lean LSP MCP server (goal queries, diagnostics, and Mathlib search).

1 Protocol Design for Lean Coding Agents

Large-scale formalization needs more than a strong model. It calls for a protocol that specifies when to inspect goals, search Mathlib, split a lemma, build, review, checkpoint, simplify, or stop. Prompts shape responses but get diluted across long sessions and many tool calls; a protocol gives repeatable structure to the interaction among the model, the Lean compiler, search tools, review, and the human supervisor. The `lean4-skills`[1] package [15] provides this protocol layer for coding agents such as Claude Code and Codex.

The package, implemented primarily as a coding-agent skill, gives an agent four kinds of support: reference material; commands, subagents, hooks, and scripts; a bounded proof loop; and guidance for using the Lean LSP MCP server. The references record recurring conventions and failure modes: search the library before attempting a proof, introduce no new axioms, leave target statements unchanged without human approval, build proofs incrementally, follow Mathlib idioms, and manage typeclass instances. The commands and subagents cover drafting, proving, reviewing, refactoring, golfing, checkpointing, and learning. A learning mode lets even informal explanations stay grounded: the agent checks claims against Mathlib and the project's Lean code.

The bounded loop alternates phases: planning (set goals, gather context), Lean-checked work (search, attempt proofs, test tactics), checkpointing (build, axiom- and `sorry`-checks), review (classify the obstruction), and replanning (split, refactor, or escalate). Figure 1 shows the loop together with the other package components. In guided modes, the loop creates natural points for human review; in autonomous modes, the same loop supplies budgets, checkpoints, and stop conditions. The Lean LSP MCP server [16] exposes the Lean language server and Mathlib search tools to LLM agents over the Model Context Protocol, giving the agent feedback analogous to what humans see in Lean's InfoView; the package's guide explains how to

[1] Project repository: https://github.com/cameronfreer/lean4-skills.

Component	Contents (partial list)
Reference material	conventions, examples, failure modes, Mathlib idioms, typeclass management
Commands	`/draft`, `/prove`, `/autoprove`, `/review`, `/golf`, `/checkpoint`, `/refactor`, `/learn`
Subagents	`axiom-eliminator`, `proof-golfer`, `proof-repair`, `sorry-filler-deep`
Scripts	`search_mathlib`, `sorry_analyzer`, `find_instances`, `minimize_imports`
Lean LSP MCP guide	goals, diagnostics, hover text, completions, tactic tests, LeanSearch, Loogle, LeanFinder
Checks and handoff	builds, axiom- and `sorry`-checks, import minimization

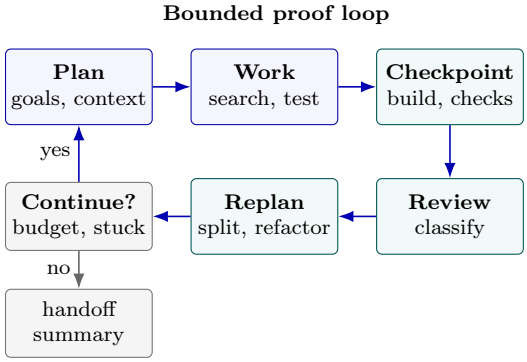

Figure 1: Components and bounded proof loop of `lean4-skills`.

query goals, diagnostics, hover text, and Mathlib search, alongside builds, axiom- and `sorry`-checks. The protocol scaffolds operations, not mathematics: a wrong target statement or a mathematical gap still leans on the model's own reasoning or on direction from the user.

Related work. LeanDojo and ReProver [1] provide dataset, retrieval, benchmark, and toolkit infrastructure. LLMSTEP [2] and Lean Copilot [3] integrate proof suggestions into Lean; COPRA [4], Pantograph [5], CoqPilot [6], Agentic Proof Automation [7], and Numina-Lean-Agent [8] explore feedback-driven or coding-agent approaches. DeepSeek-Prover [9], Goedel-Prover [10], Kimina-Prover [11], and Aristotle [12] push benchmark performance, while SWE-agent [13] and OpenHands [14] study coding-agent interfaces. In contrast to some of these approaches, `lean4-skills` targets the complementary protocol between a coding agent and Lean, rather than proposing a new prover model or benchmark search algorithm.

2 In Practice

The `lean4-skills` package grew out of an exchangeability project [17, 18] in which the author formalized three different proofs of de Finetti's theorem [21]. As the project ran into recurring problems with conditional expectation, sub-σ-algebra bookkeeping, elaboration slowdowns, Mathlib discovery, and statement drift, those lessons were recorded in the skill so that later sessions could start from what earlier sessions had learned.

The author has since used the same skill on follow-up projects on graphons [19] and infinitary logic [20]. Other groups have begun using it, spanning mathematics, physics, and program verification: Numina-Lean-Agent [22] installs it; Ramanujan–Nagell [23] and QFT [24] Lean formalizations use it; and an independent benchmark study [25] places "Claude Code + lean4-skills" among the state-of-the-art on the CLEVER [26] and VERINA [27] program-verification benchmarks, ahead of Aristotle [12].

What `lean4-skills` offers is therefore not just a prompt file, but a protocol for keeping a coding agent, Lean, Mathlib, and a human supervisor in a checked feedback loop. Such protocols can serve as shared infrastructure for the Lean community, beyond project-specific prompting notes.

References

[1] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. *LeanDojo: Theorem Proving with Retrieval-Augmented Language Models*. NeurIPS, 2023. arXiv:2306.15626.

[2] Sean Welleck and Rahul Saha. *LLMSTEP: LLM Proofstep Suggestions in Lean*. arXiv:2310.18457, 2023.

[3] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. *Lean Copilot: Large Language Models as Copilots for Theorem Proving in Lean*. Proceedings of the International Conference on Neuro-symbolic Systems, PMLR 288:144–169, 2025.

[4] Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. *An In-Context Learning Agent for Formal Theorem-Proving*. COLM, 2024. arXiv:2310.04353.

[5] 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*. arXiv:2410.16429, 2024.

[6] Andrei Kozyrev, Gleb Solovev, Nikita Khramov, and Anton Podkopaev. *CoqPilot, a Plugin for LLM-Based Generation of Proofs*. ASE, 2024. doi:10.1145/3691620.3695357.

[7] Yichen Xu and Martin Odersky. *Agentic Proof Automation: A Case Study*. arXiv:2601.03768, 2026.

[8] Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, Lihong Zhi, Jia Li, and Wenda Li. *Numina-Lean-Agent: An Open and General Agentic Reasoning System for Formal Mathematics*. arXiv:2601.14027, 2026.

[9] Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. *DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search*. arXiv:2408.08152, 2024.

[10] Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. *Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving*. COLM, 2025. arXiv:2502.07640.

[11] 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 Saxce, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Leo 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*. arXiv:2504.11354, 2025.

[12] 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*. arXiv:2510.01346, 2025.

[13] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. *SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering*. NeurIPS, 2024. arXiv:2405.15793.

[14] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. *OpenHands: An Open Platform for AI Software Developers as Generalist Agents*. ICLR, 2025. arXiv:2407.16741.

[15] Cameron Freer. *Lean 4 Skills: Theorem Proving Skill and Workflow Pack for AI Coding Agents*. GitHub repository, 2025. https://github.com/cameronfreer/lean4-skills.

[16] Oliver Dressler. *Lean Theorem Prover MCP*. GitHub repository, 2025. https://github.com/oOo0oOo/lean-lsp-mcp.

[17] Cameron Freer. *Exchangeability and de Finetti's Theorem in Lean 4*. GitHub repository, 2026. https://github.com/cameronfreer/exchangeability.

[18] Cameron Freer. *Three Roads to de Finetti's Theorem in Lean 4*. Short paper conditionally accepted at ITP, 2026.

[19] Cameron Freer. *Graphon Theory in Lean 4*. GitHub repository, 2026. https://github.com/cameronfreer/graphon.

[20] Cameron Freer. *Infinitary Logic in Lean 4*. GitHub repository, 2026. https://github.com/cameronfreer/infinitary-logic.

[21] Olav Kallenberg. *Probabilistic Symmetries and Invariance Principles*. Springer, 2005. doi:10.1007/0-387-28861-9.

[22] Project Numina. *Numina-Lean-Agent*. GitHub repository, 2026. https://github.com/project-numina/numina-lean-agent.

[23] Barinder S. Banwait. *A Formal Proof of the Ramanujan–Nagell Theorem in Lean 4*. arXiv:2604.09808, 2026.

[24] Michael R. Douglas, Sarah Hoback, Anna Mei, and Ron Nissim. *Formalization of QFT*. arXiv:2603.15770, 2026.

[25] Alessandro Sosso, Akhil Arora, and Bas Spitters. *Agentic Proving for Program Verification*. ICLR Workshop on LLM Reasoning, 2026.

[26] Amitayush Thakur, Jasper Lee, George Tsoukalas, Meghana Sistla, Matthew Zhao, Stefan Zetzsche, Greg Durrett, Yisong Yue, and Swarat Chaudhuri. *CLEVER: A Curated Benchmark for Formally Verified Code Generation*. NeurIPS, 2025. arXiv:2505.13938.

[27] Zhe Ye, Zhengxu Yan, Jingxuan He, Timothe Kasriel, Kaiyu Yang, and Dawn Song. *VERINA: Benchmarking Verifiable Code Generation*. arXiv:2505.23135, 2025.