AUTOFORMBENCH: A LEAN AUTOFORMALIZATION BENCHMARK FOR ADVANCED MATHEMATICS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI systems are increasingly capable of producing machine-checked Lean proofs of advanced mathematical results. Real mathematical formalization, however, is not isolated theorem proving: it requires building the definitions and intermediate lemmas that connect a result to a formal library. Existing evaluations often provide isolated theorem statements in settings where much of this supporting theory is already available in Mathlib. We introduce AutoformBench, a benchmark of 341 Lean 4 formalization tasks drawn from 24 graduate and research-level mathematical sources. Lean-experienced mathematicians formalized and audited each target statement, leaving the corresponding Lean proof to the evaluated agent. Each problem is embedded in a broader mathematical development: agents receive the informal proof but must translate it into Lean while constructing the missing definitions, intermediate lemmas, and supporting infrastructure. We compare the performance of frontier models on these problems. We find a largely shared difficulty frontier: success depends less on mathematical domain than on models' ability to construct missing formal infrastructure.

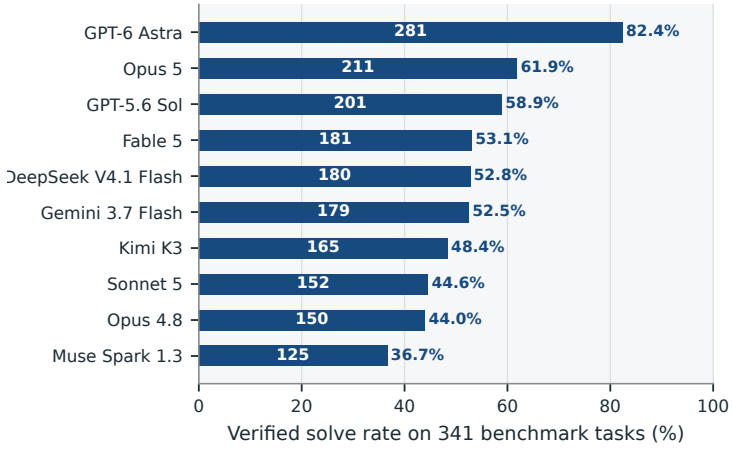


Figure 1: Verified solve rates on 341 audited benchmark tasks using the reasoning-effort setting selected for each model's headline evaluation.

1 INTRODUCTION

As language models produce increasingly sophisticated mathematical arguments, verifying their output becomes a central bottleneck. Without reliable verification, scaling mathematical generation risks producing a growing body of plausible but incorrect results that is increasingly difficult to audit. Proof assistants, such as Lean, offer a way to make verification mechanical. In Lean 4, a small kernel checks every logical step, so an accepted proof provides a precise certificate relative to its formal statement and assumptions (de Moura & Ullrich, 2021).

Producing that certificate for advanced mathematics is not merely a matter of finding the final proof. Formalization is cumulative: before proving a theorem, one must identify or construct the definitions,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

intermediate lemmas, and connections to Mathlib on which it depends (The mathlib Community, 2020).

This cumulative work is underrepresented in current evaluations. Neural theorem proving has progressed from language-model proof search (Polu & Sutskever, 2020; Lample et al., 2022) to systems capable of solving Olympiad and research-level problems (Hubert et al., 2026; Achim et al., 2025; Chen et al., 2026), but benchmarks such as miniF2F, PutnamBench, and ProofNet primarily test self-contained problems from mathematical competitions or undergraduate courses whose formal vocabulary is already supplied (Zheng et al., 2022; Tsoukalas et al., 2024; Azerbayev et al., 2023). Success on these tasks does not tell us whether an agent can navigate a new mathematical development, build its missing infrastructure, or do so efficiently.

The Jacobian Challenge (Buzzard, 2026) points toward this broader regime: its compact statements sit atop definitions and results that demand substantial formal development. Our proposed benchmark, AutoformBench, asks how far current agents can operate in this regime across mathematical domains. Each of its 341 problems provides a human-audited Lean statement, its local formal context, and the corresponding excerpt from one of 24 graduate or research-level textbooks. The agent must prove the target by interacting agentically with the provided material and with mathlib and by constructing any missing auxiliary definitions and lemmas. This makes AutoformBench a challenging test of long-horizon repository-level formalization - even the most state-of-the-art models require dozens of millions of tokens to complete 50% of the tasks, highlighting the need for much more cost-efficient approaches before autoformalization can effectively scale to the mathematical literature and new research production.

Contributions.

- A benchmark of 341 audited Lean 4 proof-completion problems drawn from 24 graduate and research-level textbooks across diverse mathematical fields.
- An agentic evaluation environment powered by state-of-the-art Lean tools for reliable and convenient model testing.
- A controlled evaluation of ten frontier models, together with analyses of efficiency, generated infrastructure, and the boundary between solved and unsolved problems.

2 Related Work

Benchmarks for formal mathematics. miniF2F and PutnamBench evaluate proofs of fixed formal statements drawn from high-school and undergraduate competitions (Zheng et al., 2022; Tsoukalas et al., 2024), while FormalProofBench pairs advanced undergraduate and graduate natural-language problems with Lean statements and evaluates proof production in a tool-enabled agent loop (Ravi et al., 2026). ProofNet, Lean Workbook, FormL4, NuminaMath-LEAN, and FormalMATH scale the construction or evaluation of paired informal and formal problems (Azerbayev et al., 2023; Ying et al., 2024; Lu et al., 2024; AI-MO, 2025; Yu et al., 2025). These collections primarily consist of independent theorem-level items drawn from competitions, educational material, or existing formal libraries. FATE instead focuses on algebra at increasing levels of difficulty: FATE-M contains textbook exercises, while FATE-X reaches beyond PhD qualifying exams and Mathlib coverage (Jiang et al., 2026). Each file centers on one final theorem with one proof hole; 38% of FATE-X tasks additionally introduce new definitions, averaging 2.4 among those tasks. SorryDB (Letson et al., 2026) moves toward realistic library context by extracting open proof obligations from Lean repositories. FormalQualBench contains 23 graduate-level theorem statements formalized in Lean and verified by a Lean expert for evaluating practical formalization workflows (Math Inc., 2026), while Formal Conjectures provides an evolving collection of solved and open research-level problems formalized in Lean (Firsching et al., 2026).

Research-level autoformalization. Autoformalization studies the translation of informal mathematics into formal language (Wu et al., 2022). Recent work has begun to formalize sustained mathematical developments: general topology in Megalodon (Urban, 2026), real- and convex-analysis textbook material (Wang et al., 2026), and a graduate textbook using a largely autonomous multi-agent workflow (Gloeckle et al., 2026). AutoformBot extends this approach across 26 textbooks using a shared multi-agent pipeline (Rammal et al., 2026). More recent systems study document-to-

project workflows for mathematical papers, long-horizon research developments, and mathematical discovery (Milikic et al., 2026; Zhang et al., 2026; Nie et al., 2026; Moakhar et al., 2026; Patel et al., 2026). Other projects formalize landmark research results, including sphere packing in dimension eight and De Giorgi–Nash–Moser theory (Hariharan et al., 2026; Armstrong & Kempe, 2026), and the formalisation of the proof of Fermat's Last Theorem by Anthropic (Anthropic, 2026e). The Jacobian Challenge (Buzzard, 2026) similarly highlights the substantial infrastructure needed for a compact advanced mathematical development.

AutoformBench complements these efforts by evaluating proof completion across many graduate-level textbook-derived developments, with human-audited statements, access to the informal source, and independent clean-room verification. It is designed to measure a model's capability to interact with a body of mathematics, testing the skills required to formalize a large body of advanced mathematics - to our knowledge the first large-scale benchmark to do so.

3 Benchmark Construction

AutoformBench consists of challenging statements drawn from graduate and research-level textbooks spanning diverse mathematical domains and formalized by human experts.

Source material. We selected 24 open-source graduate and PhD-level textbooks from a variety of domains representative of the diversity of modern mathematics, ranging from algebraic geometry to the theory of probability. Most of these come from the excellent MIT OpenCourseWare selection of open courses (MIT, 2026). The full list of selected textbooks and their provenance can be found in Table 11 in the Appendix.

Within each book, we then selected statements with challenging proofs following a balanced distribution across chapters and subtopics, so as to maximize the diversity of the resulting tasks. We deliberately selected material beyond direct Mathlib (The mathlib Community, 2020) coverage, so that success requires constructing missing formal infrastructure rather than retrieving an existing theorem, while retaining enough nearby foundations for meaningful progress within a bounded agent run. Appendix G quantifies proximity to the pinned Mathlib revision.

Formalization protocol Human experts (at least PhD-level) from the annotation company Labelbox (Labelbox, 2026) were tasked with formalizing the selected theorems, as well as all the definitions and lemmas needed to state them, in Lean 4 (de Moura & Ullrich, 2021) (version 4.32). Each resulting formalization was then reviewed by an independent human expert, as well as by the authors assisted by frontier models.

Output This process resulted in 341 self-contained Lean workspaces containing 1,155 graded proof obligations. Each workspace has one headline target; one or more proposition-valued proof bodies are replaced by the Lean keyword `sorry`. At evaluation time, the tested model must replace every graded `sorry` with a correct Lean proof and may introduce additional definitions and helper lemmas. A task counts as solved only when the Lean kernel accepts all of its graded obligations. We release all 341 tasks in Harbor format. Appendix A gives the complete distribution of obligations and characterizes the formal context visible in the target statements.

4 Evaluation Protocol

Workspace and verification. For each problem, we construct a self-contained Lean workspace around the audited target statement. It contains a frozen file containing the statement and its supporting definitions, together with an editable copy of this file. The model must complete the statement's proof in the editable file and may add further definitions and lemmas. A machine-readable specification identifies the results to grade and the foundational axioms that may be used. The workspace also contains the corresponding textbook statement, surrounding definitions, and informal proof. We execute these workspaces using Harbor (Merrill et al., 2026; Harbor Framework Team, 2026), a general evaluation harness that standardizes how agent systems receive tasks and how their outputs are graded. Harbor runs the agent in an isolated Docker environment and invokes the grader in a separate, network-disabled container. When the run ends, only the editable solution and its auxiliary

Lean files are transferred to the verifier; all other agent-side changes are discarded. The verifier inserts those files into a fresh copy of the original workspace and runs COMPARATOR (Lean FRO, 2025). Comparator checks that the submitted results prove exactly the original statements, do not alter definitions fixed by the benchmark, use only permitted axioms, and are accepted by Lean. Successful proofs are additionally replayed by Nanoda (Nanoda contributors, 2026), an independent proof-checking kernel. This strict protocol is needed to prevent models from cheating by leveraging their knowledge of Lean syntax, e.g. by using special keywords to trivialize the task by redefining key mathematical objects. Appendix B describes the workspace, budgets, tools, isolation procedure, and Harbor packaging in detail.

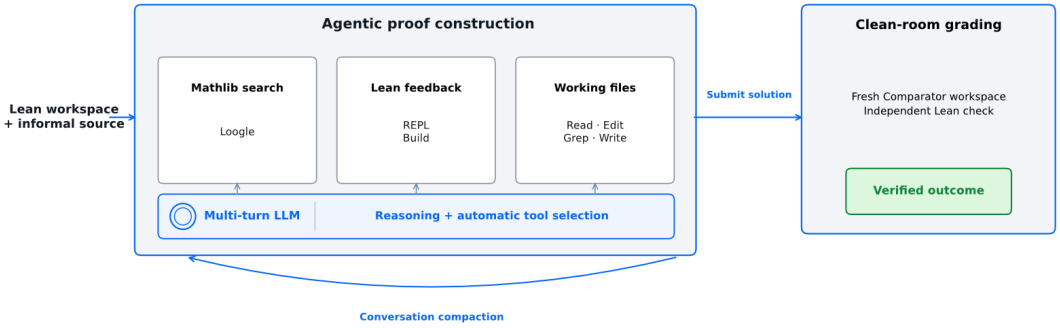


Figure 2: AutoformBench evaluation protocol. A multi-turn model edits an isolated Lean workspace using library search, executable Lean feedback, and file tools. Submitted solutions are independently checked in a fresh Comparator workspace; long runs continue through conversation compaction.

Agentic harness. Advanced formalization naturally requires an agentic workflow: the solver must navigate an unfamiliar development, search for existing library results, experiment with candidate proof fragments, interpret compiler feedback, and iteratively extend its solution. We therefore use a lightweight multi-turn harness. As shown in Figure 2, we provide three types of tools:

- **Filesystem tools** for navigating the workspace and editing submission files.
- **Lean-execution tools** that allow the model to test partial or complete candidate solutions against the Lean kernel and check their correctness.
- **Mathlib search** through Loogle (Breitner, 2023), for finding declarations in the pinned Mathlib library.

Tool execution is network-disabled, so the model cannot retrieve external solutions or documentation during a run.

The workspace serves as durable state throughout the trajectory. When the conversation approaches the model's context limit, the harness compacts the interaction history into a summary while preserving the files and running tool environment. This allows the model to continue working across multiple context windows and enables test-time scaling through longer trajectories. Each common-harness run has a flat provider-cost cap of $200 per problem. We derive reported dollar costs from provider-reported token counts using the public API prices for each model at the time of evaluation. A run terminates when the model calls `submit_report` or exhausts its budget.

5 MAIN RESULTS

We evaluate OpenAI's GPT-6 Astra and GPT-5.6 Sol (OpenAI, 2026b;a); Anthropic's Claude Opus 5, Claude Opus 4.8, Claude Sonnet 5, and Claude Fable 5 (Fable 5 below) (Anthropic, 2026c;b;d;a); Google DeepMind's Gemini 3.7 Flash (Google DeepMind, 2026); and DeepSeek V4.1 Flash, Kimi K3, and Muse Spark 1.3 (DeepSeek-AI, 2026; Kimi Team et al., 2026; Meta Superintelligence Labs, 2026). The headline evaluation uses `xhigh` reasoning effort for GPT-5.6 Sol and Muse Spark 1.3; `high` for Gemini 3.7 Flash, Claude Sonnet 5, and Claude Opus 4.8; and `max` for Claude Opus 5, Fable 5, DeepSeek V4.1 Flash, Kimi K3, and GPT-6 Astra. A turn is one model response plus its tool-call batch. Figure 1 summarizes their verified performance with the $200 budget cap. Appendix K.2 examines sensitivity to lower effort settings, while Appendix C gives the full cost breakdown.

Note that the $200 per-task budget used for the headline scores is a ceiling rather than a requirement for evaluating AutoformBench. Evaluators can choose a smaller fixed budget and still obtain a directly comparable leaderboard. We believe that this is an important point: cost efficiency is essential for the long-term impact of autoformalization, and formalizing the mathematical literature at well as the production of new research at scale will require models and inference harnesses that achieve strong performance at low computational cost. As such, AutoformBench can provide a natural benchmark for measuring progress under explicit cost constraints.

Budget-conditioned leaderboard. The tasks evaluated are extremely challenging, and most models require a very long context to solve them. Moreover, models can test their candidate solutions using Lean tools, which allows them to keep iterating until a correct solution is found. As such, the most informative metric is not the absolute success rate of a given model, but rather the number of problems solved given a certain budget (of tokens, turns, etc.).

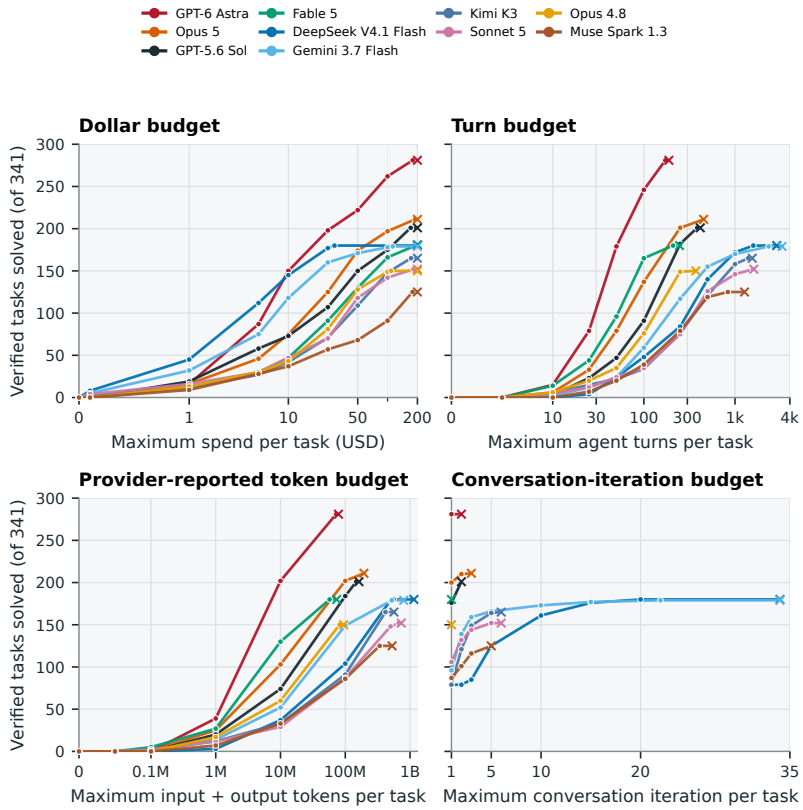


Figure 3: Budget-conditioned leaderboard on the 341 tasks. Each curve gives the number of verified tasks completed within a fixed per-task budget of dollars, agent turns, provider-reported input plus output tokens, or conversation iterations (each context compaction starts a new iteration). DeepSeek V4.1 Flash leads below approximately $9 per task and GPT-6 Astra thereafter. Astra is also highest from 10 turns, approximately 0.41 million total tokens, and the first conversation iteration onward.

A verified task enters the curve once its recorded run reaches that resource level; failed tasks never enter. Figure 3 reports this score for four per-task resource limits. At any fixed cutoff, the highest curve identifies the model with the greatest observed verified coverage under that constraint. Appendix D provides additional budget-conditioned views. We observe that our set of tasks is particularly challenging, with most frontier models requiring hundred of millions of read and written tokens to reach a 50% completion rate. A striking finding is that the efficiency frontier is dominated by two models embodying radically opposed design philosophies: Astra, a superlative frontier model which trades a high cost per token for a very high token-efficiency, and DeepSeek V4.1 Flash, an inference-optimized model with a very low cost per token.

Efficiency comparisons. Because models solve different task sets, we also compare them on the 114 tasks solved by all ten models. For each task and resource, we divide each model's value by the median across models before summarizing the resulting ratios in Figure 4. DeepSeek has the lowest median recorded cost ($0.16\times$ the same-task median), whereas Astra uses the fewest turns ($0.27\times$) and produces the most concise answers ($0.57\times$). The separation between these dimensions shows that monetary cost, interaction length, and implementation size capture distinct aspects of agent efficiency. Appendix E gives the task-level distributions and taskwise-best comparison for this shared 114-task cohort.

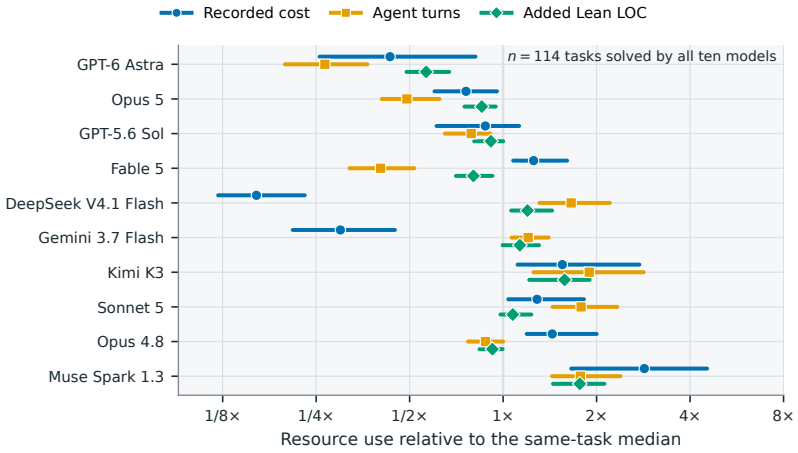


Figure 4: Compared efficiency on the 114 tasks solved by all ten models. For each task, cost, turns, and added Lean LOC are normalized by the median across models. Points show each model's median ratio and horizontal bars show the interquartile range.

What makes a problem difficult? The ten models evaluated exhibit a strongly stratified difficulty frontier. Across all ten models, 114 tasks are solved by every model and 59 by none; 168 produce disagreement (Figure 15). Across all 45 possible model pairs, the higher-scoring model solves at least 95% of the lower-scoring model's successes in 38 pairs. For the median pair, only one task is solved by the lower-scoring model but not by the higher-scoring model. Astra alone solves 281 tasks and uniquely solves 66 that every other model misses; conversely, only one task is solved by another model and missed by Astra. This strong nesting indicates a largely shared difficulty frontier.

Two related measures track this difficulty. Across the ten models, mean success falls from 64.4% on single-hole tasks to 30.9% on multi-hole tasks. Figure 5 compares GPT-6 Astra's solution LOC with the number of models that solve the same task. Across Astra's 281 verified solutions, the correlation is $\rho = -0.86$: longer solutions tend to correspond to tasks solved by fewer models. By contrast, we find little evidence of specialization by mathematical field; Appendix F reports the per-textbook results.

6 Reliability and Difficulty Analysis

Agent-Built Infrastructure. Solving AutoformBench typically requires building a local formal library around the target theorem. Among the 281 accepted GPT-6 Astra solutions, 92.9% use auxiliary lemmas, 73.3% introduce new definitions, and 94.0% of dependency-reachable proof code lies in supporting declarations rather than the graded proof bodies. This remains true for tasks exposing only one proof hole: 90.4% use auxiliary lemmas, and supporting declarations account for 93.4% of live proof code. A single-hole task is therefore not necessarily an isolated theorem-proving problem. Appendix G gives the complete methodology and distributions.

Run-to-Run Stability. Because AutoformBench permits long and expensive agent trajectories, the full-benchmark leaderboard reports one scored trajectory per model and problem (pass@1). To test whether these results are sensitive to run-to-run variation, we perform a smaller repeatability study: GPT-5.6 Sol and Gemini 3.7 Flash are each run three times on a fixed, stratified panel of 72 problems

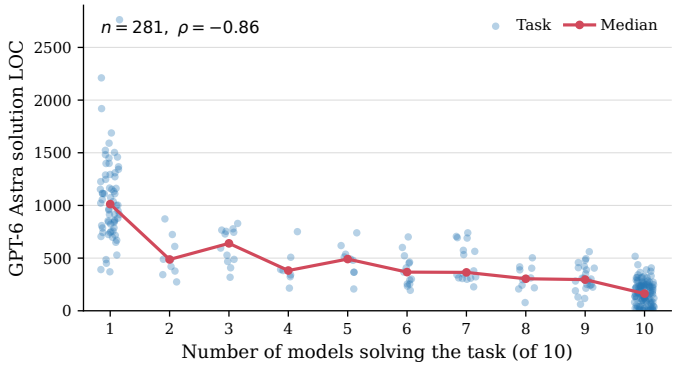


Figure 5: GPT-6 Astra solution LOC versus the number of models solving each task, restricted to Astra's 281 verified successes. Each blue point is one task; the red line connects the median solution LOC at each solver count.

spanning all 24 textbooks. Appendix H gives the complete panel construction and repeated-run protocol. Figure 6 shows both quantities as the available turn, cost, and token budgets increase.

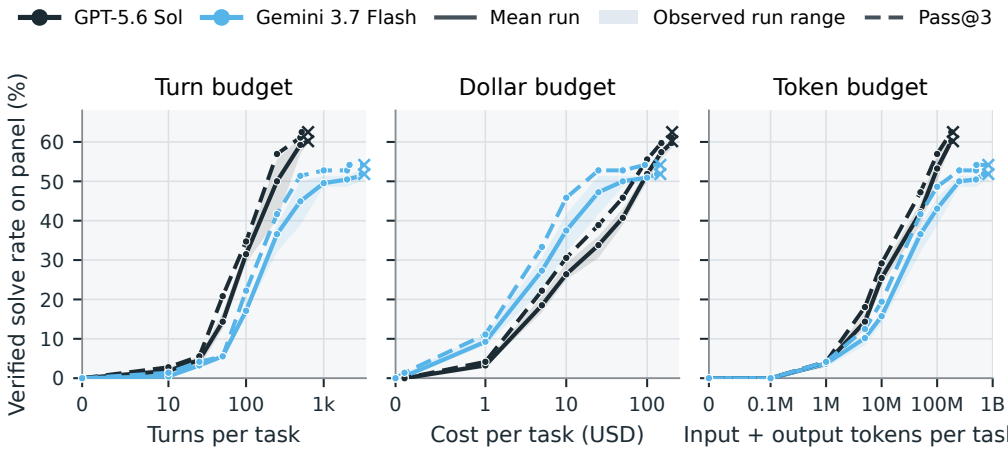


Figure 6: Verified coverage on the 72-problem panel as turn, recorded-cost, and provider-reported input-plus-output-token budgets increase.

Outcomes are highly stable. For each model, 69/72 problems receive the same result in all three runs. Sol solves 43–44 problems per run and 45 under pass@3; Gemini solves 36–38 per run and 39 under pass@3. Thus pass@3 adds only one solution beyond the best individual run for either model.

7 Ablations

Search and Executable Feedback To measure the effect of tool access, we first remove the agent loop entirely. On the 230 workspaces containing exactly one proof hole, GPT-5.6 Sol and Gemini 3.7 Flash receive the complete problem-local context and informal source in a single request, but cannot use tools, receive compiler feedback, or revise their answer. We use the headline model settings and the same clean-room grader. They solve only 6/230 (2.6%) and 3/230 (1.3%) problems, respectively, compared with 163/230 (70.9%) and 148/230 (64.3%) under the full agentic evaluation.

Condition	Gemini 3.7 Flash	GPT-5.6 Sol
Full tools, three runs	38, 36, 38	43, 44, 43
No REPL	36	45
No Loogle	35	44
No build	24	39
No REPL or build	1	4
Files only	1	1

Table 1: Verified successes under tool ablations on the fixed 72-problem panel. The full-tool row reports all three runs; other conditions use one run.

We then remove individual tools while retaining the agent loop. To control cost, these experiments reuse the fixed 72-problem panel from Section 6. We remove the REPL, Loogle, and full-project build individually and in combination; every experimental setup retains filesystem tools, report submission, the informal source, and the $200 per-problem budget. Table 1 reports the results.

Removing either the REPL or Loogle alone leaves success rates close to those observed across the three full-tool runs. Removing build has a larger effect, particularly for Gemini. Most strikingly, removing both executable Lean-feedback interfaces collapses performance from 36–45 solved problems to only 1 for Gemini and 4 for GPT-5.6 Sol. With only file operations remaining, both models solve just 1/72 problems. Thus structured Mathlib search through Loogle is not essential when filesystem search remains available, and the REPL and build seem to be to some extent substitutable. Appendix K gives the detailed tool-use and fallback analysis.

Context-Iteration Scaling We measure how much the compaction mechanism described in Section 4 contributes to task completion by comparing coverage at the end of the initial context window with final coverage after subsequent context windows. DeepSeek solves 101 additional tasks after its first context window, Kimi 86, Gemini 83, and Sonnet 46. The gains are smaller for Opus and GPT-5.6 Sol, while every successful Astra, Fable, and Opus 4.8 trajectory finishes within its initial context window.

Model	Before compaction	Final	Gain
GPT-6 Astra	281	281	0
Opus 5	200	211	11
Fable 5	181	181	0
GPT-5.6 Sol	176	201	25
Opus 4.8	150	150	0
Sonnet 5	106	152	46
Gemini 3.7 Flash	96	179	83
Muse Spark 1.3	87	125	38
DeepSeek V4.1 Flash	79	180	101
Kimi K3	79	165	86

Table 2: Verified coverage before the first conversation compaction and after all recorded context iterations.

Agent-Scaffold Comparison We evaluate Opus 5 through both our common agent loop and Claude Code (Anthropic, 2026f). Both configurations receive the same 341 formal workspaces and textbook context, have a maximum budget of $200 per task, and undergo identical clean-room verification. Claude Code uses the benchmark's Lean REPL, Loogle, build, and report services alongside its native `Read`, `Edit`, `Write`, `Glob`, and `Grep` tools; Bash, web access, and subagents are disabled.

Claude Code solves 218/341 tasks, compared with 211/341 for the common loop. The configurations agree on 308 problems: both solve 198, Claude Code alone solves 20, the common loop alone solves 13, and neither solves 110. Figure 7 compares their coverage over recorded dollar and turn budgets.

Claude Code's advantage is primarily one of efficiency and persistence rather than a different mathematical strategy. In 18 of its 20 exclusive successes, both configurations pursue the same mathematical approach, but the common loop stops before completing the final obligations. Plausible explanations include its native system prompt, internal compaction policy, and training for its own tools, none of which we can observe or isolate. Appendix K provides the detailed tool-use and failure analysis.

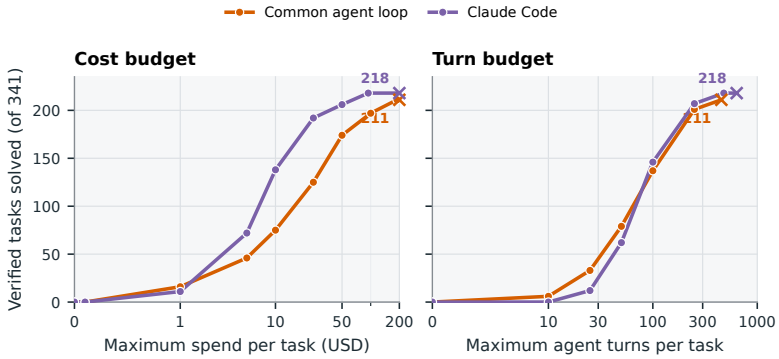


Figure 7: Budget-conditioned comparison of Opus 5 under the common agent loop and Claude Code on the same 341 tasks.

Informal-Source Ablation We ablate access to the textbook by evaluating Opus 5 and Gemini 3.7 Flash on all 341 tasks with the same formal workspace, model settings, tools, budget, stopping rule, and grader. Surprisingly, although both models consult the textbook in more than 93% of source-enabled runs, it provides little aggregate improvement: Opus solves 204 tasks with the source and 200 without it, while Gemini solves 136 and 133, respectively. Neither paired difference is statistically significant, and turns, cost, and token use are similar across conditions (Figure 8). The source also does not systematically shorten the generated Lean code on tasks solved in both conditions. Trace inspection instead shows a task-specific effect: some runs benefit from the book's proof decomposition, whereas some source-free runs find shorter arguments better adapted to Lean. Thus explicit source access changes individual proof trajectories but provides no clear aggregate advantage once the formal statement and its local context are available. We attribute this to the strong mathematical knowledge of current frontier models. Appendix K.1 gives the complete paired statistics, source-use analysis, and case studies.

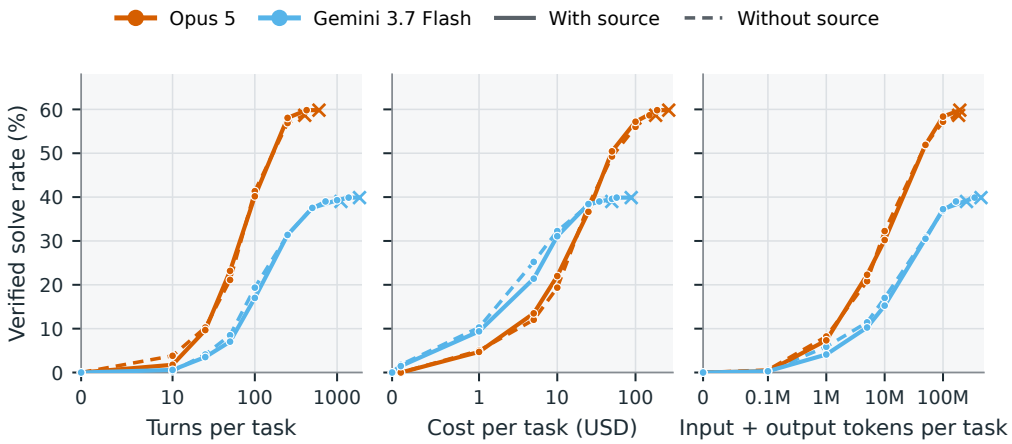


Figure 8: Efficiency by turns, recorded provider cost, and provider-reported input plus output tokens with and without the textbook source on the 341 audited benchmark tasks for Opus 5 and Gemini 3.7 Flash.

8 Limitations and Conclusion

Limitations. AutoformBench covers 24 mathematical sources, but it is not representative of all advanced mathematics: its domains and statements are constrained by what can presently be expressed against Mathlib within a bounded evaluation. The benchmark supplies expert-written formal statements, so it evaluates proof completion and construction of supporting theory rather than end-to-end statement autoformalization. Finally, kernel acceptance establishes a theorem relative to its audited statement and assumptions; it does not establish that the generated code has the abstraction quality or maintainability required for upstream inclusion in a useful library such as Mathlib.

Conclusion. AutoformBench is a natural step toward measuring how well models can construct missing formal infrastructure. Its problems, however, remain isolated in separate workspaces. The next step is to test whether agents can recognize shared missing theory, build reusable abstractions, and amortize that work across related results. Moving from isolated problems to cumulative developments would measure the transition from theorem solving to extending formal libraries.

9 AI USE STATEMENT

Beyond their role as the systems under evaluation, large language models assisted the authors in reviewing the expert-written Lean formalizations, and were used for proofreading, phrasing suggestions, and boilerplate code for data processing and plotting. Every target statement was written and independently audited by human experts, and no model judgment enters grading, which relies solely on Comparator and the Lean and Nanoda kernels. All conceptual contributions, including the benchmark design, source and theorem selection, verification protocol, and experimental design, were made by the human authors.

10 ETHICS STATEMENT

AutoformBench involves no human-subject data; formalizations were produced by paid PhD-level contractors through Labelbox, and all informal sources are openly available and attributed (Table 11). One potential but small risk is that a kernel-checked proof certifies only its formal statement, so a mis-formalized target can lend false credibility to an incorrect result, and agents occasionally claim unverified success (Appendix K.6). We therefore recommend combining human statement review with independent clean-room verification in downstream use.

11 REPRODUCIBILITY STATEMENT

We release all 341 tasks in Harbor format, each pinned to Lean 4.32.2 and a fixed Mathlib commit and including the graded-declaration and permitted-axiom specification and informal source material. Appendix B details the tools, sandbox, complete prompts, budget and termination rules, clean-room grading procedure, and exact revisions of Comparator, Nanoda, Landrun, and Loogle; model settings appear in Section 5, and the repeatability panel and its selection seed in Appendix H. Since proprietary API models may change, trajectories may not be exactly reproducible, but any submission can be re-verified independently from its Lean files alone.

REFERENCES

Tudor Achim, Alex Best, Alberto Bietti, et al. Aristotle: IMO-level automated theorem proving. *arXiv preprint arXiv:2510.01346*, 2025.

AI-MO. NuminaMath-LEAN. https://huggingface.co/datasets/AI-MO/NuminaMath-LEAN, 2025. Dataset of competition-mathematics problems formalized in Lean 4; accessed September 25, 2026.

Anthropic. Claude Fable 5 & Claude Mythos 5 system card. System card, 2026a. URL https://www.anthropic.com/claude-fable-5-mythos-5-system-card. Dated June 9, 2026; accessed September 26, 2026.

Anthropic. Claude Opus 4.8 system card. System card, 2026b. URL https://www.anthropic.com/claude-opus-4-8-system-card. Dated May 28, 2026; accessed September 26, 2026.

Anthropic. Claude Opus 5 system card. System card, 2026c. URL https://www.anthropic.com/claude-opus-5-system-card. Dated July 24, 2026; accessed September 26, 2026.

Anthropic. Claude Sonnet 5 system card. System card, 2026d. URL https://www.anthropic.com/claude-sonnet-5-system-card. Dated June 30, 2026; accessed September 26, 2026.

Anthropic. FormalQualBench, 2026e. URL https://www.anthropic.com/research/formalizing-fermats-last-theorem. Accessed September 26, 2026.

Anthropic. Claude Code. Documentation, 2026f. URL https://code.claude.com/docs/en/overview. Accessed September 25, 2026.

Scott Armstrong and Julia Kempe. Formalization of De Giorgi–Nash–Moser theory in Lean, 2026. URL https://arxiv.org/abs/2604.05984.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Joachim Breitner. Loogle. Software repository, 2023. URL https://github.com/nomeata/loogle. Mathlib search tool; evaluated revision 9f11169aaebf1ed1e7dcc4077f2aafe0fcf66fd0.

Kevin Buzzard. The Jacobian challenge. GitHub Gist, https://gist.github.com/kbuzzard/778bc714030b3e974ab5f4038783d1a9, 2026. Lean 4 challenge specification, version 0.4; accessed September 25, 2026.

Evan Chen, Chris Cummins, et al. Fel's conjecture on syzygies of numerical semigroups. *arXiv preprint arXiv:2602.03716*, 2026.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction—CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/blob/dba1be0a40aa45a94ad051997016db3960a90277/DeepSeek_V41_Tech_Report.pdf. Released September 10, 2026; accessed September 26, 2026.

Moritz Firsching, Paul Lezeau, Salvatore Mercuri, Miklós Z. Horváth, Yaël Dillies, Calle Sönne, Eric Wieser, Fred Zhang, Thomas Hubert, Blaise Agüera y Arcas, and Pushmeet Kohli. Formal Conjectures: An open and evolving benchmark for verified discovery in mathematics. *arXiv preprint arXiv:2605.13171*, 2026. URL https://arxiv.org/abs/2605.13171.

Fabian Gloeckle, Ahmad Rammal, Charles Arnal, Remi Munos, Vivien Cabannes, Gabriel Synnaeve, and Amaury Hayat. Automatic textbook formalization. *arXiv preprint arXiv:2604.03071*, 2026.

Google DeepMind. Gemini 3.7 Flash: Model card. Model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/. Published August 13, 2026; accessed September 26, 2026.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, August 2026. URL https://doi.org/10.5281/zenodo.22052497. Version 0.22.0.

Sidharth Hariharan, Christopher Birkbeck, Seewoo Lee, Ho Kiu Gareth Ma, Bhavik Mehta, Auguste Poiroux, and Maryna Viazovska. Progress in formalizing sphere packing in dimension 8. *arXiv preprint arXiv:2604.23468*, 2026.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y. URL https://doi.org/10.1038/s41586-025-09833-y.

Jiedong Jiang, Wanyi He, Yuefeng Wang, Guoxiong Gao, Yongle Hu, Jingting Wang, Nailin Guan, Peihao Wu, Chunbo Dai, Liang Xiao, and Bin Dong. FATE: A formal benchmark series for frontier algebra of multiple difficulty levels. In *International Conference on Learning Representations (ICLR)*, pp. 10543–10580, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/11c0318fc7b0315fa9542449b1c9ba32-Abstract-Conference.html.

Kimi Team et al. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL https://arxiv.org/abs/2607.24653.

Labelbox. Annotation company. https://labelbox.com/, 2026.

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. doi: 10.52202/068431-1910.

Landrun contributors. Landrun. Software repository, 2025. URL https://github.com/Zouuup/landrun. Lightweight unprivileged sandbox using Landlock; evaluated revision 5ed4a3db3a4ad930d577215c6b9abaa19df7f99f.

Lean FRO. Comparator. https://github.com/leanprover/comparator, 2025. Software repository; evaluated revision 789279735fe44c1c05dc54bb9f46ba4d9b8c7611.

Lean FRO. Lean Eval. https://github.com/leanprover/lean-eval, 2026. Software repository; accessed September 25, 2026.

Austin Letson, Leopoldo Sarra, Auguste Poiroux, Oliver Dressler, Paul Lezeau, Dhyan Aranha, Frederick Pu, Aaron Hill, Miguel Corredera Hidalgo, Julian Berman, George Tsoukalas, and Lenny Taelman. SorryDB: Can AI provers complete real-world Lean theorems? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=hcO7EwDTL5.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, et al. Process-driven autoformalization in Lean 4. *arXiv preprint arXiv:2406.01940*, 2024.

Math Inc. FormalQualBench. Project website, 2026. URL https://www.math.inc/formalqualbench. Accessed September 25, 2026.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Meta Superintelligence Labs. Introducing Muse Spark 1.3. Meta AI Research, 2026. URL https://research.meta.ai/blog/introducing-muse-spark-1-3. Published September 2, 2026; accessed September 26, 2026.

Lazar Milikic, Simon Guilloud, Khanh Nguyen, and Viktor Kuncak. LeanFlow: A case study in workflow-driven Lean autoformalization. *arXiv preprint arXiv:2607.20503*, 2026. URL https://arxiv.org/abs/2607.20503. ICML 2026 AI for Math Workshop.

MIT. OpenCourseWare. https://ocw.mit.edu/, 2026.

Arshia Soltani Moakhar, Iman Gholami, Max Springer, Mahdi JafariRaviz, and MohammadTaghi Hajiaghayi. Beyond the library: An agentic framework for autoformalizing research mathematics. *arXiv preprint arXiv:2606.31134*, 2026. URL https://arxiv.org/abs/2606.31134.

Nanoda contributors. Nanoda. Software repository, 2026. URL https://github.com/robsimmons/nanoda_lib. Independent Lean kernel; evaluated revision 68d5ca9db226849b41a6fff59d796ff19d0a8840.

Tianjiao Nie, Ao Zhang, Yusen Tang, Damiano Testa, Shing-Tung Yau, Peng Li, and Yuan Zhou. FormaTheoria: Constructing large-scale Lean theories from mathematical literature—toward the formalization of the classification of finite simple groups. *arXiv preprint arXiv:2608.10894*, 2026.

OpenAI. GPT-5.6 Sol model. OpenAI API documentation, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Released July 9, 2026; accessed September 26, 2026.

OpenAI. GPT-6 Astra model. OpenAI API documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Released September 3, 2026; accessed September 26, 2026.

Niket Patel, Ahmad Rammal, Amaury Hayat, Remi Munos, and Julia Kempe. Learning to discover interesting mathematics, 2026. URL https://arxiv.org/abs/2609.28603.

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

Ahmad Rammal, Niket Patel, Fabian Gloeckle, Amaury Hayat, Julia Kempe, Remi Munos, Charles Arnal, and Vivien Cabannes. Formalizing mathematics at scale. *arXiv preprint arXiv:2605.29955*, 2026.

Nikil Ravi, Kexing Ying, Vasilii Nesterov, Rayan Krishnan, Elif Uskuplu, Bingyu Xia, Janitha Aswedige, and Langston Nashold. FormalProofBench: Can models write graduate level math proofs that are formally verified? *arXiv preprint arXiv:2603.26996*, 2026. URL https://arxiv.org/abs/2603.26996. ICLR 2026 Workshop on AI Verification in the Wild (VerifAI-2).

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824.

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. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 37, pp. 11545–11569, 2024. doi: 10.52202/079017-0368. arXiv:2407.11214.

Josef Urban. 130k lines of formal topology in two weeks: Simple and cheap autoformalization for everyone? (short paper). In *36th International Conference on Interactive Theorem Proving (ITP 2026)*, volume 382 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 31:1–31:9. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPIcs.ITP.2026.31. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2026.31.

Zichen Wang, Wanli Ma, Zhenyu Ming, Gong Zhang, Kun Yuan, and Zaiwen Wen. M2F: Automated formalization of mathematical literature at scale. *arXiv preprint arXiv:2602.17016*, 2026.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 32353–32368, 2022. doi: 10.52202/068431-2344.

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. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 37, pp. 105848–105863, 2024. doi: 10.52202/079017-3357.

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. *arXiv preprint arXiv:2505.02735*, 2025.

Yuanhe Zhang, Yuekai Sun, Taiji Suzuki, Jason D. Lee, and Fanghui Liu. LeanMarathon: Toward reliable AI co-mathematicians through long-horizon Lean autoformalization. *arXiv preprint arXiv:2606.05400*, 2026.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: a cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2109.00110.

A BENCHMARK STRUCTURE: PROOF OBLIGATIONS AND LOCAL CONTEXT

This section gives two complementary views of the benchmark : the number of explicit proof obligations in each workspace and the local formal context on which the target statements depend.

A.1 PROOF OBLIGATIONS

Table 3 gives the complete distribution of graded proof obligations across the frozen benchmark. Of the 1,155 graded declarations, 1,147 use theorem syntax and 8 use instance syntax for proposition-valued typeclasses; there are no graded definition or data holes.

Holes	Tasks	Share	Oblig.	Holes	Tasks	Share	Oblig.
1	230	67.45%	230	14	4	1.17%	56
2	25	7.33%	50	15	1	0.29%	15
3	19	5.57%	57	16	1	0.29%	16
4	9	2.64%	36	17	1	0.29%	17
5	7	2.05%	35	18	2	0.59%	36
6	5	1.47%	30	19	1	0.29%	19
7	2	0.59%	14	20	2	0.59%	40
8	4	1.17%	32	24	1	0.29%	24
9	3	0.88%	27	26	2	0.59%	52
10	5	1.47%	50	27	2	0.59%	54
11	4	1.17%	44	30	1	0.29%	30
12	5	1.47%	60	33	1	0.29%	33
13	3	0.88%	39	59	1	0.29%	59
341 tasks; 1,155 graded proof obligations							

Table 3: Number of graded proof holes per benchmark task. **Oblig.** is the total number of obligations contributed by that row.

The 111 multi-hole tasks have a median of 5 and a mean of 8.33 holes. The distribution is concentrated: the largest 10 tasks contain 292/1,155 obligations (25.3%), the largest 20 contain 451 (39.0%), and the 44 tasks with at least eight holes contain 703 (60.9%). Designating the final graded declaration in each workspace as its headline target leaves 814 supporting obligations, or 70.5% of all graded declarations. Agent-added definitions and lemmas are additional to these benchmark-provided holes.

A.2 FORMAL CONTEXT REQUIRED TO STATE A TASK

A formal target uses constants from both Mathlib and its benchmark workspace. To quantify the context supplied before proof search, we start from each elaborated target type and recursively follow references into `Challenge.lean` and `ChallengeDeps.lean`; for a multi-target task, we take the union of the resulting graphs. Table 4 reports the number of reachable declarations, their source lines of code, and the longest local dependency chain. No model-generated code is counted. We use the 281 tasks solved by GPT-6 Astra so that these input-side measurements can be paired with the accepted-proof analysis in Appendix G.

Input-side metric	Median	IQR	Maximum
Reachable workspace declarations	5	2–12	54
Source LOC of those declarations	22	7–49	293
Longest workspace dependency chain	2	1–3	9

Table 4: Formal context required by the graded target types in the 281-task GPT-6 Astra accepted-solution cohort.

The median task depends on five workspace declarations occupying 22 lines, with a longest local chain of two. Among constants mentioned directly in its target types, the median Mathlib fraction is 65.0%. Thus predominantly Mathlib vocabulary can coexist with a nontrivial local API. These measurements describe context supplied with the task, not infrastructure built by the agent.

B EXECUTION AND CLEAN-ROOM VERIFICATION

B.1 WORKSPACE STRUCTURE

LEAN EVAL (Lean FRO, 2026) generates a standalone Comparator workspace for each benchmark item. All workspaces use Lean 4.32.2 and Mathlib commit `905b95818eb32af7874a58b427f50c1711a5e96c`. The principal files are:

```
ChallengeDeps.lean        supporting definitions
Challenge.lean            trusted target declarations
Submission.lean           agent's candidate proofs
Submission/Helpers.lean   optional auxiliary lemmas
Solution.lean             trusted grading/build wiring
config.json               graded names and permitted axioms
book/                     informal textbook material
TASK.md                   agent instructions
```

The agent receives a disposable copy of this workspace. Although it can inspect the trusted challenge and build configuration, its only graded deliverables are `Submission.lean` and the optional `Submission/` directory. `Submission.lean` is initialized with the target declarations and `sorry` proof bodies; agents may add definitions and auxiliary lemmas but must fill every target proof.

B.2 TOOLS AND EXECUTION SANDBOX

The agent receives five classes of MCP service: scoped filesystem operations; `run_lean_code`, a scratch Lean REPL using the benchmark's pinned Mathlib environment; `loogle_search`, for searching Mathlib declarations; a fixed build operation equivalent to `lake build Solution`; and `submit_report`. The build service exposes compiler diagnostics without granting a general shell. The report service records a structured outcome (`solved` or `unsolved`), optional diagnostic flags, and a short free-text account of progress or failure. It is a source of diagnostic metadata, not the grading oracle.

Each MCP subprocess runs under Bubblewrap. The problem workspace is its only writable mount; required system, Lean, and tool directories are mounted read-only; temporary storage and the home directory are private; the environment is allowlisted; and network access is disabled.

B.3 AGENT PROMPTS

API-model runs use a fixed solver prompt. The complete machine-readable templates are released with the evaluation artifacts; we reproduce the result-relevant instructions below, omitting the diagnostic-report flag taxonomy.

Solver instruction.

> You are an expert Lean 4 theorem-proving agent working in a self-contained workspace. Read `TASK.md` and `Challenge.lean`, then replace every `sorry` in `Submission.lean`; auxiliary definitions and lemmas may be added to `Submission/Helpers.lean`. Use the filesystem tools to edit the submission, `run_lean_code` as a scratchpad, `loogle_search` to search Mathlib, and `build` to compile the candidate. Missing Mathlib infrastructure is part of the task: build the necessary supporting theory rather than stopping when a lemma cannot be found. Do not modify target statements, introduce axioms, or bypass proof checking. Finish by calling `submit_report` with a solved or unsolved outcome and a concise account of the work completed and any remaining obstacle.

Continuation prompt. If an agent stops without a report, it is reminded that it must either submit a completed proof or continue using the available tools. If it reports `unsolved` before reaching the minimum-effort threshold (10% of its budget), it receives:

> You have not worked hard enough on this problem yet. These problems are intentionally difficult and may require substantial new Lean infrastructure; missing Mathlib lemmas are expected, not a reason to stop. Spend more time developing the necessary definitions and helper lemmas, test them with the available tools, and keep iterating until the solution builds.

Conversation compaction (common-agent runs). To support trajectories longer than one context window, the common agent loop compacts its conversation after a completed model turn when the context approaches its limit.

For compaction, the harness calls the same model at the same reasoning setting, with tools disabled, using the following prompt:

> **System instruction.** You are a summarizer. Produce a concise summary of the conversation below. Focus on: key findings, decisions made, actions taken, and remaining work. Be specific about important details like file paths, names, and values.
>
> **User template.**
>
> ```
> Summarize this conversation:
> {conversation_text}
> ```

The original solver system prompt is then restored, and the prior non-system history is replaced by one user message prefixed `[Context summary of previous conversation]`. Compaction changes neither the workspace files nor the running tool services; thus verified Lean progress survives even though the detailed dialogue does not. If summarization fails or returns no usable text, the original history is restored. The procedure may repeat within a run.

B.4 Budget and Termination

All models are evaluated under a maximum provider-cost budget of $200 per problem. A solved report may terminate immediately. An unsolved report submitted before 10% of the budget is normally rejected and the agent is instructed to continue; the local-model driver accepts the sixth such report to prevent an unproductive rejection loop. Appendix K.5 analyzes the effect of this rule. The final 10% is reserved for at most three wrap-up turns so that an agent that exhausts its proving budget can still record partial progress and remaining obstacles. Reported dollar costs are computed from provider-reported input, output, and cached-token counts, including separate cache-read and cache-write counts where available, using the public API prices applicable to each model at evaluation time. They measure estimated API-equivalent inference cost rather than hardware-normalized compute or invoiced expenditure.

B.5 Clean-Room Grading

When an agent finishes, the harness creates a new workspace from the trusted benchmark copy and overwrites exactly two paths with agent output: `Submission.lean` and `Submission/`. All other files—including the reference declarations, build wiring, graded-name list, pinned definitions, and permitted-axiom configuration—come from the trusted copy. Consequently, changes made elsewhere in the disposable workspace cannot affect the verdict.

Comparator then compiles the candidate in its Landrun sandbox (Landrun contributors, 2025) and checks that each selected declaration has the same statement as its trusted counterpart, that pinned definitions retain their values, and that the proof uses only permitted axioms. Accepted submissions are additionally replayed through the independent Nanoda kernel (Nanoda contributors, 2026). The resulting Comparator verdict, rather than the agent's `submit_report` claim or its local build result, defines benchmark success.

Harbor task format and release. Harbor's container-based task format and execution harness, also used by Terminal-Bench 2.0, provide a common interface for heterogeneous agent scaffolds. The public release packages every benchmark workspace as a Harbor 0.22.0 task, allowing any Harbor-supported or custom agent to be evaluated under the grading semantics described above.

C Evaluation Cost

Evaluation spending and the unsolved frontier. Figure 9 separates recorded provider spending by model, run outcome, and cross-model task coverage. Across all 3,410 model–task runs, the 1,585 unsuccessful runs account for $102,453.36 of the $157,649.74 total (65.0%). The 59 tasks solved by none of the ten models consume $34,075.64 and 192,811 agent turns. The 66 tasks solved by exactly one model consume a further $42,801.86; together, these 125 frontier tasks account for $76,877.50, or 48.8% of all recorded leaderboard spending. Thus nearly half of the evaluation cost is devoted to tasks that remain unsolved or lie at the edge of current model capability.

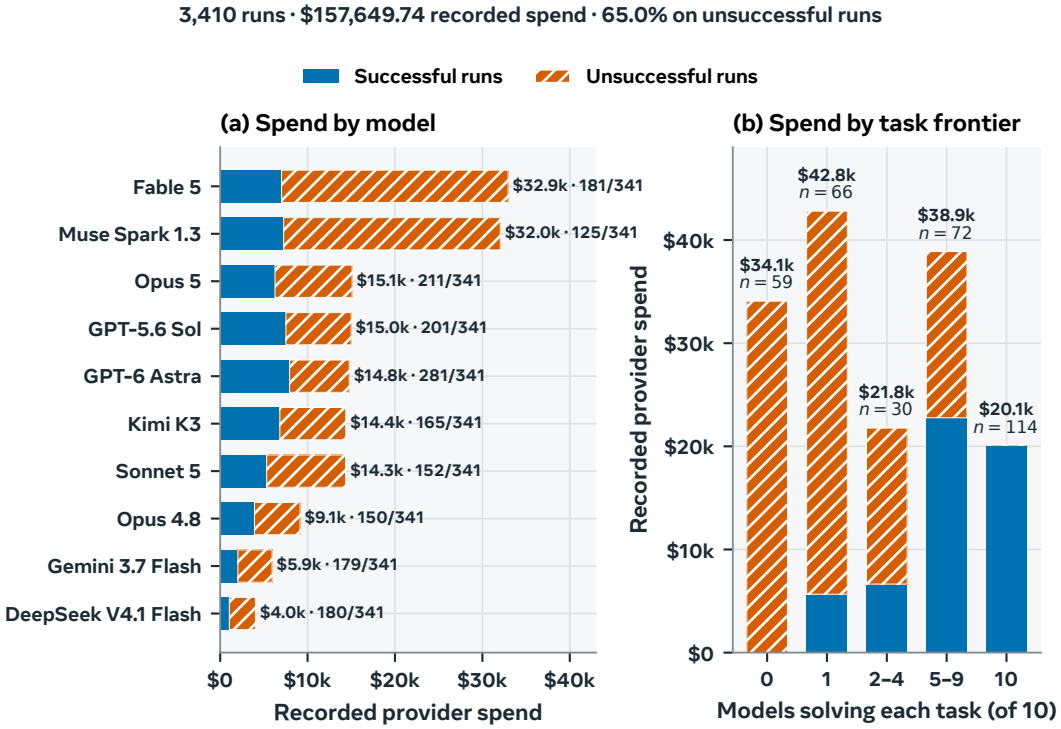


Figure 9: Recorded provider spending across the ten common-harness leaderboard campaigns. Left: spend by model, split between verified and unsuccessful runs; labels give total spend and verified score. Right: the same spending grouped by the number of models that solve each task; n is the number of tasks in each group.

D Budget-Conditioned Efficiency

The main-text leaderboard in Figure 3 uses the canonical recorded trajectory for each model–task pair. This appendix provides complementary presentations of the same per-task resource accounting.

Resource efficiency. Figure 10 reveals different efficiency profiles. At any fixed per-task cutoff on one of its three resource axes, the highest curve identifies the model with the greatest observed verified coverage under that budget. This matters when evaluating all tasks at the full $200 cap is impractical. GPT-6 Astra leads across nearly the entire turn frontier and at recorded dollar budgets above approximately $9 per task; DeepSeek V4.1 Flash leads below that cost threshold. Astra also leads once the total provider-reported token budget reaches approximately 0.41 million tokens per task. Turns, tokens, and monetary cost therefore capture different aspects of efficiency, and inexpensive early success does not necessarily predict the highest eventual coverage.

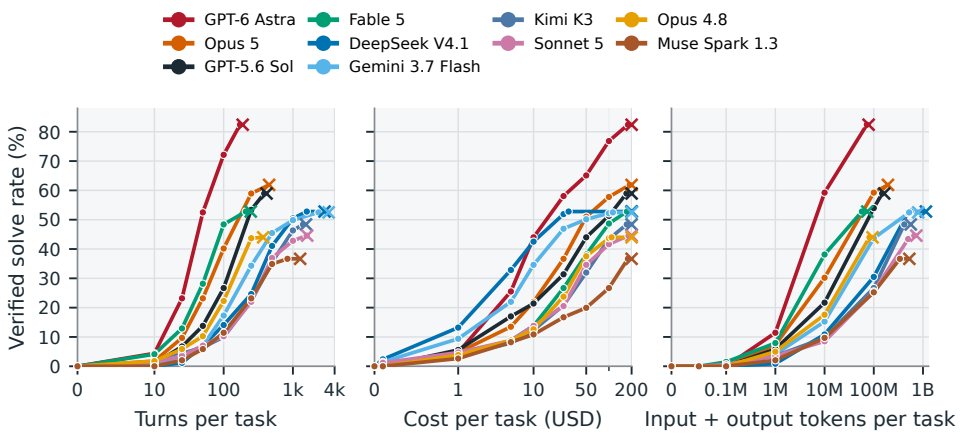


Figure 10: Efficiency profiles on the 341 audited benchmark tasks by turns (left), recorded provider cost (center), and provider-reported input plus output tokens (right). Circles give exact scores at selected retrospective cutoffs, straight segments are visual guides, and × marks the end of observed support. Resource axes begin at zero and use a symmetric-log scale. Token counts use each provider's reporting. One Fable success with inseparable retry accounting is omitted from all three resource curves but retained in its final score.

The four panels in Figure 3 use dollars, turns, total provider-reported tokens, and conversation iterations. A conversation iteration is the segment before the first compaction or between successive compactions, so iteration equals the recorded compaction count plus one. Token counts are the provider-reported input plus output tokens and are tokenizer-dependent. One verified Fable run has retry-confounded resource totals: we conservatively place it at the $200 cap in the dollar panel and omit it from the other three panels. Circles give exact scores at selected retrospective cutoffs, straight segments connect them as visual guides, and × marks the end of each model's observed support; no curve is extrapolated beyond that point.

Figure 11 isolates generated output tokens, which are a useful complement to total input-plus-output volume. GPT-6 Astra leads essentially throughout the measured output-token range. As with total tokens, this comparison is descriptive because provider tokenizers and accounting conventions differ.

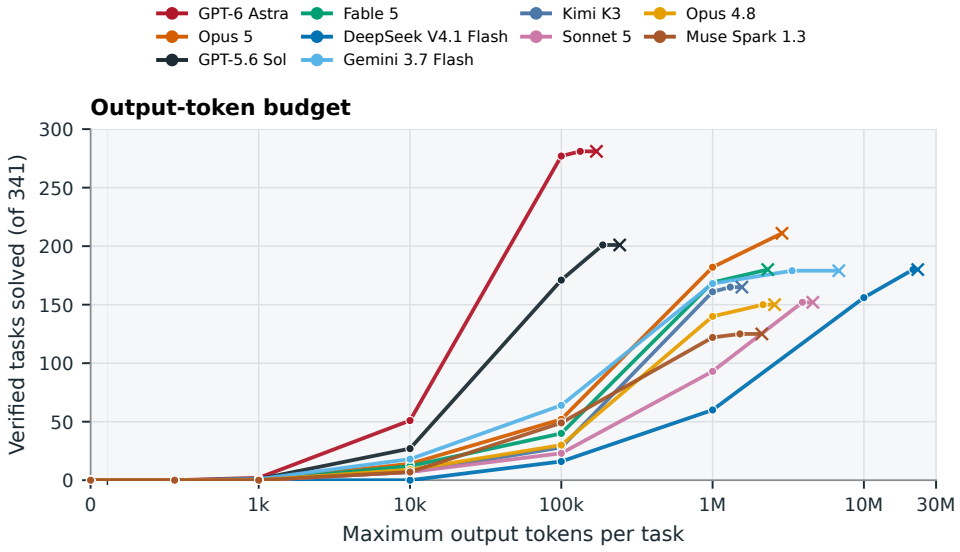


Figure 11: Budget-conditioned leaderboard by provider-reported output tokens. Markers are exact checkpoint evaluations and connecting segments are visual guides; the axis begins at zero and uses a symmetric-log scale. One Fable success with inseparable retry accounting is omitted.

E MATCHED MODEL EFFICIENCY

Controlled comparison on common successes. To separate model behavior from differences in the tasks each model solves, we restrict to the 114 tasks solved by all ten common-harness models. For every task and resource, we divide each model's recorded value by the minimum among the ten models, then report the median ratio in Table 5. A ratio of one is best. The final three columns count tasks on which a model attains the minimum, counting ties for each model. DeepSeek is cheapest on 79/114 tasks. Astra uses the fewest turns on 95/114 and leaves the least added code on 93/114.

	Median ratio to taskwise best			Tasks attaining best		
Model	Cost	Turns	Added LOC	Cost	Turns	Added LOC
GPT-6 Astra	3.01	1.00	1.00	13	95	93
Opus 5	4.88	1.77	1.49	0	2	4
GPT-5.6 Sol	5.83	2.83	1.62	1	1	2
Fable 5	8.52	1.46	1.46	0	20	13
DeepSeek V4.1 Flash	1.00	6.26	2.22	79	0	0
Gemini 3.7 Flash	2.09	4.50	2.01	19	0	2
Kimi K3	11.54	9.44	2.73	0	1	1
Sonnet 5	10.25	7.86	2.08	2	2	1
Opus 4.8	10.71	3.30	1.62	0	2	6
Muse Spark 1.3	19.70	6.77	3.09	0	0	1

Table 5: Efficiency fingerprints on the identical 114-task cohort solved by all ten models. Added LOC is the net code remaining in the final submission. Best-task counts include ties.

Matched resource profiles. Figure 4 summarizes the matched comparison in the main text. Figures 12–14 show the underlying task-level distributions on the same 114 tasks. For each task and metric, we divide each model's value by the median across the ten models. This normalization removes the large between-task variation and exposes systematic differences among models.

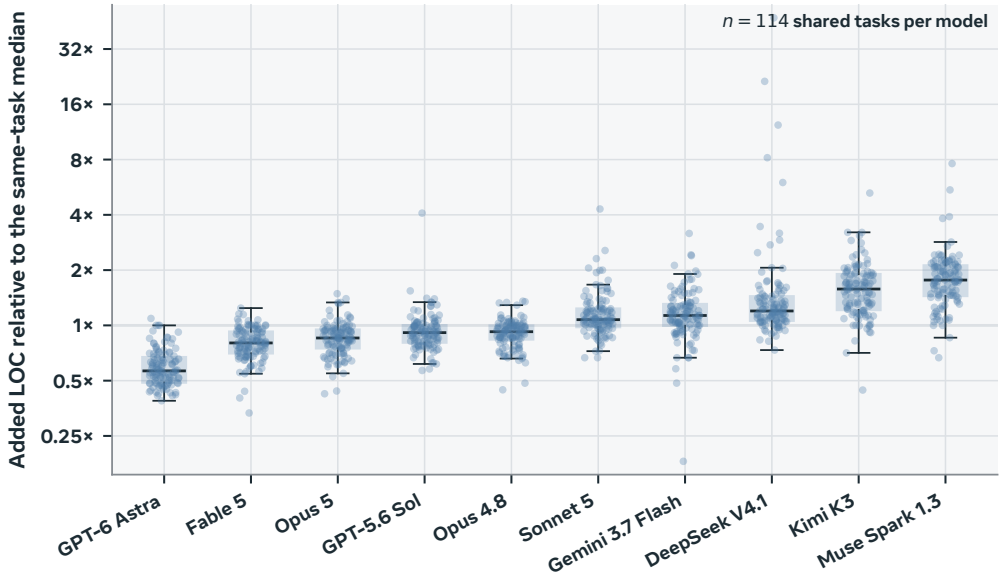


Figure 12: Added Lean code on the 114 tasks solved by all ten common-harness models, normalized by the median LOC for the same task. Each point is one model–task pair; boxes summarize the per-model distributions. A value of $1\times$ is the within-task median, while $0.5\times$ and $2\times$ denote half and twice that amount of code. Models are ordered by their median ratio.

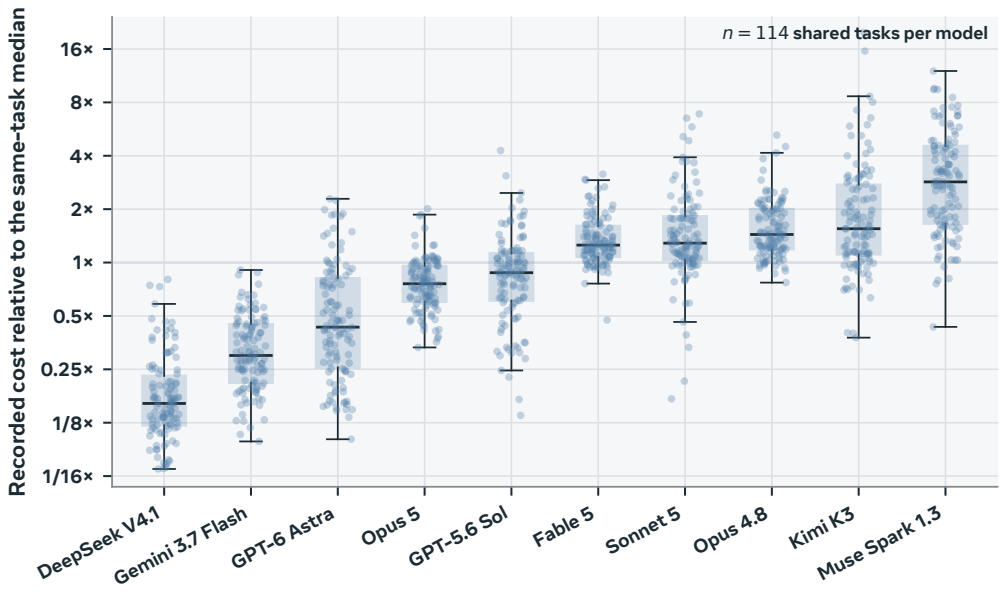


Figure 13: Recorded provider cost on the 114 shared tasks, normalized by the median cost for the same task. Each point is one model–task pair; boxes summarize the per-model distributions. Models are ordered by median ratio.

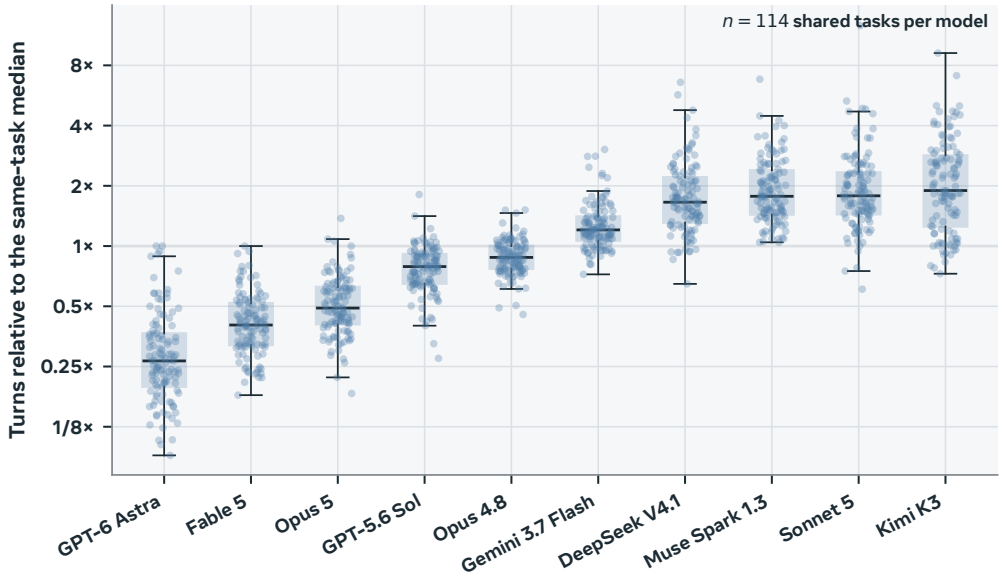


Figure 14: Agent turns on the 114 shared tasks, normalized by the median turn count for the same task. Each point is one model–task pair; boxes summarize the per-model distributions. Models are ordered by median ratio.

F Shared Difficulty Frontier

Table 6 reports the complete per-textbook results. Absolute difficulty varies substantially across sources, but the model outcomes do not reveal a broad domain-specific specialization that would explain the shared frontier.

Table 6: Per-book verified solve counts on the 341-task benchmark. Model counts use the headline condition at the highest evaluated effort. **Astra**: GPT-6 Astra; **O5**: Opus 5; **Sol**: GPT-5.6 Sol; **F5**: Fable 5; **DS**: DeepSeek V4.1 Flash; **G3.7F**: Gemini 3.7 Flash; **K3**: Kimi K3; **S5**: Sonnet 5; **O4.8**: Opus 4.8; **MS**: Muse Spark 1.3. **Any** counts tasks solved by at least one model. Row-wise best model counts are bold.

Book	Tasks	Astra	O5	Sol	F5	DS	G3.7F	K3	S5	O4.8	MS	Any
Algebraic Combinatorics	2	**2**	**2**	**2**	**2**	1	1	**2**	1	**2**	0	2
Algebraic Geometry	8	**6**	1	1	1	1	1	1	0	0	0	6
Algebraic Topology	16	**10**	5	4	5	3	3	3	3	2	2	10
Arithmetic Geometry	26	**20**	16	16	13	13	14	11	10	12	10	20
Boolean Functions	2	**2**	**2**	**2**	**2**	**2**	1	1	0	1	1	2
Buildings	11	**11**	9	10	7	7	9	7	5	6	6	11
Cherednik Algebras	15	**12**	6	4	3	3	2	4	1	1	1	12
Complex Variables	12	**12**	**12**	10	9	9	8	9	9	8	8	12
Differential Analysis	18	**17**	14	13	12	14	10	14	12	12	10	17
Elliptic Curves	27	**11**	7	7	5	5	7	5	4	4	4	12
Graph/Additive Comb.	9	**6**	4	4	4	4	4	2	4	4	3	6
High-Dimensional Statistics	25	**25**	**25**	**25**	24	24	22	22	22	21	16	25
Lie Groups and Algebras	28	**18**	11	12	10	11	10	9	10	8	8	18
Low-Rank Models	5	**5**	4	4	4	3	4	3	3	3	3	5
Matrix Concentration	3	**3**	0	0	0	0	0	0	0	0	0	3
Noncommutative Algebra	22	**17**	14	14	11	13	12	10	10	9	6	17
Number Theory	24	**18**	8	8	5	5	5	5	3	3	2	18
PDEs	8	**8**	**8**	7	7	6	7	7	6	6	6	8
Probability	4	**4**	**4**	3	**4**	**4**	**4**	**4**	3	2	2	4
Sparsity Penalties	8	**8**	**8**	**8**	**8**	**8**	**8**	7	7	7	6	8
Stacks Project	32	**31**	28	24	24	22	27	20	20	20	17	31
Submodular Functions	15	**15**	**15**	14	13	14	13	12	11	13	9	15
Tensor Categories	13	**13**	4	5	4	4	3	3	4	2	2	13
Theory of Computation	8	**7**	4	4	4	4	4	4	4	4	3	7
Total (24 books)	**341**	**281**	**211**	**201**	**181**	**180**	**179**	**165**	**152**	**150**	**125**	**282**

Cross-model coverage. Figure 15 gives the complete distribution of task coverage across the ten models evaluated with the common harness.

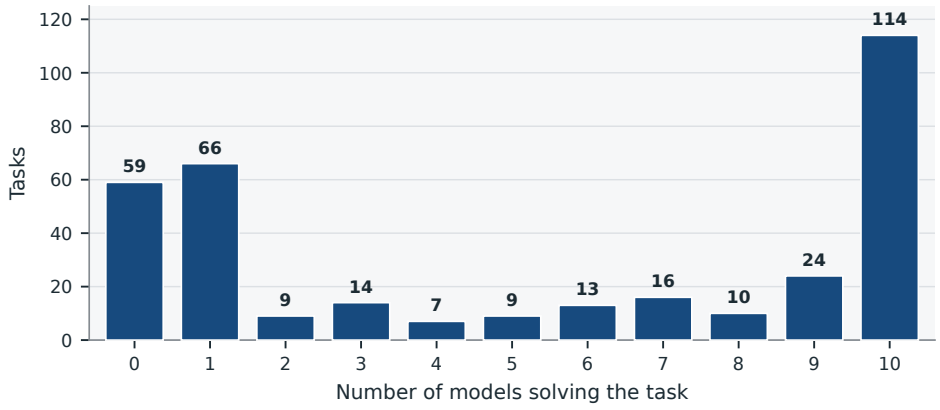


Figure 15: Number of benchmark tasks solved by exactly k of the ten models evaluated with the common harness.

The distribution is strongly polarized: 114 tasks are solved by all ten models, 59 by none, and 168 produce disagreement. Across all 45 model pairs ordered by total score, the higher-scoring model solves at least 95% of the lower-scoring model's successes in 38 pairs; for the median pair, it misses only one such task. Astra solves 281 tasks, while the union of all ten systems reaches only 282.

G Agent-Built Infrastructure and Proximity to Mathlib

G.1 From Generated Submission to Live Proof

Accepted-solution cohort and generated code. We analyze all 281 clean-room-accepted GPT-6 Astra solutions, comprising 208 single-hole and 73 multi-hole tasks. This is the broadest accepted cohort from a single model. Before filtering unused code, these solutions contain a median of 319 skeleton-subtracted, nonblank, noncomment lines of added Lean; the largest contains 2,763. Figure 16 shows the full distribution.

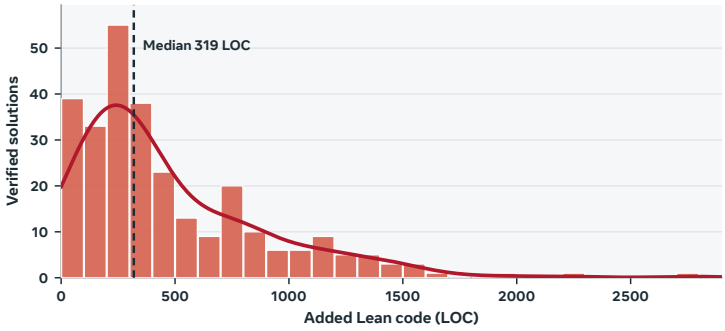


Figure 16: Distribution of skeleton-subtracted, nonblank, noncomment Lean code added by GPT-6 Astra across its 281 accepted solutions.

Live dependency closure. Raw generated code can include abandoned proof attempts. To isolate the code that supports an accepted result, we start from every graded proof and recursively follow dependencies into declarations added by the agent. For a multi-hole task, we take the union of these closures. Everything unreachable from a graded proof is excluded from the live measurements.

Table 7 reports the resulting proof developments. Auxiliary lemmas occur in 261/281 solutions (92.9%), new definitions in 206/281 (73.3%), and new instances in 68/281 (24.2%); only 20/281 (7.1%) use no auxiliary declarations. Supporting declarations contribute 118,853 of 126,388 aggregate live LOC (94.0%), while the graded target proof bodies contribute 7,535 (6.0%). The live closures retain 95.4% of all generated code in aggregate, with a median per-task share of 96.4%.

Live proof metric	Median	IQR	Maximum
Total live proof LOC	300	182–669	2,640
Used helper declarations	24	10–56	223
Used helper LOC	283	151–638	2,639
Used definitions	3	0–10	77
Used lemmas	20	9–44	179

Table 7: Dependency-closed infrastructure in accepted GPT-6 Astra solutions. Unused generated declarations are excluded.

G.2 Single- and Multi-Hole Workspaces

Table 8 shows that substantial supporting theory is not confined to workspaces exposing many holes.

Metric	Single-hole ($n = 208$)	Multi-hole ($n = 73$)
Median live proof LOC	271.5	636
Median used helper declarations	20	53
Median used definitions	2	9
Median used lemmas	16	47
Uses auxiliary lemmas	90.4%	100%
Uses new definitions	67.8%	89.0%
Supporting share of aggregate live LOC	93.4%	95.1%

Table 8: Dependency-closed proof infrastructure by workspace hole count.

Even a single exposed hole therefore usually requires a substantial local theory: 90.4% of single-hole successes use auxiliary lemmas, 67.8% introduce definitions, and 93.4% of their live LOC lies outside the final target body. Conversely, a multi-hole workspace may expose many routine obligations explicitly, so hole count alone is not a complete measure of difficulty.

G.3 Resource Use Across Successful Runs

Finally, we broaden from Astra to every verified common-harness run and compare raw added code with the resources used to produce it. Figures 17 and 18 show that generated code is strongly associated with both interaction length and spending: within-model Spearman correlations range from 0.76 to 0.90 for turns and from 0.81 to 0.90 for cost. Because models solve different task sets, these plots characterize successful-run behavior rather than a controlled comparison.

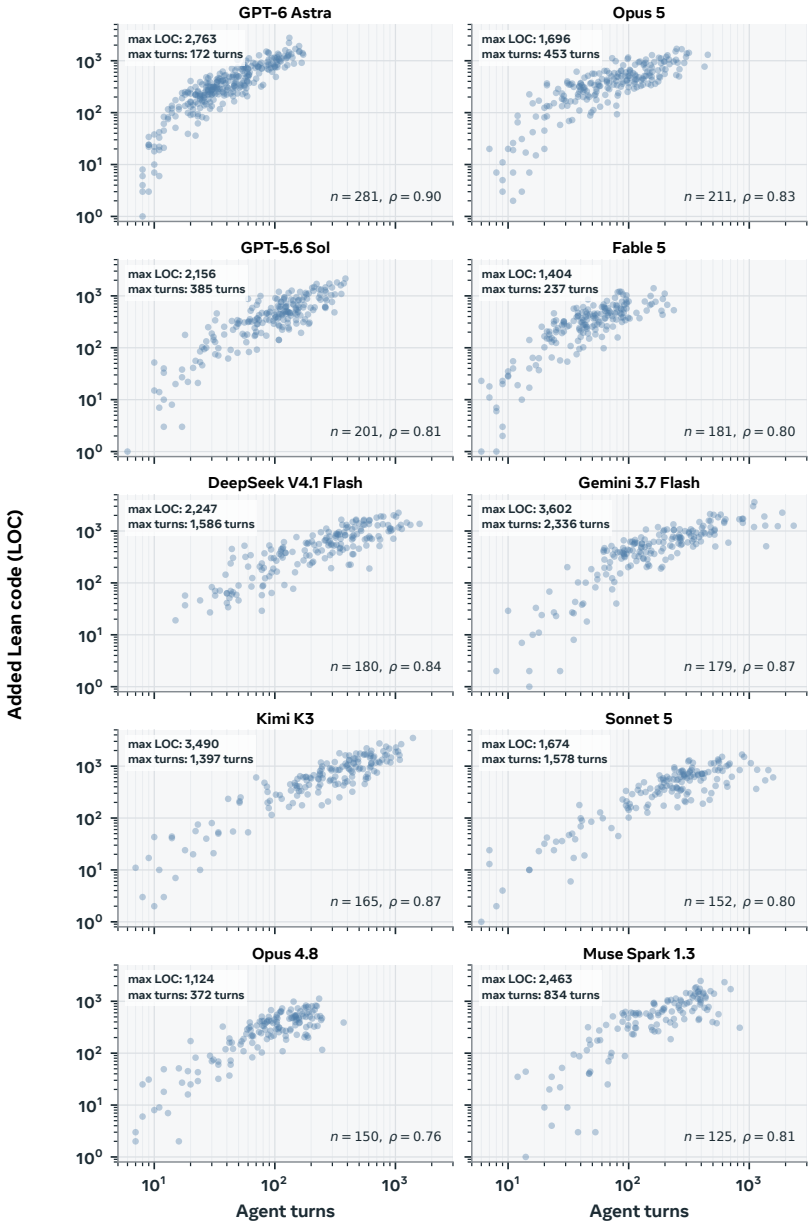


Figure 17: Added Lean code versus agent turns for every verified common-harness run, separated by model. Each panel reports the maximum added LOC and maximum number of turns. Both axes are logarithmic and shared across panels.

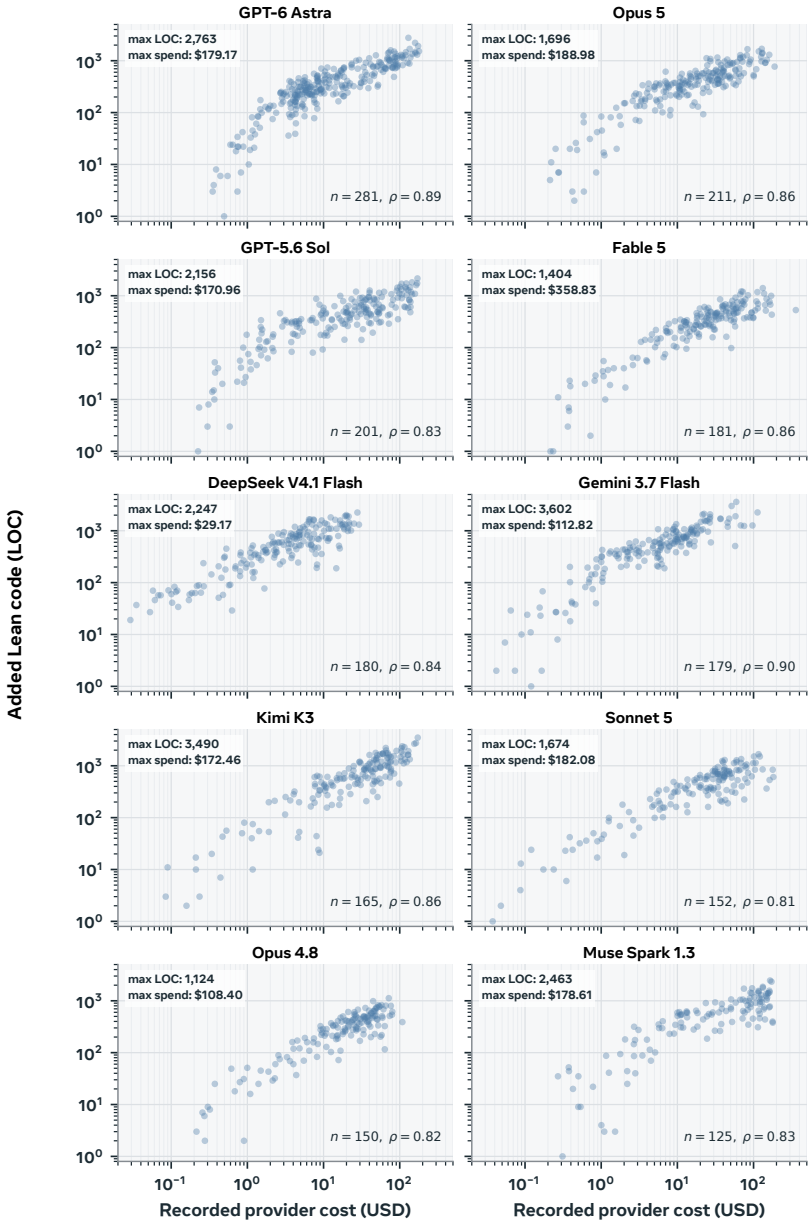


Figure 18: Added Lean code versus recorded provider spending for every verified common-harness run, separated by model. Each panel reports the maximum added LOC and maximum recorded spending. Both axes are logarithmic and shared across panels. The Fable maximum includes inseparable retry usage in its retained record.

H Run-to-Run Stability: Panel Construction and Detailed Results

Repeating every benchmark item several times is unusually expensive in our long-horizon setting. We therefore freeze a 72-problem panel before observing any of the new repeatability runs. The eligible population is the final set of 341 non-defective problems for which both GPT-5.6 Sol and Gemini 3.7 Flash have complete historical results. A problem is marked as historically solved when the clean-room grader accepts it in the final reported evaluation.

Each problem receives one of four joint labels, with the first letter denoting Sol and the second Gemini. Table 9 gives their population counts and panel quotas. Quotas are obtained by multiplying the population proportions by 72 and applying largest-remainder rounding.

Label	Historical headline outcome	Full cohort	Panel
SS	Both solved	173	37
FF	Neither solved	134	28
SF	Sol only	28	6
FS	Gemini only	6	1

Table 9: Historical outcome strata used to construct the repeatability panel.

We selected a deterministic stratified panel of 72 tasks, balanced across all 24 textbooks, the historical outcomes in Table 9, and three structural-size measures: workspace size, number of graded proof obligations, and target-statement size. The task list and selection seed are released for reproducibility.

Repeated-run protocol and interpretation. For each model and panel item, we record three completed runs. Provider or transport failures are replaced under the same replicate identifier. We report the solve rate of each run, pass@3, and the number of items whose outcome agrees in all three. These quantities describe the fixed panel and are not extrapolated to full-benchmark pass@3.

Aggregate outcomes. Table 10 gives the individual-run totals and their union. Figure 6 in the main text additionally shows mean single-run coverage, its observed range, and pass@3 across resource budgets.

Model	Run 1	Run 2	Run 3	pass@3	Unanimous
GPT-5.6 Sol	43	44	43	45	69/72
Gemini 3.7 Flash	38	36	38	39	69/72

Table 10: Verified outcomes across the three runs on the 72-problem panel. Pass@3 counts their union; unanimous counts problems with the same outcome in all three runs.

Strategies across successful runs. We compare the three accepted proofs for every problem solved in all three runs, using their target dependency closures and a manual comparison of the central mathematical argument. GPT-5.6 Sol follows the same high-level route in 38/42 cases (90.5%), and Gemini 3.7 Flash in 34/36 (94.4%). For example, all runs prove Boolean-quotient Sperner through Stanley's up/down operators, injectivity, Hall matchings, and rank symmetry despite markedly different Lean developments. Genuine alternatives also occur: for a spectral-endpoint theorem, runs alternate between compact-spectrum/continuous-functional-calculus and centered-operator/resolvent arguments. Stable success therefore usually reflects convergence on the same mathematics, with substantial freedom in its Lean realization.

I TEXTBOOK SOURCE PROVENANCE

Table 11: Provenance of the 24 textbook sources used in AutoformBench. Each link points to the canonical public course page, project page, or arXiv record from which the benchmark source was obtained

Source	Author(s)	URL
Topics in Algebraic Combinatorics	Richard P. Stanley	MIT OCW
Algebraic Geometry I	Roman Bezrukavnikov et al.	MIT OCW
Lectures on Algebraic Topology	Haynes Miller; notes by Sanath Devalapurkar	MIT OCW
Introduction to Arithmetic Geometry	Andrew V. Sutherland	MIT OCW
Analysis of Boolean Functions	Dor Minzer	MIT OCW
Buildings	Paul Garrett	author PDF
Lecture Notes on Cherednik Algebras	Pavel Etingof and Xiaoguang Ma	arXiv:1001.0432
Functions of a Complex Variable	Sigurdur Helgason	MIT OCW
Differential Analysis	Richard B. Melrose	MIT OCW
Elliptic Curves	Andrew V. Sutherland	MIT OCW
Graph Theory and Additive Combinatorics	Yufei Zhao	MIT OCW
High-Dimensional Statistics	Philippe Rigollet and Jan-Christian Hütter	MIT OCW
Lie Groups and Lie Algebras	Pavel Etingof	arXiv:2201.09397
Generalized Low Rank Models	Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd	arXiv:1410.0342
An Introduction to Matrix Concentration Inequalities	Joel A. Tropp	arXiv:1501.01571
Noncommutative Algebra	Roman Bezrukavnikov	MIT OCW
Number Theory I	Andrew V. Sutherland	MIT OCW
Introduction to Partial Differential Equations	Jared Speck	MIT OCW
Theory of Probability	Scott Sheffield	MIT OCW
Optimization with Sparsity-Inducing Penalties	Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski	arXiv:1108.0775
The Stacks Project	The Stacks Project Authors	project site
Learning with Submodular Functions	Francis Bach	arXiv:1111.6453
Tensor Categories	Pavel Etingof, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik	MIT OCW
Introduction to the Theory of Computation	Michael Sipser	MIT OCW

J CORRECTION TO A SOURCE FORMULA

Proposition 10.3 of *Learning with Submodular Functions* contains a typographical omission in its ℓ_1 characterization of submodular minimization. With the source's notation

$$s_-(V) = \sum_{k \in V} \min\{s_k, 0\}, \qquad \|s\|_1 = \sum_{k \in V} |s_k|,$$

the printed proposition states

$$\min_{A \subseteq V} F(A) = \max_{s \in B(F)} s_-(V) = F(V) - \min_{s \in B(F)} \|s\|_1.$$

The final equality is missing factors of $1/2$. Indeed, the scalar identity $|x| = x - 2\min\{x, 0\}$ gives

$$\|s\|_1 = s(V) - 2s_-(V), \qquad s_-(V) = \tfrac{1}{2}\big(s(V) - \|s\|_1\big).$$

Since every $s \in B(F)$ satisfies $s(V) = F(V)$, the corrected formula is

$$\min_{A \subseteq V} F(A) = \tfrac{1}{2}F(V) - \tfrac{1}{2} \min_{s \in B(F)} \|s\|_1.$$

The audited Lean target uses this corrected statement. The accepted proof makes the normalization explicit through the helper lemma

```
l1Norm s = modularSetFunction s Finset.univ
   - 2 * modularSetFunction (negativePart s) Finset.univ,
```

then uses the defining base-polyhedron equality $s(V) = F(V)$ to derive the two $1/2$ factors. Thus the benchmark corrects a source typo rather than weakening or otherwise changing the intended mathematical result.

K Agent Behavior and Ablation Details

K.1 Informal-Source Ablation

Experimental design. We test the marginal value of informal mathematics once the formal task has already been fixed. In the full-source condition, the agent receives the Lean workspace together with the textbook material in `book/` and an explicit instruction to consult it. In the formal-workspace-only condition, the agent receives the same expert-written target statement and local definitions, but all source-specific prompt text are removed.

Paired outcomes and source use. Across the original 341-task source-enabled campaigns, Opus accesses the textbook in 323/341 selected traces, while Gemini's attempt cascade contains source-use evidence for 330/341 problems. For Opus, the paired outcomes are 198 successes in both conditions, 6 with-source-only successes, 2 without-source-only successes, and 135 failures in both conditions. Gemini has 126 successes in both conditions, 10 with-source-only successes, 7 without-source-only successes, and 198 failures in both conditions.

Among tasks solved in both conditions, the source does not consistently shorten the resulting Lean development. For Opus, the median with-source/without-source LOC ratio is 1.00: the source-conditioned version is shorter in 99/198 pairs, longer in 96, and equal in 3. For Gemini, the median ratio is 1.04: the source-conditioned version is shorter in 55/126 pairs, longer in 68, and equal in 3. These comparisons concern generated code length rather than mathematical or software quality.

Illustrative cases. Opus 5 solved each of the following four problems only when given the textbook. In every case, its trace shows that it read a relevant passage and used the mathematical structure of that passage in the successful formalization.

- **Nagell–Lutz theorem.** The source supplied the sequence of local valuation and formal-group lemmas needed to prove integrality and the final discriminant divisibility statement. The no-source run developed much of the same machinery but did not connect it into the required torsion argument.
- **Rouché's theorem.** Opus 5 followed the source's decomposition through the argument principle and a straight-line homotopy. The no-source run constructed the zero-counting machinery but stopped before proving its final connection to winding number.
- **Commutativity of Dunkl operators.** The successful run used the source's reduction through a commutator identity and the Jacobi identity, avoiding the classification argument pursued by the no-source run.
- **Universal splitting.** The source gave a three-part roadmap: representability by polynomial equations, formal smoothness by lifting matrix units, and faithful flatness through splitting over algebraically closed fibres. The successful run implemented all three; the no-source run stopped at the final faithful-flatness step.

Opus 5 also solved two problems only without the source:

- **Grading-element local finiteness.** The source-enabled run followed the textbook's quotient-module argument but stalled on two technical steps. The no-source run instead used local-finiteness saturations and generalized eigenspaces, which translated more directly into Lean.
- **Soft-thresholded SVD.** The source-enabled run followed a stationary-point argument whose relaxation lost an essential rank constraint. The no-source run instead orthogonalized the factors and completed the proof using scalar soft thresholding, convexity, and a fractional greedy bound.

K.2 Reasoning-Effort Sensitivity

Reasoning-effort controls allocate additional test-time computation within a model, but their labels are provider-specific and should not be compared as matched compute across model families. To measure within-model sensitivity, we rerun four models below their headline settings: GPT-5.6 Sol and Gemini 3.7 Flash at `medium`, and Opus 5 and Fable 5 with adaptive thinking at `high`.

The effect is sharply model dependent. Opus 5 drops only from 211/341 (61.9%) to 204/341 (59.8%), a decrease of 2.1 percentage points, while Fable 5 remains unchanged at 181/341 (53.1%). In contrast, Gemini falls from 179/341 (52.5%) to 136/341 (39.9%), a decrease of 12.6 percentage points, and GPT-5.6 Sol falls from 201/341 (58.9%) to 114/341 (33.4%), a decrease of 25.5 percentage points.

Lower reasoning effort therefore has little observed effect on Fable and only a modest effect on Opus, but causes a substantial loss of verified coverage for Gemini and Sol.

K.3 One-Shot and Tool-Ablation Details

GPT-5.6 Sol's six one-shot successes are finite-dimensional norm equivalence, Riesz representation, a convex-hull support identity, Jacobson density, the Lasso optimality conditions, and a Stacks open-refinement lemma. Gemini's three are universal representation by an isotropic quadratic form, the nonpositive curvature inequality for affine buildings, and existence of a projective-module surjection. The two success sets are disjoint, but all nine problems are solved by every model in the agentic leaderboard, including GPT-6 Astra, DeepSeek V4.1 Flash, and Kimi K3. Their corresponding full agentic runs require far fewer interactions than a typical success on the single-hole cohort: a median of 19 versus 102 turns for Sol and 17 versus 137.5 for Gemini. The one-shot comparison measures the complete one-shot and agentic systems rather than the effect of any single tool.

The REPL, Loogle, and build endpoints are heavily used in the three baseline runs underlying pass@3. Across 216 task–run observations, GPT makes 8,715 REPL calls, 18,459 Loogle calls, and 5,159 build-service calls; Gemini makes 33,209, 12,416, and 19,058, respectively. Every observation invokes build, and 214/216 GPT observations and 215/216 Gemini observations use both REPL and build. Tool use is therefore a central part of the baseline behavior rather than a rarely exercised option.

Tool group	GPT-5.6 Sol	Gemini 3.7 Flash
REPL	36 [17, 58] (99.1%)	106.5 [40.5, 217.8] (99.5%)
Loogle	62 [24, 125.3] (97.7%)	46.5 [18.8, 82] (97.2%)
Build	20 [9, 35.3] (100%)	69 [11.8, 138.5] (100%)

Table 12: Baseline tool calls per task–run, reported as median [first quartile, third quartile] across the 216 pass@3 observations for each model. Parentheses give the percentage of observations using the tool group.

Because Gemini trajectories are longer, raw calls per run should not be read as cross-model efficiency. Per 100 turns, GPT makes 23.7 REPL, 50.1 Loogle, and 14.0 build-service calls, compared with 30.7, 11.5, and 17.6 for Gemini. Failed runs use more tools because successful runs terminate early while failures commonly run to budget.

Filesystem search provides a fallback when specialized search is removed: median filesystem-read calls rise from roughly 7 per GPT full-tool run to 55 without Loogle and 85 with only file operations; for Gemini, they rise from 20 to 40 and 52, respectively. In the no-build condition, GPT reconstructs `lake build` through the REPL on 26/72 problems, illustrating why these treatments concern endpoints rather than abstract capabilities. Among the 34 problems solved by Gemini both with full tools and without the REPL, median task-level spending increases by approximately $1.84\times$ without the REPL. GPT shows no comparable increase, suggesting that the REPL primarily improves Gemini's efficiency rather than its aggregate success rate.

K.4 Alternative Agent Scaffold

Claude Code uses a hybrid tool stack rather than a wholly separate proving environment. Across its selected traces, 60.6% of tool calls go to the benchmark-provided MCP services for the Lean REPL, Loogle, build, and report submission; the remaining 39.4% use Claude Code's native `Read`, `Glob`, `Grep`, `Edit`, and `Write` operations. It invokes the REPL on 312/341 problems, Loogle on 301, and build on 317.

In 18 of the 20 Claude-only successes, both configurations follow the same mathematical roadmap, but the common loop stops with only one or two target holes open. Claude Code's accepted developments have median 1,402 added lines in these cases, compared with 415 in the common loop's partial submissions; it performs more persistent editing while making fewer REPL and Loogle calls. For example, on Bézout's theorem the common loop makes 55 Loogle calls and adds 442 lines but leaves the homological core unfinished; Claude Code uses 38 Loogle calls and adds 1,740 lines to complete the Hilbert–Koszul and local-intersection-algebra development. Claude Code has lower recorded

spending on all 20 problems (median $56 versus $120) and uses fewer turns on 17/20 (median 235 versus 305).

K.5 Persistence Intervention

Early in our experiments, agents could end a run by reporting a problem `unsolved` after using only a small fraction of their budget. We therefore introduced the 10% continuation rule described in Appendix B. In the canonical leaderboard exports, more than one valid `submit_report` call identifies a selected trajectory whose first surrender was rejected. Table 13 reports how often this occurred and how often the additional work eventually produced a verified proof.

Model	Forced to continue	Eventually passed	Conversion
GPT-6 Astra	13	12	92.3%
Opus 5	3	0	0%
GPT-5.6 Sol	88	17	19.3%
Fable 5	2	0	0%
DeepSeek V4.1 Flash	198	39	19.7%
Gemini 3.7 Flash	199	41	20.6%
Kimi K3	84	3	3.6%
Sonnet 5	114	6	5.3%
Opus 4.8	134	0	0%
Muse Spark 1.3	62	0	0%

Table 13: Effect of requiring agents to continue after an early `unsolved` report in the selected headline trajectories. Conversion is the fraction of forced continuations that eventually pass clean-room verification.

Continued work recovers proofs for Astra, Gemini, DeepSeek, Sol, Sonnet, and Kimi, but none for Opus 5, Opus 4.8, Fable, or Muse. The intervention is therefore most useful for models that tend to stop before exhausting their productive search, and offers little benefit when an `unsolved` report already reflects a stable failure.

Impact on headline scores. The canonical leaderboard exports record the number of valid `submit_report` calls made in each selected trajectory. A verified run with more than one call necessarily submitted `unsolved` first: a `solved` report terminates immediately. We therefore mechanically truncate each trajectory at its first surrender to obtain Table 14. Without enforced continuation, Astra would fall from 281 to 269, DeepSeek from 180 to 141, Gemini from 179 to 138, and GPT-5.6 Sol from 201 to 184. The gains show that sustained iteration on compiler and tool feedback is itself an important form of test-time scaling.

Model	Final score	Later recoveries	Stop-at-surrender score
GPT-6 Astra	281	12	269
Opus 5	211	0	211
GPT-5.6 Sol	201	17	184
Fable 5	181	0	181
DeepSeek V4.1 Flash	180	39	141
Gemini 3.7 Flash	179	41	138
Kimi K3	165	3	162
Sonnet 5	152	6	146
Opus 4.8	150	0	150
Muse Spark 1.3	125	0	125

Table 14: Headline verified coverage with the continuation intervention and after counterfactually terminating each selected trajectory at its first valid `unsolved` report.

Model	Budget consumed	Model gave up
GPT-6 Astra	21	39
Muse Spark 1.3	92	124
GPT-5.6 Sol	10	130
Sonnet 5	4	185
Kimi K3	3	173
Opus 5	2	128
Gemini 3.7 Flash	2	160
Fable 5	0	160
DeepSeek V4.1 Flash	0	161
Opus 4.8	0	191
All models	134	1,451

Table 15: Termination reasons among unsuccessful selected leaderboard trajectories. Muse accounts for 68.7% of all budget-consumed runs.

K.6 Self-Report Accuracy

Agents generally assess their own success accurately, but their reports are not perfect substitutes for independent grading. Table 16 compares each recorded final `solved`/`unsolved` report with the clean-room verdict. Of 3,301 recorded reports, 3,289 agree with the grader. All 12 disagreements are false claims of success; no model reports `unsolved` for a submission that Comparator accepts. A further 109 rows have no recorded final report and are shown separately rather than counted as correct or incorrect.

Model	Correct/recorded	False `solved`	No report	Accuracy
GPT-6 Astra	339/341	2	0	99.41%
Opus 5	341/341	0	0	100.00%
Opus 4.8	341/341	0	0	100.00%
GPT-5.6 Sol	340/341	1	0	99.71%
Gemini 3.7 Flash	336/340	4	1	98.82%
Sonnet 5	340/341	1	0	99.71%
Fable 5	243/243	0	0	100.00%
DeepSeek V4.1 Flash	340/341	1	0	99.71%
Kimi K3	339/340	1	1	99.71%
Muse Spark 1.3	330/332	2	9	99.40%
All models	3,289/3,301	12	109	99.64%

Table 16: Agreement between recorded agent self-reports and clean-room grading. Accuracy is conditional on a recorded final report.