LEANIFY: VERIFIED PROOF COMPRESSION AND REUSABLE LEMMA DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models can prove difficult mathematical statements, but their proofs still need to be organized for understanding and reuse. It has been argued that mathematics advances by compression: concepts and theorems turn long deductions into reusable arguments. We formulate proof compression as a verifiable task: shorten Lean proof collections while preserving their original statements. Our agents compress the Lean proofs of the permanent lower bound and of the counterexamples to the compactness conjecture from OpenAI's Ten Advances in Mathematics and Theoretical Computer Science by 94.2% and up to 99.2%, respectively, under our proof-term size metric. For the permanent proof, compression also reduces the number of required intermediate theorems and shortens model-generated mathematical explanations. Using a lemma-discovery pipeline, we observe two scaling trends: later compression stages yield a larger share of lemmas judged non-trivial by a language model, and compressing more related Fermat's Last Theorem results together yields wider lemma reuse. These trends suggest that scaling compression may uncover more reusable mathematical arguments. We therefore release Leanify as a benchmark to test this possibility at scale.

1 INTRODUCTION

From Olympiad problems (Hubert et al., 2026) to long-standing open problems (OpenAI, 2026b;a; Alpöge, 2026), AI-generated mathematics has entered an era of abundance. Yet mathematical progress depends not only on solving new problems but also on digesting existing results: finding simpler arguments, isolating ideas that several proofs share, and restating them as results others can understand and build on (Riehl, 2026). Tao (2026) observes that solving a designated problem can leave the accompanying insights and new directions unexplored. With proofs now abundant, the bottleneck shifts to digesting them.

Unlike proving a theorem, digesting a result has no clear criterion of success, which makes it hard to assign to an AI agent. Compression gives part of this work such a criterion. As Aksenov et al. (2026) argue, mathematicians introduce definitions and theorems that package long chains of reasoning into reusable "macros." The notion of a group, for instance, captures the structure shared by addition on the real numbers, arithmetic modulo 12, and the symmetries of a square (Figure 1), so that one theorem replaces separate arguments in each setting. Their analysis of Mathlib quantifies this: the fully unwrapped length of a declaration grows exponentially with its definitional depth, while its wrapped length stays roughly constant, and the longest declaration unwraps to about 10^{104} primitive terms. We therefore study proof compression: given a collection of formal proofs, an agent keeps the target statements fixed and searches for shorter proofs, possibly introducing lemmas that several proofs share. Lean checks that each shortened proof still establishes its original statement.

Compression would be most useful if it did more than remove redundancy. A shorter proof may follow a simpler argument, and a lemma that shortens several proofs may isolate an idea they share. Such changes would make formal mathematics easier to build on, both for mathematicians and for agents that reason within a bounded context. Ultimately, compression could turn an abundance of proofs into an abundance of insight: arguments reduced to their essential ideas, and shared lemmas that become the building blocks of new theory.

To study this possibility, we need an objective that rewards agents for reorganizing existing mathematical reasoning while preserving its conclusions. But not every notion of length will do. A

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.

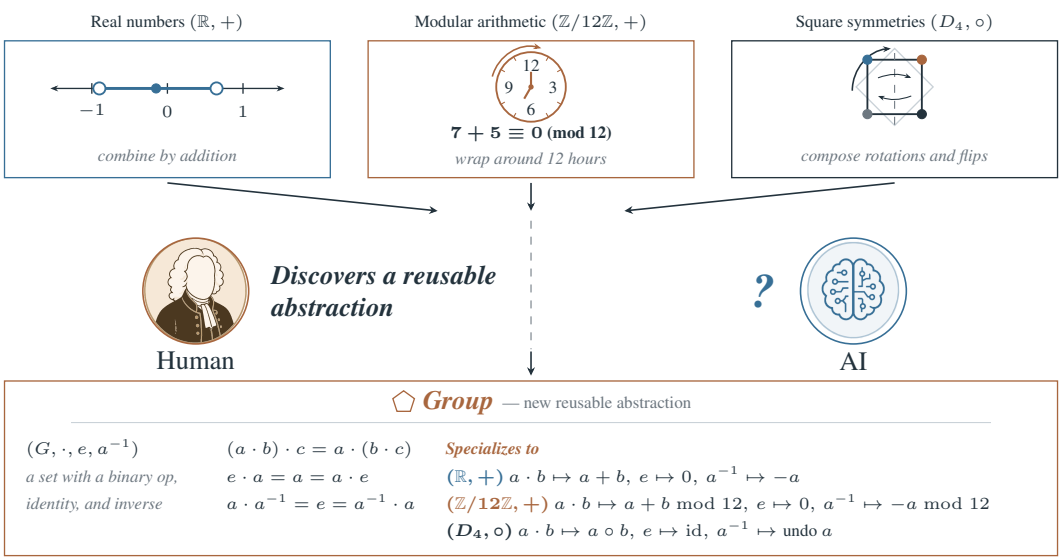


Figure 1: A group captures structure shared by three settings, so one argument about groups applies to all of them.

natural-language proof can compress an argument into a phrase, skip it with "obviously", or expand it at will, so shorter prose need not mean less mathematics. A formal proof, e.g., Lean tactic code, has a better-defined notion of length, yet its source length can still hide work: a single call to `omega`, `linarith`, or `grind` launches an arbitrary search behind a few characters. Under the Curry–Howard reading (Howard, 1980) on which Lean is built, the proof is the term the kernel checks, not the script that produced it. We therefore measure, and minimize, the size of that proof term.

Our experiments suggest why this objective deserves study at a larger scale. Consider the formal proof of a lower bound for arithmetic formulas computing the permanent (OpenAI, 2026b). Over 30 hours of search, agents reduce its proof size from 85,690,155 to 4,967,346 (94.2%), and the number of theorems it recursively depends on falls from 552 to 319. Some of these savings change the argument itself. For example, the original proof shows that certain polynomials are algebraically independent and then applies an algebraic-matroid argument to bound a number of parameters; the compressed proof obtains the same bound directly from the linear independence of their derivatives (Appendix G). As compression proceeds, the helper lemmas (helpers) our agent proposes also change: a language-model judge rates 61% of those found from the original proof as routine, and none at the last two checkpoints. As we rewrite more related Fermat's Last Theorem results (Buzzard & Taylor, 2024) together, the lemmas are reused more widely: with 32 joint targets, a single lemma helps up to 20 of them. These trends raise the question of what compression would find across all major theorems formalized in Lean. Such a search exceeds our compute, so we release the task as a benchmark.

Our contributions can be summarized as follows:

- **Objective and benchmark.** We define abstraction discovery as minimizing the size of Lean kernel proof terms[1]: a lemma counts only if the savings it produces exceed its own cost. We release this as a benchmark for comparing agents (model plus harness).
- **Lemma discovery.** Our agent proposes lemmas, proves them in Lean, and rewrites target proofs to use them. We analyze 793 accepted helper instances from the Fermat's Last Theorem formalization, each with a mathematical-significance rating, a measured statement-and-proof cost, and recorded reuse.
- **Scaling experiments.** Compressing the proof of a lower bound for arithmetic formulas computing the permanent (OpenAI, 2026b) by 94.2% reduces its logical dependencies and shortens its natural-language exposition. Further compression and larger joint target sets yield more non-trivial and more widely reused lemmas.

[1]Throughout, proof size counts AST nodes in the kernel-checked proof terms and required auxiliary declarations (Section 2.1).

2 PROOF COMPRESSION

Proof compression turns the open-ended search for better ways to organize proofs into a concrete optimization problem: prove the same set of target statements with a smaller proof term. This matches how mathematicians work: when several proofs share an argument, they extract it as a lemma and apply it to each case. Compressing targets jointly thus searches for connections among them without fixing in advance which intermediate results to find.

Early savings may come mostly from removing redundancy. Near the optimum, we expect compression to shorten the mathematics itself and extract useful abstractions.

2.1 WHY COMPRESS PROOF TERMS

Prior work (Appendix F) has optimized proof scripts and learned reusable tactics and lemmas (Hetzl et al., 2014; Kaliszyk & Urban, 2013; Poesia & Goodman, 2023; Wang et al., 2024; Zhou et al., 2024; Wernhard & Zombori, 2025; Fu et al., 2026). TacMiner discovers tactic libraries that compress Rocq proof scripts (Xin et al., 2025b). ProofOptimizer, ImProver, and Lean Refactor shorten individual Lean proofs by token or tactic count (Gu et al., 2026; Ahuja et al., 2025; Lu et al., 2026). ImProver 2 minimizes tactic count in its length experiments and separately studies explicit dependencies and modularity (Ahuja et al., 2026). These metrics suit proof refactoring. For our purpose, measuring the length of code or tactics has a problem.

A short proof script often only describes how to search for a proof.[2] Its length therefore depends on the automation available. A stronger or more specialized tactic shortens the script and leaves the argument unchanged. In the extreme, a single tactic that enumerates and checks candidate proofs would prove every provable statement with the same short script, much like a short *program* in the sense of Kolmogorov complexity (Li & Vitányi, 2019). The proof term records the argument the kernel actually checks, whichever tactic produced it.

We instead measure the proof itself. Lean's kernel checks a *proof term*, an explicit expression whose type is the theorem, and accepts a tactic's output only after this check (de Moura & Ullrich, 2021). The *proof term* we measure consists of the targets' proof terms and the statement and proof of every non-background declaration that the target statements or proofs use, directly or indirectly (Figure 2). Each such declaration is counted once, and each use of it adds a short reference. Moving an argument into a helper therefore does not make it free. We measure proof size by counting nodes in the abstract syntax trees (ASTs) of the proof terms and required auxiliary declarations, including universe-level nodes and excluding expression metadata. Names, indices, and literal payload lengths add no nodes. Shared declarations contribute once; repeated subterms contribute at each occurrence unless explicitly shared through a binding or helper. Appendix A defines the counting rule. Checking a proof term needs no search, though kernel reduction can still be costly (The Lean Team, 2026). Checking time, readability, and mathematical significance are separate questions.

2.2 COMPRESSION OBJECTIVE

Fix a background library B and target statements T. Let the proof collection P contain the proofs of every target and all auxiliary declarations required by their statements and proofs. A valid collection must pass Lean kernel verification against B, preserve the target statements, and respect the task's edit permissions (Section 3.1). We seek to minimize the length of a valid proof collection, given the fixed background library and target statements:

$$\min_{P \text{ valid for } T} L_B(P),$$

where $L_B(P)$ is the proof size relative to B, as defined in Equation (A.1). Within the task's permissions, intermediate declarations may be introduced, rewritten, or removed.

Because $L_B(P)$ includes each new lemma's statement and proof, a lemma lowers $L_B(P)$ only when the proofs it shortens save more than the lemma itself costs.

[2] For example, `by omega` closes many linear-arithmetic goals over integers and natural numbers with the same eight characters, while the proof term it builds grows with the goal.

Section 4.2 describes how our lemma-discovery system searches for such improvements through lemma proposal and proof rewriting.

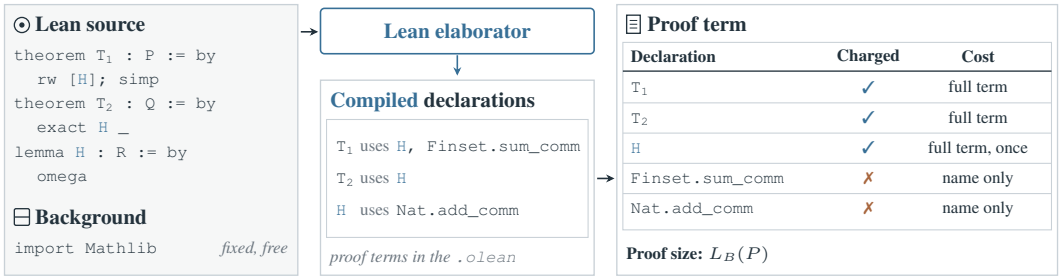


Figure 2: Proof-term accounting. **Left:** Lean tactics build proof terms, which the kernel checks. **Middle:** Compiled `.olean` files store these terms, and targets T_1 and T_2 both use the helper H. **Right:** We count the proof terms of T_1 and T_2 and the statement and proof of H, once. Each use of H or of a background lemma adds a short reference, and background contents are excluded. H pays off when it shortens the targets by more than its own size. Code is schematic.

3 A Benchmark for Verified Proof Compression

The benchmark turns the objective of Section 2.2 into a task: given target statements, a fixed background, and the original proofs, an agent must submit shorter verified proofs of the same targets within a search budget. Any model and harness can be compared this way.

3.1 Task Definition

Input. Each instance provides a Lean checkout with target statements T, a fixed background library B, and original proofs P_0. The original proofs give the agent a starting solution and provide the baseline for scoring. The task specifies which intermediate declarations may be edited. The target statements, the declarations determining their meanings, and the allowed assumptions remain fixed.

Submission. The agent edits the Lean source to prove every target, rewriting proofs and introducing, rewriting, or removing intermediate declarations within the permitted scope. It compiles the edited modules and submits their `.olean` files together with the manifest and companion artifacts required by the pinned Lean version. The evaluator reads declarations and proof terms from these compiled artifacts. Source changes are retained for reproduction. Appendix E.1 gives the submission protocol and illustrates the task in Figure 8.

Choice of targets. The choice of rewrite targets determines which results the agent must preserve. If we select only the final theorems, the agent can replace intermediate results and reorganize their proofs. If we retain all theorems as targets, the agent must consider the full collection and can seek shared arguments that shorten several proofs together. A richer background can reduce the work needed for each target, while shared intermediate results can reduce the cost of proving several targets together.

3.2 Verification and Scoring

Verification. Every target must have a valid proof of its original statement. We check the submitted proofs and their required declarations using Lean's kernel against the fixed background. We also check that the definitions determining the targets' meanings and the allowed assumptions remain unchanged. A submission is rejected if any target lacks a proof or if it adds axioms or references declarations outside the task environment.

Measurement. We measure the original and submitted collections by their proof size L_B (Section 2.2), with the same counting rule and background for both.

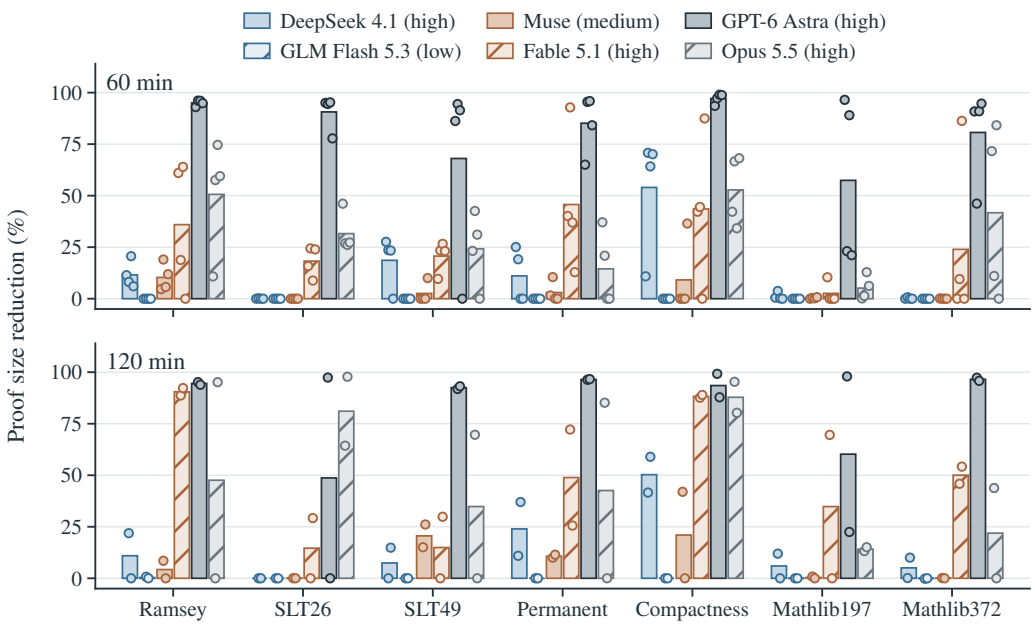


Figure 3: Proof size reduction with the original proofs available, under 60-minute (top) and 120-minute (bottom) search budgets. Each bar averages four searches at 60 minutes and two at 120 minutes, counting searches without a valid endpoint as zero. Dots show individual searches. Legend labels give reasoning effort. Appendix E gives sample counts and inclusion rules.

Score. The score is the percentage reduction in size from the original collection P_0 to a valid submission P:

$$100\left(1-\frac{L_B(P)}{L_B(P_0)}\right).$$

A larger valid submission receives a negative score. Invalid submissions and attempts that end without a completed evaluation receive zero.

3.3 Experimental Setup and Results

We evaluate proof compression on the Ramsey-number and Permanent formalizations and the formalization of the compactness conjecture in extremal graph theory (Compactness) released by OpenAI (2026b), the statistical learning theory (SLT) library of Zhang et al. (2026), and two collections of Mathlib leaf theorems (The mathlib Community, 2020). The Mathlib collections contain 197 and 372 target theorems from 10 and 20 import-leaf modules across algebra, calculus, category theory, manifolds, and measure theory. Table 5 in Appendix E lists the target counts and original proof sizes for the instances in the results.

The run configuration specifies the model, harness, tools, information available to the agent, and search budget. The benchmark accepts a configurable search-time budget. The experiments reported here use 60 and 120 minutes. The task supports both general proof editing and systems that search for reusable lemmas. Section 4.2 describes our lemma-discovery system.

Results. We give agents the original proofs and evaluate them under these search budgets. Figure 3 reports mean proof-size reductions at 60 and 120 minutes over all assigned searches, counting searches without a valid endpoint as zero. An endpoint is the final submission of one search. We evaluate DeepSeek 4.1, GLM Flash 5.3, Muse, Fable 5.1, GPT-6 Astra, and Opus 5.5, each in its own agent harness. Each budget starts separate searches and permits early stopping. Table 6 gives the 60-minute values and sample counts, and Appendix E explains the aggregation.

Permanent protects its division-free and rational-formula lower bounds. Compactness protects five declarations: the quantitative and asymptotic compactness counterexamples, the two-degenerate extremal counterexample, and the negations of Erdős problems 180 and 146. Ramsey protects all 96

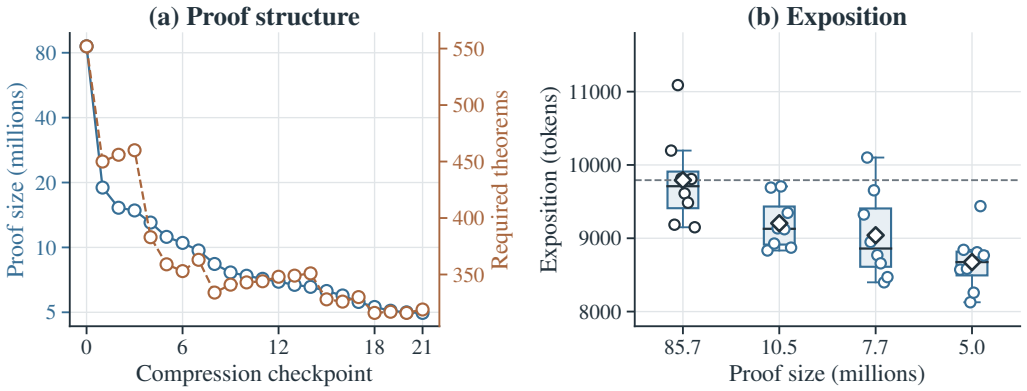


Figure 4: Compressing the Permanent lower-bound proof (OpenAI, 2026b). Proof size falls from 85,690,155 to 4,967,346, and recursively required theorems from 552 to 319. Step 0 denotes the original proof; each subsequent step denotes a verified checkpoint that improves on the smallest proof size previously recorded. Left: proof size and recursively required theorems across compression checkpoints, using separate vertical axes (logarithmic for proof size). Right: exposition lengths for eight independent translations of each of four proof versions, grouped by proof size. Boxes show quartiles and diamonds mark means. The dashed horizontal line is the original mean.

statements, including intermediate constructions, and the SLT and Mathlib tasks protect each selected theorem (Appendix E.2).

Compression across target collections. Completing a formal proof can leave substantial room for simplification, both within individual results and across theorem collections. At 120 minutes, GPT-6 Astra's mean reduction is 93.5% on Compactness's five targets (best run 99.2%) and 96.5% on Permanent's two targets. Substantial compression is also possible across many targets: on the 372-target Mathlib collection, GPT-6 Astra's mean reduction is 96.6% at 120 minutes. The 197-target collection gives a different picture, with mean reductions of 57.4% at 60 minutes and 60.2% at 120 minutes. Compressibility thus varies across collections and budgets, and target count alone does not explain it.

Section 4 examines what compression changes through direct proof rewriting and a separate lemma-discovery procedure, studying searches from more compressed proofs and across broader target collections.

4 What Compression Finds

Section 3 measures how much agents can shorten formal proofs. We now ask what changes within those proofs and what reusable arguments further compression can reveal. In *direct compression*, an agent may reorganize a proof freely while preserving its targets. The resulting changes can combine new lemmas, definitions, and rewritten arguments. To study reusable lemmas explicitly, we also use *lemma-based compression*: propose a helper, prove it, and test whether the target proofs it shortens save more than its statement and proof cost. Each retained helper is a single lemma whose content and reuse we can inspect. We study this search in two directions: *deeper*, starting from increasingly compressed proofs, and *broader*, considering more related targets together.

4.1 Direct Compression: How Far Proofs Shrink

Longer search. We first ask how much additional search helps agents shorten proofs across benchmark tasks. Figure 3 compares separate searches under different time budgets. On Compactness, Fable 5.1's mean reduction rises from 43.5% at 60 minutes to 88.3% at 120 minutes. Gains also occur in library tasks: Muse's SLT49 reduction rises from 2.5% to 20.6%.

Fewer intermediate results. Over 30 hours of search, the Permanent lower-bound proof shrinks by 94.2%, and the number of theorems its two targets recursively depend on falls from 552 to

319 (Figure 4, left). The retained Lean source is also shorter after removing unused declarations. The argument itself also changes. To bound the number of parameters a formula supplies, the original proof shows that certain coefficient functions are algebraically independent and then applies an algebraic-matroid argument. The compressed proof replaces both steps with one lemma: if r functions lie in a field generated by t elements and have linearly independent derivative vectors, then $r \leq t$. Its proof uses only the rules of differentiation and linear algebra. A later revision proves both targets with one parameter bound. It represents each intermediate value as $x = (au + b)/(cu + d)$ and uses Lean's convention $z/0 = 0$, so formulas with and without division share one argument. Appendix G describes these and two other changes.

Shorter exposition. We translate seven versions of the proof into mathematical prose, eight times each. Each translation is a fresh one-hour session of GPT-6 Astra with high reasoning effort, given the same instructions and the published Permanent chapter as reference (Appendix H). Every compressed version gives shorter expositions on average, by 5.3–11.4%, and the final version is 11.4% shorter (Figure 4, right, shows four versions). This suggests using computation to simplify a proof before asking mathematicians to read it. Whether the shorter expositions stay faithful and read more easily requires further review.

The direct searches show how a complete proof changes. We next search for shared lemmas explicitly.

4.2 Lemma-Based Compression: Finding Shared Arguments

Procedure. Each round selects related target theorems T_t, proposes a helper lemma ℓ, proves it in Lean, and attempts shorter proofs using it. Let $\Delta_t(\ell, \varphi)$ be the saving of a helper-using proof of φ over its lemma-free counterpart, and let $\mathrm{Cost}_t(\ell)$ be the size of the helper's statement and proof. The helper is retained only when

$$\sum_{\varphi \in T_t} \Delta_t(\ell, \varphi) > \mathrm{Cost}_t(\ell). \tag{4.1}$$

The workflow and accounting scopes appear in Appendix C. Table 2 and Appendix D.1 compare retrieval methods and model assignments.

Deeper: lemmas from more compressed proofs. We ran 12 lemma-discovery searches per checkpoint in each of two batches, with at most 20 model calls per search. The 228 searches produced 170 helpers across 13 distinct proof versions (Appendix E.8). We ask GPT-6 Pro to score helpers for mathematical significance on a 0–10 rubric with checkpoint identities hidden. Scores 0–2 denote restatement, definition unfolding, or routine manipulation. Scores 3–4 denote a modest, identifiable mathematical observation, and scores 5–6 a substantive observation or non-obvious explanatory mechanism. The rubric and scoring prompt appear in Appendices E.8 and E.8.1.

The non-trivial share (score at least 3) rises from 39.1% for the original proof to 75.0–100% at the last five checkpoints (Figure 5a). We interpret this trend as follows. The original proof contains routine redundancy, and 60.9% of the helpers found from it score at most 2. Once compression removes this redundancy, further savings require reorganizing the argument itself, which favors helpers that state a mathematical observation.

A separate three-version comparison gives a concrete example. Searches from the original proof proposed variants of matrix-row factorization. Searches from a compressed version found a shared induction over finite-set deletion. If $P_\varnothing = 1$ and $P_S = c(S) - \sum_{a \in S} X^{w(a)} P_{S \setminus \{a\}}$, then P_S has degree at most $\sum_{a \in S} w(a)$ and its coefficient at that degree is $(-1)^{|S|}|S|!$, whatever the constants $c(S)$. Four successful searches supported rewrites of both main Permanent theorems. This experiment holds the callable library and sixteen target statements fixed (Appendix E.9).

Broader: lemmas shared by more theorems. We use one fixed, ordered collection of targets from the Frey curves and level-lowering development of Fermat's Last Theorem (FLT). For each $K \in \{2, 4, 8, 16, 32\}$, the first K targets serve as both proof context and rewrite targets, so the target sets are nested. We run six fresh lemma-discovery searches on each fixed set. The repetitions reuse the same targets (Appendix E.5). Each search retains at most one helper. At $K = 32$, a search helps 14.67 distinct targets on average, compared with 1.17 at $K = 2$ (Figure 5b). Larger joint tasks therefore show wider reuse.

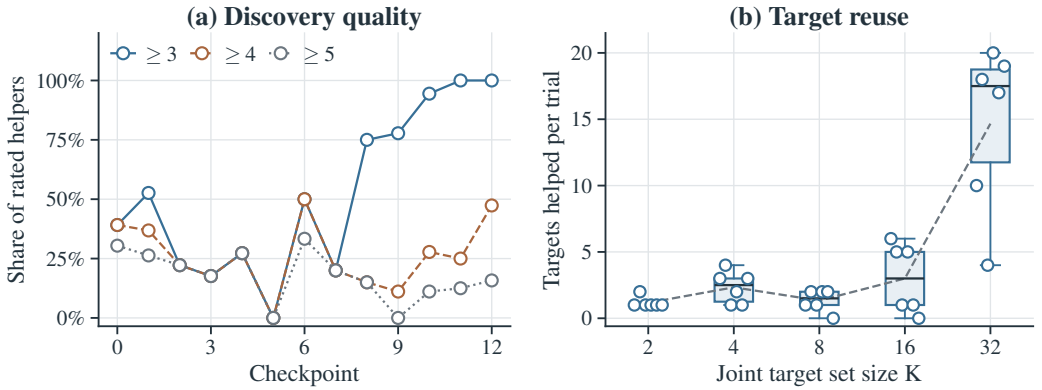


Figure 5: Discovery quality and target reuse in two experiments. (a) Permanent: percentage of rated helpers scoring at least 3, 4, or 5 for mathematical significance. Checkpoint 0 is the original proof. Subsequent checkpoints follow compression order. (b) An FLT subset (Frey): distinct targets helped in each of six trials per joint target set size K. Trials with no accepted helper contribute zero. Boxes show quartiles and dashed lines connect all-trial means. The scoring prompt and trial accounting appear in Appendices E.8.1 and E.5.

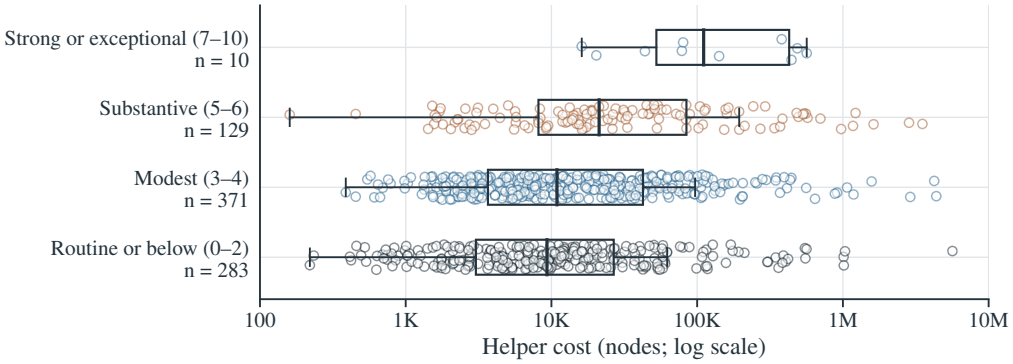


Figure 6: Helper cost by mathematical-significance rating for all 793 instances in the fixed FLT cohort. Each dot is one helper; boxes span the quartiles, center lines mark medians, and whiskers reach the farthest dot within 1.5 times the interquartile range. Cost is the node count of the helper's statement and proof, on a logarithmic axis. Ratings use the original 0–10 rubric and supplied mathematical context; repeated mathematical ideas are retained. Small vertical offsets separate overlapping points.

Our cohort of 793 rated FLT helper instances relates helper size to mathematical significance: the highest-rated band contains ten instances and has a median cost of 110,802 expression nodes, compared with 9,322 for the routine band. Figure 6 and Appendices E.6 and E.7 give the full analysis. A costly helper can pay for itself when the total savings on its uses exceed its cost. Larger target sets give such a helper more proofs to shorten. Appendix B gives example lemmas, such as a multivariate chain rule for derivations, $D(\varphi(p)) = \sum_i \varphi(\partial p/\partial X_i) \cdot D(\varphi(X_i))$, which generalizes Mathlib's one-variable `Derivation.map_aeval`.

Conjecture: deeper and broader compression favors shared abstractions

Sustained compression may reduce opportunities for routine simplification, while larger target collections offer more opportunities to reuse substantial arguments. Pushing compression further along both directions may therefore uncover useful shared abstractions across larger bodies of mathematics.

5 CONCLUSION

We formulate proof compression as a benchmark for simplifying established mathematics while preserving its results. Direct compression reveals substantial room to shorten verified proofs. The Permanent study also finds fewer intermediate results and shorter generated expositions. Lemma-based compression lets us examine reusable arguments found from more compressed proofs and broader target collections. Later checkpoints yield a larger share of helpers rated non-trivial, while larger joint tasks yield wider absolute reuse. These observations motivate testing deeper and broader compression at larger scales. We hope the benchmark helps develop methods that let mathematicians and agents build on existing results.

Limitations. Our experimental design is shaped by the compute available to us. Lemma-based compression is a probe of what compression finds. It fixes the target set and admits one helper at a time, which lets us credit each saving to a single lemma within a limited budget. Compression at larger scale may find abstractions that this controlled procedure misses. The same budget limits us to one extended compression trajectory, of the Permanent proof, and to one part of the FLT formalization for the reuse study. Our analysis of helper quality relies on ratings from a language model, since expert review of hundreds of helpers was beyond our resources, and no mathematician has reviewed the fidelity of the generated expositions. The benchmark runs use search budgets of 60 and 120 minutes. Reductions still grow with the budget on several tasks, so these runs do not show how far agents can ultimately compress a proof, and measuring that limit requires longer searches with more test-time compute than we could afford. Finally, we do not compare against compressing tactic scripts. Such a comparison at the scale of our kernel-level searches would require compute we did not have.

AI USE STATEMENT

Generative AI is part of the object of study in this work: large language model agents perform the proof compression we evaluate (Sections 3 and 4.1), and our lemma-discovery pipeline uses them to propose candidate lemmas, write Lean proofs, and rewrite target theorems (Section 4.2). Every artifact produced by these agents is checked by the Lean kernel before it is counted, and all reported compression numbers come from our evaluator, which measures kernel-checked proofs.

Generative AI also assisted with manuscript editing and analysis code. A language model produced the helper-quality ratings reported in Section 4.2 and Appendices E.7 and E.8, and wrote the proof expositions measured in Section 4.1. Language models also helped draft and revise sections of this paper and find related work. The authors reviewed all text and citations and take full responsibility for the content.

REPRODUCIBILITY STATEMENT

Appendix E specifies verification and scoring. Appendix C describes the lemma-discovery workflow, and Appendix H gives the translation protocol.

REFERENCES

Riyaz Ahuja, Jeremy Avigad, Prasad Tetali, and Sean Welleck. ImProver: Agent-based automated proof optimization. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.04753.

Riyaz Ahuja, Tate Rowney, Jeremy Avigad, and Sean Welleck. ImProver 2: Iteratively self-improving LMs for neurosymbolic proof optimization. *arXiv preprint arXiv:2605.22885*, 2026. URL https://arxiv.org/abs/2605.22885.

Vitaly Aksenov, Eve Bodnia, Michael H. Freedman, and Michael Mulligan. Compression is all you need: Modeling mathematics. *arXiv preprint arXiv:2603.20396*, 2026. URL https://arxiv.org/abs/2603.20396.

Yousef Alhessi, Sólrún Halla Einarsdóttir, George Granberry, Emily First, Moa Johansson, Sorin Lerner, and Nicholas Smallbone. Lemmanaid: Neuro-symbolic lemma conjecturing. *arXiv preprint arXiv:2504.04942*, 2025. URL https://arxiv.org/abs/2504.04942.

Levent Alpöge. A counterexample to the Jacobian conjecture. https://x.com/__alpoge__/status/2079028340955197566, July 2026. Post on X announcing an explicit three-dimensional counterexample found with Claude Fable 5.

Maissam Barkeshli, Michael R. Douglas, and Michael H. Freedman. Artificial intelligence and the structure of mathematics. *arXiv preprint arXiv:2604.06107*, 2026. URL https://arxiv.org/abs/2604.06107.

Abhiram Bellur, Razan Alghamdi, Kidus Workneh, and Joseph Izraelevitz. Leroy: Library learning for imperative programming languages. In *Human Aspects of Types and Reasoning Assistants*, 2024. URL https://arxiv.org/abs/2410.06438.

Ian Berlot-Attwell, Frank Rudzicz, and Xujie Si. Library learning doesn't: The curious case of the single-use "library". In *MATH-AI Workshop at NeurIPS*, 2024. URL https://arxiv.org/abs/2410.20274.

Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023. doi: 10.1145/3571234. URL https://arxiv.org/abs/2211.16605.

Kevin Buzzard and Richard Taylor. Towards a Lean proof of Fermat's Last Theorem. https://imperialcollegelondon.github.io/FLT/, 2024.

David Cao, Rose Kunkel, Chandrakana Nandi, Max Willsey, Zachary Tatlock, and Nadia Polikarpova. babble: Learning better abstractions with E-graphs and anti-unification. *Proceedings of the ACM on Programming Languages*, 7(POPL):396–424, 2023. doi: 10.1145/3571207.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, Shufa Wei, Yonghui Wu, Yuchen Wu, Yihang Xia, Huajian Xin, Fan Yang, Huaiyuan Ying, Hongyi Yuan, Zheng Yuan, Tianyang Zhan, Chi Zhang, Yue Zhang, Ge Zhang, Tianyun Zhao, Jianqiu Zhao, Yichi Zhou, and Thomas Hanwen Zhu. Seed-Prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025. URL https://arxiv.org/abs/2507.23726.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe (eds.), *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635, Cham, 2021. Springer. doi: 10.1007/978-3-030-79876-5_37.

Gabriel Ebner, Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. On the generation of quantified lemmas. *Journal of Automated Reasoning*, 63:95–126, 2019. doi: 10.1007/s10817-018-9462-8.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sablé-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Growing generalizable, interpretable knowledge with wake–sleep Bayesian program learning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251):20220050, 2023. doi: 10.1098/rsta.2022.0050.

Yiming Fu, Peixuan Liu, Zichen Wang, and Kun Yuan. Proof-Refactor: Refactoring generated formal proofs into modular artifacts. *arXiv preprint arXiv:2606.03743*, 2026. URL https://arxiv.org/abs/2606.03743.

Gabriel Grand, Lionel Wong, Maddy Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.19791.

Peter D. Grünwald. *The Minimum Description Length Principle*. Adaptive Computation and Machine Learning. MIT Press, 2007. ISBN 9780262072816. doi: 10.7551/mitpress/4643.001.0001.

Alex Gu, Bartosz Piotrowski, Fabian Gloeckle, Kaiyu Yang, and Aram H. Markosyan. ProofOptimizer: Training language models to simplify proofs without human demonstrations. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=huptrb4JTa.

Stefan Hetzl. Project presentation: Algorithmic structuring and compression of proofs (ASCOP). In *Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pp. 438–442. Springer, 2012. doi: 10.1007/978-3-642-31374-5_32. URL https://www.dmg.tuwien.ac.at/hetzl/research/ascop.pdf.

Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. Algorithmic introduction of quantified cuts. *Theoretical Computer Science*, 549:1–16, 2014. doi: 10.1016/j.tcs.2014.05.018.

William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Cezary Kaliszyk and Josef Urban. Lemma mining over HOL Light. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov (eds.), *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19)*, volume 8312 of *Lecture Notes in Computer Science*, pp. 503–517. Springer, 2013. doi: 10.1007/978-3-642-45221-5_34.

Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science. Springer, Cham, fourth edition, 2019. ISBN 978-3-030-11298-1. doi: 10.1007/978-3-030-11298-1.

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. In *Conference on Language Modeling*, 2025.

Jialin Lu, Soonho Kong, Rodrigo Stehling, Kaiyu Yang, Zhangyang Wang, Weiran Sun, and Wuyang Chen. Lean Refactor: Multi-objective controllable proof optimization via agentic strategy search. *arXiv preprint arXiv:2605.20244*, 2026. URL https://arxiv.org/abs/2605.20244.

Roy L. McCasland, Alan Bundy, and Patrick F. Smith. MATHsAiD: Automated mathematical theory exploration. *Applied Intelligence*, 47(3):585–606, 2017. doi: 10.1007/s10489-017-0954-8.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

Naoto Onda, Kazumi Kasaura, Yuta Oriike, Masaya Taniguchi, Akiyoshi Sannai, and Sho Sonoda. Lean-Conjecturer: Automatic generation of mathematical conjectures for theorem proving. *arXiv preprint arXiv:2506.22005*, 2025. URL https://arxiv.org/abs/2506.22005.

OpenAI. On the Navier–Stokes Millennium Prize Problem. https://openai.com/index/navier-stokes-solution/, September 2026a. Announced 8 September 2026 with a Lean formalization.

OpenAI. Ten advances in mathematics and theoretical computer science. https://openai.com/index/ten-advances-in-mathematics/, August 2026b. Results of an internal version of Astra. Lean proofs at https://github.com/openai/ten-proofs.

Gabriel Poesia and Noah D. Goodman. Peano: Learning formal mathematical reasoning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251):20220044, 2023. doi: 10.1098/rsta.2022.0044.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025. URL https://arxiv.org/abs/2504.21801.

Emily Riehl. Why I do mathematical research. Guest post on Terence Tao's blog, *What's New*, September 2026. URL https://terrytao.wordpress.com/2026/09/14/why-i-do-mathematical-research/.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. ReGAL: Refactoring programs to discover generalizable abstractions. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 46605–46624. PMLR, 2024. URL https://proceedings.mlr.press/v235/stengel-eskin24a.html.

Terence Tao. In this modern era of heavy AI use, it is important to remember what it is like to be a child. Mathstodon, September 2026. URL https://mathstodon.xyz/@tao/117244104044239500.

The Lean Team. Validating a Lean proof. Lean Language Reference, 2026. URL https://lean-lang.org/doc/reference/latest/ValidatingProofs/. Accessed September 24, 2026.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pp. 367–381, New Orleans, LA, USA, January 2020. ACM. 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, Datasets and Benchmarks Track*, 2024.

Jiří Vyskočil, David Stanovský, and Josef Urban. Automated proof compression by invention of new definitions. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*, volume 6355 of *Lecture Notes in Computer Science*, pp. 447–462. Springer, 2010. doi: 10.1007/978-3-642-17511-4_25.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3f5PALef5B.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025. URL https://arxiv.org/abs/2504.11354.

Christoph Wernhard and Zsolt Zombori. Mathematical knowledge bases as grammar-compressed proof terms: Exploring Metamath proof structures. *arXiv preprint arXiv:2505.12305*, 2025. URL https://arxiv.org/abs/2505.12305.

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. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/b3b55c366d641c07180c40e4f978f311-Paper-Conference.pdf.

Yutong Xin, Jimmy Xin, Gabriel Poesia, Noah D. Goodman, Qiaochu Chen, and Işıl Dillig. Automated discovery of tactic libraries for interactive theorem proving. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA2):1974–2001, 2025b. doi: 10.1145/3763121. URL https://doi.org/10.1145/3763121.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=96zNuQrH9Y.

Yuanhe Zhang, Jason D. Lee, and Fanghui Liu. AI4SLT: Empirical processes in Lean 4 for formal statistical learning theory. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=dfqmQ9WhCP.

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

Jin Peng Zhou, Yuhuai Wu, Qiyang Li, and Roger Baker Grosse. REFACTOR: Learning to extract theorems from proofs. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fgKjiVrm6u.

A Proof-Term Size

We measure proof size by counting nodes in the abstract syntax trees of the kernel expressions, including their universe levels. Let $N(e)$ denote this count. Each expression constructor contributes one node plus the counts of its expression and universe-level children. Metadata wrappers contribute no node. Names, binder annotations, indices, and literal payload lengths contribute no additional nodes. A universe-level constructor contributes one node plus the counts of its level children.

Shared declarations. Fix the target declarations T and background B. Starting from the statements and proofs of T, follow declaration dependencies through their types and bodies, stopping at B. Include the declarations required by inductive families and recursor rules in this closure. Let H be the resulting non-background declarations other than the targets. A shared declaration belongs to H once, even if several targets use it. Target statements are fixed and do not contribute directly to the size, but their non-background dependencies do.

Size. For a declaration d, let $V(d)$ be the AST size of its value, or zero when it has no value, and let $R(d)$ sum the AST sizes of its recursor-rule right-hand sides, or zero for other declaration kinds. Define

$$L_B(P) = \sum_{t \in T} V(t) + \sum_{h \in H} \big[N(\mathrm{type}(h)) + V(h) + R(h)\big]. \tag{A.1}$$

A repeated subtree contributes at every occurrence. Explicit sharing through a local binding or an auxiliary declaration can reduce this total. The same counting rule, background, and target statements apply to the original and rewritten proofs.

Examples. An application $h\,x$, with no universe arguments on h, has three nodes: the application and its two children. A constant reference with no universe arguments has one node regardless of its name. A reference with universe arguments additionally pays for the ASTs of those levels. A helper contributes its statement and proof once, as well as one reference node at each use (plus any universe arguments).

Helper cost. We define a helper's cost as the sum of the node counts of its statement and proof, using the proof-size definition above.

B Selected Lemmas Extracted by Proof Compression

The four examples below show what proof-compression search can extract: a coefficient formula for a polynomial recurrence, a polynomial chain rule, a bound on transcendence degree, and a criterion for equality of scheme morphisms. The first three examples come from Permanent experiments. The fourth, equality of scheme morphisms by relative affine tests, is one of the 793 FLT helper instances analyzed in Figure 6. For each, we give the Lean declaration, a mathematical statement, and the proof idea. They illustrate the mathematical content of the helpers. They do not estimate how often such helpers are found. We make no claim that the underlying principles are new.

B.1 Degree and Coefficient of a Polynomial Deletion Recurrence

Lean statement.

```
theorem dc_weighted_deletion_polynomial.{u, v}
    {α : Type u} {R : Type v} [DecidableEq α] [CommRing R]
    (w : α → ℕ) (d : Finset α → ℕ)
    (hdzero : d ∅ = 0)
    (hdpos : ∀ s : Finset α, s.Nonempty → d s ≠ 0)
    (hdegree : ∀ (s : Finset α) (a : α), a ∈ s → d (s.erase a) + w a =
    d s)
    (p : Finset α → Polynomial R) (c : Finset α → R)
    (hzero : p ∅ = 1)
    (hstep : ∀ s : Finset α, s.Nonempty →
      p s = Polynomial.C (c s) -
```

```
        ∑ a : s, Polynomial.X ^ w a.1 * p (s.erase a.1))
    (s : Finset α) :
    (p s).natDegree ≤ d s ∧
      (p s).coeff (d s) =
        (-1 : R) ^ s.card * (s.card.factorial : R)
```

Mathematical statement. Let R be a commutative ring and A a set. Assign a nonnegative integer $w(a)$ to each $a \in A$ and a nonnegative integer $d(S)$ to each finite subset $S \subseteq A$. Suppose that $d(\varnothing) = 0$, that $d(S) > 0$ for nonempty S, and that

$$d(S \setminus \{a\}) + w(a) = d(S) \qquad (a \in S).$$

For each finite S, let $P_S \in R[X]$ and $c(S) \in R$. Assume

$$P_\varnothing = 1,$$
$$P_S = c(S) - \sum_{a \in S} X^{w(a)} P_{S \setminus \{a\}} \qquad (S \neq \varnothing).$$

Then P_S has no nonzero coefficient above degree $d(S)$, and

$$[X^{d(S)}]P_S = (-1)^{|S|}|S|!,$$

where $[X^j]P$ denotes the coefficient of X^j and the integer on the right is mapped into R. The hypotheses on d imply $d(S) = \sum_{a \in S} w(a)$ and $w(a) > 0$. The displayed coefficient may vanish in R. Exact degree $d(S)$ follows only when it is nonzero. Lean's natural degree assigns degree zero to the zero polynomial.

The coefficient at degree $d(S)$ is independent of the constants $c(T)$ in the recurrence, even though lower coefficients can depend on them. To see why, induct on $|S|$. For nonempty S, the constant $c(S)$ contributes nothing at the positive degree $d(S)$. Multiplication by $X^{w(a)}$ shifts degrees. Since $d(S \setminus \{a\}) + w(a) = d(S)$, the summand for a contributes the coefficient at degree $d(S \setminus \{a\})$ in the smaller polynomial. Writing $k = |S|$, induction gives $(-1)^{k-1}(k-1)!$ for each contribution. The recurrence subtracts these k terms, so the coefficient at degree $d(S)$ in P_S is

$$-k\,(-1)^{k-1}(k-1)! = (-1)^k k!.$$

The same induction bounds the degree of every summand by $d(S)$. The factorial counts the orders in which the elements can be deleted.

B.2 Expanding a Derivation after Polynomial Substitution

Lean statement.

```
theorem dc_derivation_map_expansion.{u, v, w, x}
    {R : Type u} {I : Type v} {A : Type w} {M : Type x}
    [CommSemiring R] [Fintype I] [CommSemiring A] [Algebra R A]
    [AddCommMonoid M] [Module R M] [Module A M] [IsScalarTower R A M]
    (φ : MvPolynomial I R →ₐ[R] A) (D : Derivation R A M)
    (p : MvPolynomial I R) :
    D (φ p) = ∑ i : I, φ (MvPolynomial.pderiv i p) • D (φ
    (MvPolynomial.X i))
```

Mathematical statement. Let R and A be commutative semirings, with A an R-algebra, and let I be a finite set of variables. Let M be an additive commutative monoid with R- and A-module structures satisfying $(ra)m = r(am)$ for $r \in R$, $a \in A$, and $m \in M$, where the R-action on A is its algebra structure. Let $D : A \to M$ be an R-linear derivation, meaning that $D(1) = 0$ and $D(ab) = aD(b) + bD(a)$. For every R-algebra homomorphism $\varphi : R[X_i \mid i \in I] \to A$ and every polynomial p,

$$D(\varphi(p)) = \sum_{i \in I} \varphi\left(\frac{\partial p}{\partial X_i}\right) \cdot D(\varphi(X_i)).$$

The dot denotes the A-action on M, and the derivatives are formal polynomial derivatives. No field or characteristic-zero hypothesis is imposed.

This is a polynomial chain rule: once the values $D(\varphi(X_i))$ are known, the formula determines D on every substituted polynomial. The familiar one-variable version is

$$D(P(a)) = P'(a) \cdot D(a),$$

formalized in Mathlib's `Derivation.map_aeval`. The extracted lemma allows several variables, with φ specifying their substituted values. To prove the formula, regard both sides as derivations on the polynomial algebra. They agree on each variable and on the base coefficients, hence on every polynomial.

B.3 Bounding Transcendence Degree by a Finite Generating Set

Lean statement.

```
theorem dc_trdeg_le_finset_adjoin.{u, v}
    {F : Type u} {E : Type v} [Field F] [Field E] [Algebra F E]
    (K : IntermediateField F E) (s : Finset E)
    (hK : K ≤ IntermediateField.adjoin F (s : Set E)) :
    Algebra.trdeg F K ≤ (s.card : Cardinal)
```

Mathematical statement. Let E/F be a field extension, let $S \subseteq E$ be finite, and let K be an intermediate field with $F \subseteq K \subseteq F(S)$. Here $F(S)$ is the smallest subfield of E containing F and S. Then

$$\operatorname{trdeg}_F K \leq |S|.$$

The transcendence degree is the cardinality of a transcendence basis, or equivalently the maximal number of elements algebraically independent over F. The comparison is a cardinal inequality. In particular, it asserts that this transcendence degree is finite.

To bound the transcendence degree of K, it suffices to find generators for a field containing it. The bound follows from

$$\operatorname{trdeg}_F K \leq \operatorname{trdeg}_F F(S) \leq |S|.$$

The first inequality follows from the inclusion $K \subseteq F(S)$. The second bounds transcendence degree by the number of field generators. The Lean proof combines monotonicity under field embeddings with Mathlib's `Algebra.IsAlgebraic.trdeg_le_cardinalMk` after expressing the generated field in the required form. There is no need to construct a transcendence basis of K.

B.4 Equality of Scheme Morphisms by Relative Affine Tests

Lean statement.

```
theorem dc_relative_affine_hom_ext.{u}
    {R : Type u} [CommRing R]
    {X Y : AlgebraicGeometry.Scheme.{u}}
    (s : Quiver.Hom X
      (AlgebraicGeometry.Spec (CommRingCat.of R)))
    (f g : Quiver.Hom X Y)
    (h : ∀ (C : Type u) [CommRing C] [Algebra R C]
      (z : Quiver.Hom
        (AlgebraicGeometry.Spec (CommRingCat.of C)) X),
      CategoryTheory.CategoryStruct.comp z s =
        AlgebraicGeometry.Spec.map
          (CommRingCat.ofHom (algebraMap R C)) →
      CategoryTheory.CategoryStruct.comp z f =
        CategoryTheory.CategoryStruct.comp z g) : f = g
```

Mathematical statement. Let R be a commutative ring, let X, Y be schemes, and fix a scheme morphism $s : X \to \operatorname{Spec} R$. Let $f, g : X \to Y$ be scheme morphisms. Suppose that for every

commutative R-algebra C, with structure homomorphism $\eta_C : R \to C$, and every scheme morphism $z : \operatorname{Spec} C \to X$ satisfying

$$s \circ z = \operatorname{Spec}(\eta_C),$$

one has

$$f \circ z = g \circ z.$$

Then $f = g$. Here $\operatorname{Spec} C$ is the affine scheme associated with C, and $\operatorname{Spec}(\eta_C) : \operatorname{Spec} C \to \operatorname{Spec} R$ is the morphism induced contravariantly by η_C. The tests range over all such R-algebras and compatible maps, not merely field-valued points. No affineness assumption on X or Y, or structure morphism from Y to $\operatorname{Spec} R$, is required. As in the Lean statement, fix a universe containing R, X, and Y, and take the test rings C in that universe.

Recall that every scheme has an affine open cover, and two morphisms out of a scheme are equal if their restrictions to every member of such a cover agree. It therefore suffices to check that the inclusion maps of an affine open cover qualify as tests over $\operatorname{Spec} R$.

Take an affine open piece $\operatorname{Spec} C$ of X with inclusion map $z : \operatorname{Spec} C \to X$. The composite

$$s \circ z : \operatorname{Spec} C \longrightarrow \operatorname{Spec} R$$

corresponds to a ring homomorphism $R \to C$, which equips C with an R-algebra structure. For this structure, z satisfies the required compatibility with s, so the hypothesis gives $f \circ z = g \circ z$. Since f and g agree on every piece of an affine open cover, they are equal. The Lean proof uses `Spec.preimage` and `Spec.map_preimage` for the ring homomorphism and then `openCover.hom_ext` for equality on the cover.

A proof can therefore apply the lemma by checking all affine tests over R, without choosing an open cover.

C Agent-System Details

Each round selects related target proofs, proposes a helper, proves it in Lean, and attempts shorter proofs of the targets using that helper. The local acceptance rule in Equation 4.1 charges the helper's statement and proof against the savings on the selected targets. We use the AST node-count rule in Appendix A; Table 1 distinguishes the accounting scopes.

Each target is identified by its module and declaration name. The accounting retains the largest positive saving for each target and then charges the installed helper and its candidate-owned auxiliary declarations once across those retained rewrites. In multi-file searches, credit is restricted to targets in compatible contexts, as shown in Figure 7. These local credits do not measure the saving from installing all accepted helpers together. That quantity requires a joint proof-term evaluation of the combined proofs against the fixed background, as in Appendix E.1.

Prompt templates derived from a Permanent checkpoint search are given in Appendix C.1.

C.1 Discovery Prompt Templates

The templates below show the instructions for proposing a helper, shortening its proof, and rewriting a target to use it. Braced fields such as `{target_statements_and_proofs}` are filled with the selected Lean declarations, legal background, measured costs, and earlier feedback. The excerpts come from one Permanent checkpoint search. Fixed instructions are retained. The rewrite excerpt omits file-operation details.

Configuration. This search is one of the Permanent searches in Appendix E.8. It used GPT-6 Astra with low reasoning effort for all three roles and a total cap of 20 model calls. The normal allocation was one helper proposal, three helper-proof optimization calls, and one rewrite call for each of 16 targets.

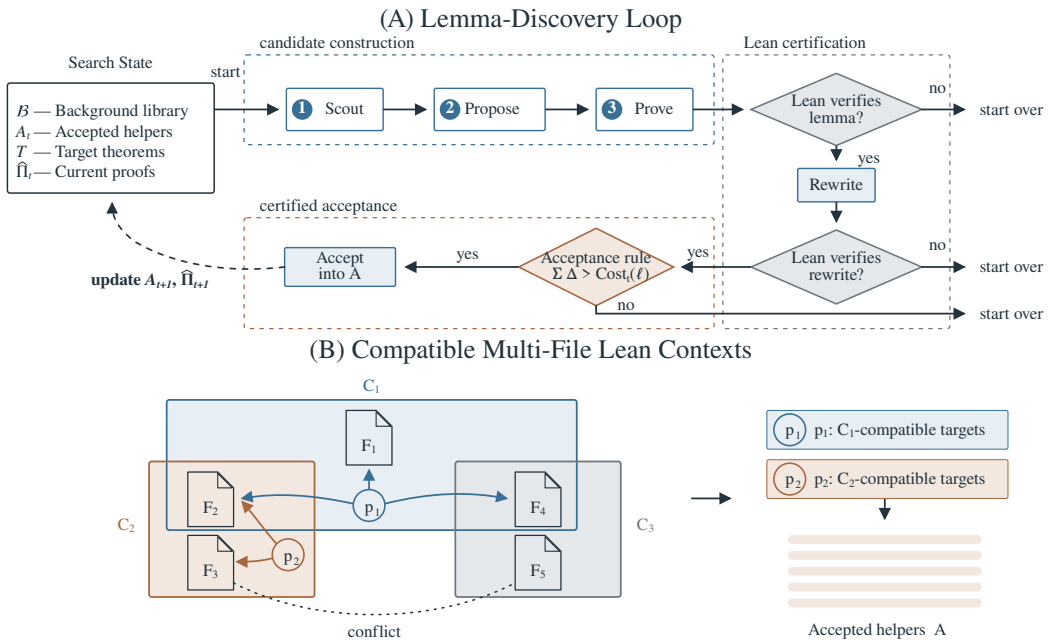


Figure 7: **Lemma-discovery workflow.** (A) In each round, the scout selects related target theorems, the proposer suggests a candidate helper, the prover proves it in Lean, and the rewriter attempts helper-using rewrites. The helper is accepted only if its savings exceed its cost (Equation 4.1). (B) In multi-file runs, candidates are evaluated inside compatible Lean contexts: a context is the set of files that can be compiled together with the candidate without Lean-level conflict. A file may appear in multiple contexts (e.g. $F_2 \in C_1 \cap C_2$ in panel B), but each candidate is credited only on theorem targets in files compatible with the context where it was verified.

Proposal.

```
FLT exploration declaration policy: propose and prove only lemma or
↪  theorem declarations. Do not introduce def, abbrev, instance, opaque,
↪  axiom, structure, inductive, or class declarations. Unsupported
↪  declarations are rejected before consumer rewrites. Keep all original
↪  target statements and legal imports intact.

Propose ONE reusable Lean 4 lemma or theorem that compresses the actual
↪  proofs below.
The frozen exploration group contains exactly {target_count} distinct
↪  rewrite targets. Every target below is eligible for an independent
↪  consumer rewrite after the helper is proved and frozen. Your TARGETS
↪  comment describes expected applicability; it does not shrink the
↪  eligible group. You need not help every target. Identify proof
↪  patterns actually present in these declarations and proofs.

SCORING: node-primary compiled expression nodes (ConstantInfo.type +
↪  ConstantInfo.value). Charge a shared helper once; choose at most one
↪  final rewrite per target. The helper must be actually used in the
↪  target proof. Decorative uses, unchanged proofs and an unverified
↪  novelty claim are not compression.

OUTPUT CONTRACT:
- Emit exactly one complete `lemma dc_*` or `theorem dc_*` declaration and
↪  proof, in a Lean code block, with no sorry.
- Do not generate def, abbrev, instance, axiom, import, namespace,
↪  section, variable, open or attribute declarations.
- Precede it with `-- TARGETS: name1, name2, ...` naming at least 2
↪  applicable names from this exact group and explain the repeated proof
↪  pattern.
```

```
- Use only legal imports and definitions at the helper installation point. Do not use target theorems to prove the helper, import their owner modules, or assume later source declarations are available at an earlier insertion point. Background source is reference context, not a claim that every declaration is available in every namespace or at every insertion point.
- Reject simple wrappers of existing lemmas, commutativity wrappers and definitional aliases. Existing reconstruction through simp?, exact? or rw? is trivial. An unresolved detector result does not establish non-triviality.
- Rewrite consumers must use the helper in a proof-relevant way such as rw/apply/exact; `have _ := dc_*` alone is decorative.
- If there is no useful shared abstraction, return GIVE_UP: followed by the concrete reason.

HELPER INSTALLATION MODULE: {helper_module}
Imports: {legal_imports}
Write all type and typeclass binders explicitly; source variable blocks are not inherited.
Check that exact helper file with the bound native options:
cd {project_root} && lake env lean {helper_file}
A plain lake env lean invocation does not apply these frozen options. Keep the options unchanged. Bind universe parameters explicitly in the declaration name, for example theorem dc_name.{u, v}; source universe declarations are not inherited.

PREVIOUS RESULTS AND ENGLISH TRIVIAL FEEDBACK:
{previous_feedback}

LEGAL SOURCE BACKGROUND (imports, definitions, namespaces and options; theorem declarations omitted):
```lean
{legal_background}
```

FROZEN THEOREM CONTEXT: COMPLETE DECLARATIONS AND PROOFS, WITHOUT TRUNCATION
{target_statements_and_proofs}

COMPLETE CHECKPOINT PROOF SOURCE
This is the complete source of the checkpoint used by the consumer rewrite environment.
The {root_count} final theorem roots provide the purpose of the proof. Only the explicitly listed
consumer targets are rewrite targets. Source-support declarations may remain in this file;
their presence does not make them eligible targets. The helper must compile using only
the stated helper-host imports ({legal_imports}); it cannot import this consumer owner.
```lean
{checkpoint_source}
```
```

Proof.

```
FLT exploration declaration policy: propose and prove only lemma or theorem declarations. Do not introduce def, abbrev, instance, opaque, axiom, structure, inductive, or class declarations. Unsupported declarations are rejected before consumer rewrites. Keep all original target statements and legal imports intact.

You are part of a system that discovers hidden mathematical structures in formal proofs across the full Mathlib library.
```

````
A hidden structure exists whenever many proofs share an unnamed pattern
↪  -- a repeated tactic sequence, a repeated expression, a typeclass
↪  instance that would unlock a family of simp lemmas, an abbreviation
↪  that would collapse boilerplate. Our job is to NAME it.

Structures can take the form of DEFINITIONS (def, abbrev, instance,
↪  structure) or LEMMAS (lemma, theorem). Both compress proofs; both are
↪  first-class targets. Prefer whichever abstraction most faithfully
↪  captures the pattern you see.

We measure compression by recursively counting occurrences in the
↪  compiled Lean expressions for ConstantInfo.type + ConstantInfo.value.
↪  expr_nodes_with_levels_v1 is the authoritative score. pp_raw_chars_v1
↪  is retained only as a secondary diagnostic and never changes
↪  acceptance. A structure's OWN cost is its full compiled declaration:
↪  the STATEMENT (its type/signature) PLUS its proof term. Every
↪  expression occurrence counts, so a lean statement (fewer/implicit
↪  arguments, the least-general type that still applies) is as valuable
↪  as a short proof. A structure is valuable when this full node cost is
↪  amortized by node savings across many theorems. The more theorems a
↪  structure helps, and the cheaper its own declaration, the better. A
↪  small structure that saves many times its own cost across multiple
↪  proofs is excellent. A larger structure must save proportionally more
↪  to pay for itself.

You proposed the following lemma, and we measured its kernel proof term
↪  cost:

LEMMA STATEMENT:
```lean
{helper_statement_and_proof}
```

MEASURED COMPILED EXPRESSION COST: {helper_cost} nodes (ConstantInfo.type
↪  + ConstantInfo.value)

This is the cost every theorem that uses this lemma must amortize. A
↪  cheaper proof
of the same statement would make the lemma more valuable.

Can you prove this exact same statement with a shorter proof? The
↪  statement must
remain identical -- only the proof body should change.

Return the complete lemma (statement + new proof) in ```lean ... ```
↪  markers.
Edit and test the same prepared file {helper_file}. Preserve its header
↪  and helper statement. Its contents, not the returned code block,
↪  determine the proof being measured. Return the same fully qualified
↪  helper name.

FLT role skill (authoritative node-primary runtime):

---
name: flt-prover
description: Prove or minimize FLT helpers using definitive MCP
↪  verification and node-primary accounting.
---
Use lean-lsp tools for interactive proof-state checks. Use
↪  proof-term-size's
verify_proof for the final submitted source, with the exact theorem_name
↪  and the
complete legal Lean source. Return that same source unchanged after
↪  verification;
a changed source needs another receipt. Preserve the caller's artifact
↪  format.
````

```
expr_nodes_with_levels_v1 is authoritative; pp_raw_chars_v1 is diagnostic only.
Charge helper type plus value and every added declaration. Missing or negative
metrics are measurement failures, never zero costs. Compilation and absence of
sorry are required. When proving a consumer using a helper, supply lemma_name and
require both zeta and full-reduction usage evidence; a syntactic occurrence alone
is insufficient. A helper's own proof does not need to use itself.

The controller performs final source-bound validation. Tool success does not
constitute Benchmark FINAL. Never edit a helper already frozen for consumers;
submit any optimization as a separate candidate for controller verification.
```

Rewriting.

```
DISCOVERY AUTHORITY BOUNDARY
======================================================================

This is the fast INNER rewrite loop. Copy all of ``context.lean`` into each
attempt and edit only the proof of ``{helper_name}``'s consumer theorem.
Preserve every import, attribute, declaration modifier, theorem statement,
namespace, option, and suffix declaration.

``verify_attempt`` is the only official receipt tool for this inner loop. Its positive
result means ``FAST_CANDIDATE_READY``: the source compiles in the fast harness,
uses ``{helper_name}``, and appears smaller. It is not yet
``PROVISIONAL_DISCOVERY`` and it is not Benchmark FINAL.

On the first such result, STOP IMMEDIATELY. Do not inspect sidecar files, run
shell or Lean-LSP probes, or create another attempt. After the agent exits, the
trusted controller exact-owner compiles that saved source once and binds the
receipt to the source, environment, Core Helper, and B/C/H/G accounting. The
selector reuses that receipt without recompiling. Only the controller may emit
``PROVISIONAL_DISCOVERY``. Benchmark FINAL is a later independent clean replay.

The detailed historical rewrite protocol follows. Within it, a ``PASS`` from
``verify_attempt`` means ``FAST_CANDIDATE_READY`` under the boundary above.


[File-operation instructions omitted.]

TASK: Rewrite this proof to use `{helper_name}` and produce a smaller node score than the original.

Acceptance criteria (checked by ``mcp__proof-term-size__verify_attempt``):
```

```
  • Lean compiler -- the proof must compile cleanly (no errors, no
    ``sorry``).
  • Axioms -- the proof's axiom set must NOT include ``sorryAx``.
    ``verify_attempt`` returns ``status = "HAS_SORRY"`` if it does.
  • ZETA_USAGE_PROBE -- the lemma name must appear in the proof term
    after ``zetaReduce`` (i.e. structurally used, not just textually).
    ``verify_attempt`` returns ``status = "LEMMA_DECORATIVE"`` otherwise.
  • Node score -- expr_nodes_with_levels_v1 is authoritative; a
  ↪  same-size-or-larger rewrite is REJECTED.

The authoritative baseline is {baseline_nodes} expr_nodes_with_levels_v1
↪  nodes. You MUST pass both `score_original = {baseline_nodes}` and
↪  `kernel_original = {baseline_raw_chars}` to `verify_attempt`. The
↪  attempt passes the size gate only when `score_saved > 0`.
↪  pp_raw_chars_v1 is diagnostic only and must not change the
↪  node-primary verdict.

AVAILABLE dc_* LEMMA (Lean source):

----- BEGIN LEMMA -----
{helper_statement_and_proof}
----- END LEMMA -----

THEOREM TO REWRITE (Lean source):

----- BEGIN THEOREM -----
{target_statement_and_proof}
----- END THEOREM -----

Remember: write your attempts to ``attempts/attempt_NNN.lean``. The
harness picks the best verified attempt -- you do not return the proof
in your final message.

FLT role skill (authoritative node-primary runtime):

---
name: flt-rewrite
description: Rewrite FLT consumer proofs with source-bound MCP
↪  verification, reduction usage checks and node costs.
---
Preserve the exact target statement, owner context and frozen helper
↪  contents. Use
lean-lsp for intermediate proof states. Use proof-term-size's
↪  verify_proof on the
final full source with exact theorem_name, lemma_name and the supplied
↪  baseline.
Require compilation, no sorry, zeta usage and full-reduction usage. A
↪  helper that
is mentioned but erased by reduction is decorative. Return the exact
↪  checked
source and preserve the requested output format. Editing it invalidates
↪  its receipt.

expr_nodes_with_levels_v1 is authoritative; pp_raw_chars_v1 is diagnostic
↪  only.
B and C include target values and their target-bound auxiliaries; H
↪  includes helper
type and value once; G includes all other candidate-owned declarations
↪  once.
Missing or negative measurements are errors. Do not sum competing
↪  rewrites or
charge a shared helper repeatedly. The controller computes B-C-H-G and
↪  verifies
the exact owner. A fast positive stops search as FAST_CANDIDATE_READY;
↪  independent
```

```
Benchmark FINAL remains separate. Respect private output and import
↪  boundaries.
```

C.2 Discovery Procedure and Accounting Details

Selection. At round t, a retrieval agent (the scout) selects a local set of target theorems T_t and related library statements. This local set defines the scope of the round. The benchmark fixes the full target collection T. The proposer looks for common reasoning across the selected proofs and states a helper lemma that could shorten several of them. In cumulative searches, the context includes helpers retained from earlier rounds. The choice of material matters: in a calculus comparison, similarity-based selection yields the most successful rewrites, while LLM-based selection yields greater total savings. Figure 7 shows the workflow. Appendix Table 2 reports a comparison of different retrieval methods. It motivates selecting related arguments together when searching for reusable helpers.

Rewriting. A prover constructs a Lean proof $\pi_t(\ell)$ of the proposed lemma ℓ. For each selected target, a rewriter attempts a shorter proof using the verified helper lemma.

Cost. The search uses the shared-cost principle of Section 2.2: a helper must save more than it costs. Let $s(\cdot)$ denote the AST node count defined in Appendix A. The lemma's statement and constructed proof have cost

$$\mathrm{Cost}_t(\ell) := s(\ell) + s(\pi_t(\ell)). \tag{C.1}$$

For each $\varphi \in T_t$, let $\Delta_t(\ell, \varphi) \geq 0$ be the saving of a successful lemma-using rewrite relative to its lemma-free counterpart. The local selection rule retains the lemma when

$$\boxed{\sum_{\varphi \in T_t} \Delta_t(\ell, \varphi) > \mathrm{Cost}_t(\ell).} \tag{C.2}$$

The experiments share the principle of charging helper costs. Their size measures are distinguished in Table 1.

The model-allocation experiment in Appendix D.1 examines which roles benefit most from a stronger model.

D Additional Experiments

D.1 Discovery Ablations

We measure proof size with a common AST node-count rule. Table 1 distinguishes the scope of each calculation.

Experiment	Size measure	Accounting scope
Benchmark and Permanent trajectory	AST nodes	One proof term for all task targets, with required declarations counted once.
Workflow ablation	AST nodes	Local net savings, summed over independent runs after helper charges.
Model allocation	Counts	Accepted slots and successful rewrites.
FLT helper cost analysis	AST nodes	Cost of each helper, not a joint net saving.

Table 1: Accounting scopes. All reported proof sizes and helper costs use the AST node-count rule in Appendix A.

Workflow ablation. We compare proposer-led search with three methods that preselect 80 theorem examples. Two retrieve the top 400 examples by embedding similarity and then choose 80 by similarity ranking or an LLM. The third uses an LLM to select 80 from the full eligible pool. On Mathlib calculus, similarity-only selection yields the most successful rewrites, while the two LLM selection methods yield greater net node savings (Table 2). This ordering describes the observed runs: their rewrite targets and realized compute differ, so it does not isolate the effect of selection alone.
Additional examples counts distinct eligible proofs shown to the proposer beyond its preselected set, averaged over six runs. It includes initial source context and later reads, but excludes proofs shown only to the selector. The counts are lower bounds. In the three preselection methods, the initial 80 examples are additional to this column. Ratios between its entries are therefore not ratios of total exposure.

Initial material selection	Additional examples	Successful rewrites	Net nodes saved
Proposer searches itself	≥ 138.8	74/288 (25.7%)	996,025
Embedding top 400 → similarity top 80	≥ 20.7	**88/288 (30.6%)**	1,163,542
Embedding top 400 → LLM selects 80	≥ 23.8	53/288 (18.4%)	1,549,417
Full eligible pool → LLM selects 80	≥ 44.7	66/288 (22.9%)	**1,666,540**

Table 2: GPT-5.5/xhigh, with six runs per method. Each run allows up to 1,800 s for helper discovery and 48 rewrite targets (900 s per target, eight attempts). The denominator counts planned opportunities, with zero successful rewrites for no-helper runs. Targets and realized compute differ across runs. The eligible pool contains 3,643 theorems. Proposer-led search has no preselected set. Additional-example counts omit partial proofs and source outside this pool. Savings are summed over independent runs after charging helper costs. They are measured in expression nodes (Table 1), not as the saving of a jointly installed collection.

Model allocation. We test model allocation on one calculus cover, with 27 slots per arm. Each arm assigns a weaker model to one role while retaining the stronger models elsewhere: the scout (retrieval and re-retrieval), the proposer, or the prover and rewriter together. Table 3 reports accepted slots and successful rewrites. Weakening the scout gives 18 accepted slots and 126 successful rewrites, compared with 18 and 115 for the control; weakening the proposer gives 12 and 63, and weakening the prover and rewriter gives 8 and 16. The model pairs differ across roles, and the prover and rewriter are changed together, so this comparison does not isolate the importance of each role.

Role given the weak model	Accepted slots	Successful rewrites
None (control)	18/27	115
Scout and re-scout	18/27	126
Proposer	12/27	63
Prover and rewriter	8/27	16

Table 3: Model-allocation outcomes. The strong scout and proposer use GPT-5.4, the strong prover and rewriter use GPT-5.5, and the weak model is GPT-5.4-mini. Prover and rewriter share one model role and are weakened together. Each arm has 27 slots on one calculus cover.

E Benchmark Details

E.1 Task Specification and Evaluation Procedure

The organizer fixes the task before the agent starts. The task specification lists the exact target declaration names, types, and modules. It also records the fixed background, protected declarations, editable proof-internal declarations, Lean and library versions, and cost policy. A separate run configuration records the model, harness, available tools, proof visibility, and maximum search time. The task thus specifies the mathematical problem, and the run configuration specifies the resources available to solve it.

Submission. The agent works in a prepared checkout and invokes `./benchmark-submit` to compile and package its candidate modules. The submission manifest records their module names,

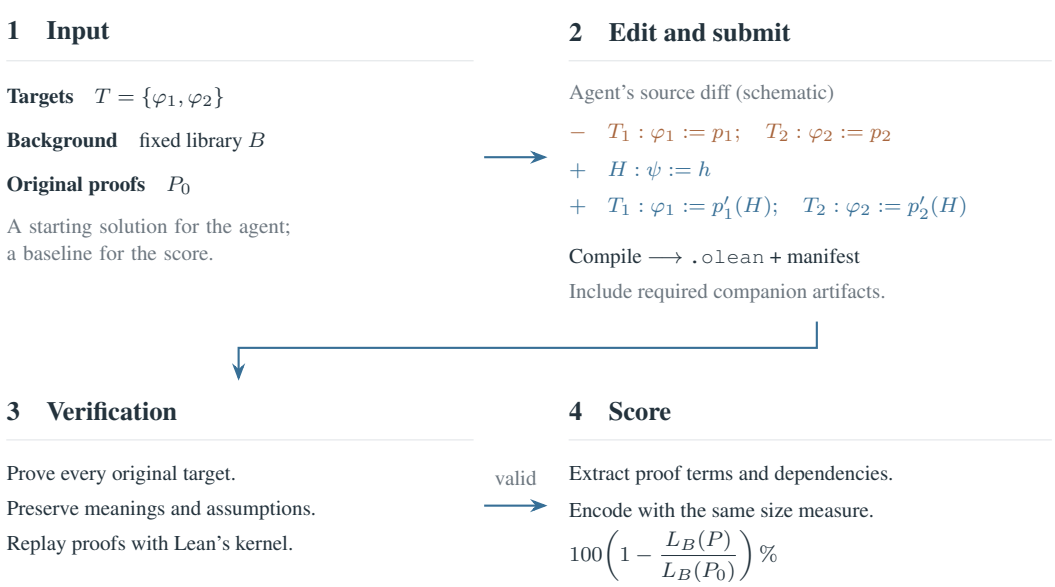


Figure 8: A proof-compression task. The agent receives targets, a fixed background, and original proofs, then edits and compiles the Lean source. The evaluator checks the submitted proofs against the original targets and measures their required proof material. The source diff is schematic. H illustrates an intermediate lemma introduced during rewriting.

dependency order, and corresponding compiled artifacts: the `.olean` files and any companion files required by the pinned Lean version. The operator also invokes submission when the search budget ends to capture the final workspace. The evaluator reads the compiled declarations as data, and the source is retained for reproduction.

Verification. For each target, the evaluator resolves its exact declaration name and module, then checks its proof against the original type specified by the task. It checks that declarations determining the target's meaning remain fixed and replays the candidate declarations in dependency order against the prescribed background. Every target must be covered. Unresolved proofs, additional axioms, and references that cannot be resolved within the task environment cause rejection.

Verification records validity and coverage. Under the task scoring rule, invalid or unfinished attempts receive zero. The endpoint averages reported below are computed under this rule.

Measurement. Starting from the target statements and proofs, the evaluator collects required non-background declarations, stopping at the fixed background. We count target proof terms and the types and bodies of auxiliary declarations under the AST node-count rule in Appendix A. Fixed target statements do not contribute directly to the size, but their dependencies do. Shared declarations contribute once; references and repeated subtrees contribute at each occurrence. Baseline and candidate use the same counting rule and background.

This counting rule measures proof size, not the time needed to check it or the difficulty of its mathematics. Explicit sharing through a binding or helper can reduce proof size. The case study and translation experiment examine changes in argument and exposition separately.

E.2 Benchmark Tasks and Protected Results

Each task fixes a collection of Lean theorem statements to preserve. A task can protect final conclusions, intermediate results, or a collection of library theorems. Table 4 gives the mathematical content and the scope of each target collection.

Task	Targets	Mathematical content and protected collection
Permanent	2	Lower bounds for arithmetic formulas computing the permanent: the division-free and rational-formula conclusions, in `Permanent.lean`.
Compactness	5	The two results on the compactness and degeneracy conjectures formalized in OpenAI's released codebase (OpenAI, 2026b). The benchmark preserves five Lean declarations expressing these results: the quantitative and asymptotic compactness counterexamples, the negation of Erdős problem 180, the two-degenerate extremal counterexample, and the negation of Erdős problem 146.
Ramsey	96	Multicolour triangle Ramsey numbers: a fixed collection of statements in `MulticolorTriangleRamsey.lean`, including intermediate combinatorial constructions and the final logarithmic growth bounds. All 96 statements are targets, not only the final bound.
SLT26	26	Statistical learning theory and supporting probability and matrix results across 15 files: Dudley bounds, Hanson–Wright concentration, spectral perturbation, random-matrix bounds, Gaussian concentration, and least-squares error bounds.
SLT49	49	A separate, broader SLT collection across 29 files, additionally including covering numbers, Efron–Stein, entropy, Gaussian Poincaré and log-Sobolev inequalities, and supporting analytic results.
Mathlib197	197	Theorem targets in 10 import-leaf modules drawn from algebra, calculus, category theory, manifolds, and measure theory.
Mathlib372	372	A larger collection in 20 import-leaf modules from the same five areas. Each selected theorem statement is protected separately.

Table 4: Mathematical scope of the seven benchmark tasks.

E.3 Instances and Run Conditions

Reasoning effort. The runs use high reasoning effort for GPT-6 Astra, Opus 5.5, Fable 5.1, and DeepSeek 4.1, medium for Muse, and low for GLM Flash 5.3.

These are requested settings, which need not correspond to equal computation across models. Search-time budgets are a separate experimental variable, and the comparisons do not establish that each model's effort setting is optimal. The Muse series combines runs from two execution configurations.

Table 5 gives the target counts and baseline proof sizes. SLT's two target sets and Mathlib's two leaf-target sets are separate tasks.

Task	Targets	Original proof size
Ramsey	96	11,338,478
SLT26	26	392,854,783
SLT49	49	406,689,591
Permanent	2	85,690,155
Compactness	5	1,307,221,424
Mathlib197	197	114,567,106
Mathlib372	372	186,421,329

Table 5: Target counts and original proof-proof sizes.

The reported runs make the original proofs available and use maximum search budgets of 60 or 120 minutes. Each budget starts a separate run, and an agent may finish early. Model names identify model–harness configurations.

Aggregation. The figures and tables measure proof size (Table 1). For each task, model, and budget, the bar is the mean over the assigned searches, four at 60 minutes and two at 120 minutes. Searches without a valid endpoint count as zero, and each search appears as a dot. A Fable 5.1 run resumed after an interruption counts once.

Missing, rejected, and unfinished results enter these averages as zeros.

E.4 Benchmark Measurements

Task	DeepSeek 4.1	GLM Flash 5.3	Muse	Fable 5.1	GPT-6 Astra	Opus 5.5
Ramsey	11.55 (4)	> -0.01 (4)	10.32 (4)	35.94 (4)	95.00 (4)	50.63 (4)
SLT26	0.05 (4)	0.00 (4)	< 0.01 (4)	18.25 (4)	90.64 (4)	31.59 (4)
SLT49	18.64 (4)	0.00 (4)	2.55 (4)	20.69 (4)	68.04 (4)	24.22 (4)
Permanent	11.06 (4)	0.01 (4)	3.02 (4)	45.70 (4)	85.15 (4)	14.49 (4)
Compactness	53.99 (4)	< 0.01 (4)	9.12 (4)	43.55 (4)	97.16 (4)	52.81 (4)
Mathlib197	1.11 (4)	< 0.01 (4)	0.33 (4)	2.60 (4)	57.43 (4)	5.15 (4)
Mathlib372	0.18 (4)	> -0.01 (4)	0.03 (4)	23.95 (4)	80.67 (4)	41.71 (4)

Table 6: Mean proof-size reduction (%) over the four searches at 60 minutes, counting searches without a valid endpoint as zero. Parentheses give the number of searches.

Conditional means use the same inclusion rules as Figure 3. —: no eligible endpoint. < 0.01: positive below displayed precision.

E.5 Context Scaling and Target Reuse

Target sets and repetitions. This experiment uses a fixed ordering of targets from the Frey curves and level-lowering development. The set for each $K \in \{2, 4, 8, 16, 32\}$ is the first K targets in that ordering, supplied as both proof context and eligible rewrite targets. The six fresh searches at each K reuse exactly the same target list with different random seeds. They do not sample six different theorem sets. The plotted variation therefore describes repeated searches on these fixed target sets.

Each point counts distinct targets with a successful rewrite credited to helpers from one trial. Repeated attempts on the same target count once. The means include all six trials at each K. Terminal searches that produce no accepted helper contribute zero targets helped.

K	Trials	Mean targets / trial	Mean helped / K
2	6	1.17	58.3%
4	6	2.33	58.3%
8	6	1.33	16.7%
16	6	3.00	18.8%
32	6	14.67	45.8%

At $K = 8$ and $K = 16$, one search produced no accepted helper. Both no-helper trials remain in the figure and averages as zeros.

Larger K supplies more context and more rewrite opportunities, with a different set of targets. The number helped per trial increases overall, but the helped fraction is non-monotone. These results measure delivered target utility under the tested search conditions. They do not isolate an effect of context on helper generality. A fixed target pool or held-out targets would be needed to test transfer.

E.6 Helper Size and Mathematical Significance

The context experiment measures how many targets use a helper. A separate FLT cohort asks what kinds of helpers the search finds, by comparing their own statement-and-proof costs with model-rated mathematical significance. Figure 6 groups helpers by rating and shows their costs in expression nodes.

The routine-or-below and modest bands have median costs of 9,322 and 10,917 nodes. The substantive band has a median of 21,243 nodes; the strongest band has a median of 110,802 nodes and contains ten instances. Median helper cost rises across these rating bands, although the highest band is small. This association concerns the amount of formal material needed to state and prove a helper. Figure 5b separately measures reuse. Appendix E.7 adds the rating distribution, the cost–reuse relationship, and reuse by rating band for this cohort.

A costly lemma can still shorten a collection if enough proofs benefit from it. Together, these observations motivate searching across related targets: a larger collection may support shared arguments whose construction would not pay off for a single theorem.

E.7 Additional FLT Helper Analysis

Figure 9 supplements the main-text cost comparison in Figure 6. We analyze 793 accepted FLT helper instances, each with a mathematical-significance rating, a statement-and-proof node count, and recorded reuse. Helper costs and reuse counts come from the original compression results. We include all rated instances without filtering on rating or cost. The 25-helper context-rating study and the fixed-target Frey reuse experiment are separate experiments and are not added to this cohort. The unit is an accepted instance, with repeated mathematical ideas retained. These counts neither enumerate distinct theorems nor measure the joint saving from installing all helpers together.

The median significance rating is 3, and 139 of the 793 rated instances (17.5%) score at least 5. Panels (b) and (c) analyze reuse across all 793 helper instances. We count recorded successful rewrites marked as requiring the helper. Helpers with the same number of successful uses span a wide range of costs. Median reuse is 2, 2, 2, and 2.5 in the rating bands 0–2, 3–4, 5–6, and 7–10, respectively; all ten highest-rated instances have reuse records. Higher ratings thus do not coincide with a large increase in typical reuse. These success counts are neither distinct-target counts nor joint net savings. Assessing a helper's compression benefit also requires the downstream saving after its own cost is charged.

E.8 Permanent Helper Significance

The pooled review contains 170 generated helpers from 13 Permanent proof versions: 68 from an 84-search batch and 102 from a 144-search batch. All scored helpers are retained, including

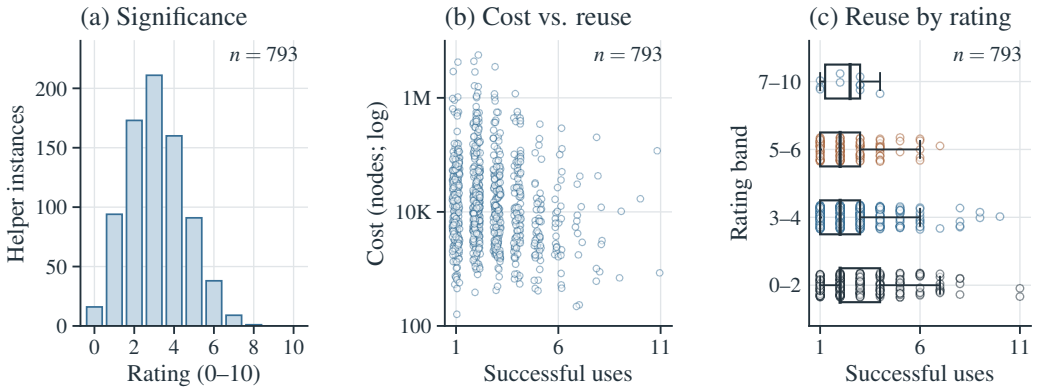


Figure 9: Additional analyses of the same 793-instance FLT cohort. (a) Significance ratings for all 793. (b) Node cost versus proof-relevant successes for all 793 instances. (c) Reuse by rating band for those same 793 instances, with band counts 283, 371, 129, and 10 from bottom to top. Cost is the node count of the helper's statement and proof, on a logarithmic axis. Each dot is an instance; offsets separate overlapping points. Box plots follow Figure 6.

equivalent or repeated discoveries. Searches that produced no scored helper do not enter the figure's denominator. Target neighborhoods can differ across checkpoints.

The review presents helpers in random order under opaque identifiers, with each helper's complete Lean statement and proof. It withholds generation batches, checkpoint identities, previous scores, and downstream rewrite outcomes. The GPT Pro review scores significance and insight separately on a 0–10 scale, and Figure 5a uses significance.

Significance asks how worthwhile the mathematical observation is and whether it merits explicit attention in an exposition, given the supplied background. The shared scoring anchors are:

- 0: No substantive content beyond restatement or definition unfolding.
- 1–2: Routine manipulation or bookkeeping with little significance or explanatory insight.
- 3–4: A modest, identifiable mathematical observation or explanation.
- 5–6: A substantive mathematical observation or non-obvious explanatory mechanism.
- 7–8: A strong conceptual contribution, connection, reduction, or organizing argument.
- 9–10: Exceptional mathematical significance or insight, justified concretely within the supplied context.

The instructions request an explanation and both scores for each helper, without forcing a score distribution or differences between similar lemmas. They exclude reuse frequency, proof savings, and tactic complexity as evidence of significance. We chose the cutoff of 3 after the scores were returned. This was an exploratory choice, based on the rubric's distinction between routine manipulation and an identifiable mathematical observation.

Table 7 gives the denominator and counts at cutoffs 3, 4, and 5 for every checkpoint. At the last two checkpoints, all 8 and all 19 rated helpers score at least 3, respectively. At cutoff 4, these counts fall to 2 and 9, and at cutoff 5, to 1 and 3. The late increase is therefore concentrated at score 3. It does not show the same rise in helpers assigned higher significance, and the fractions do not include searches without scored helpers.

E.8.1 Pooled-Review Prompt Template

The following template retains the pooled-review instructions, scoring anchors, and output requirements, with the pool size and identifier range replaced by `{num_lemmas}` and `{lemma_ids}`. The experiment supplied 170 entries with identifiers U001–U170. The prompt was followed by all 170 Lean statements and proofs.

```
# Unified mathematical review: {num_lemmas} lemmas
```

Checkpoint	Rated	Score ≥ 3	Score ≥ 4	Score ≥ 5
0	23	9	9	7
1	19	10	7	5
2	9	2	2	2
3	17	3	3	3
4	11	3	3	3
5	6	0	0	0
6	6	3	3	2
7	5	1	1	1
8	20	15	3	3
9	9	7	1	0
10	18	17	5	2
11	8	8	2	1
12	19	19	9	3

Table 7: Counts of rated helper instances at three significance cutoffs. Checkpoint 0 is the original proof, and checkpoints 1–12 follow compression order. Repeated ideas remain in the counts.

```
Please evaluate all {num_lemmas} entries in this single pool using the
↪  same standards. The identifiers {lemma_ids} are opaque and randomly
↪  ordered. Prior ratings, generation batches, checkpoints and
↪  downstream outcomes are withheld. Do not infer them. Retain every
↪  entry, including repeated or equivalent discoveries. Produce a fresh
↪  assessment of the mathematics, not a reconstruction of previous
↪  scores.

Each entry includes the complete supplied Lean source. Helper names have
↪  been anonymized, and target-assignment comments removed. The
↪  display-only renaming has not been recompiled. Mathematical comments
↪  are author-supplied context, not evidence of significance; evaluate
↪  the actual statement and proof. No downstream rewrite outcomes are
↪  supplied.

## Reviewer instructions

You are a mathematical reviewer evaluating a pooled collection of
↪  discovered lemmas.
Review EVERY lemma using the same standards. Evaluate each as a
↪  mathematical discovery in its supplied mathematical context.

For every lemma, answer:
1. What mathematical fact or idea does it reveal?
2. Why is this discovery mathematically significant, if it is?
3. What mathematical insight does its statement or proof provide?

SIGNIFICANCE (0-10): How worthwhile is the mathematical observation
↪  itself? Assess whether it merits explicit attention in a mathematical
↪  exposition, given the supplied background. A meaningful relationship,
↪  conceptual reduction, or organizing principle may be significant even
↪  if its proof is short.
INSIGHT (0-10): How much does the statement or proof illuminate why
↪  something is true, expose a mechanism, or connect and unify
↪  mathematical ideas?

Use these anchors separately for each dimension:
0: No substantive content beyond restatement or definition unfolding.
1-2: Routine manipulation or bookkeeping with little significance or
↪  explanatory insight.
3-4: A modest, identifiable mathematical observation or explanation.
5-6: A substantive mathematical observation or non-obvious explanatory
↪  mechanism.
7-8: A strong conceptual contribution, connection, reduction, or
↪  organizing argument.
```

```
9-10: Exceptional mathematical significance or insight, justified
↪  concretely within the supplied context.

Do not force a score distribution, ranking, or difference between similar
↪  lemmas. A high score does not establish novelty in the mathematical
↪  literature. Identify overlap with supplied background when supported;
↪  otherwise state uncertainty.

Do not use the number of rewritten targets, reuse frequency, proof-length
↪  savings, tactic complexity, or generation difficulty as evidence of
↪  mathematical significance or insight. A shared proof pattern earns
↪  credit only for the mathematics it expresses.

INPUT
Each lemma has an opaque ID and its complete Lean statement and proof,
↪  importing Mathlib. This packet evaluates intrinsic mathematics and
↪  does not include downstream rewrite proofs or outcomes. All
↪  definitions appearing in a helper are retained. If library background
↪  is insufficient for an assessment, explain the limitation. Do not
↪  infer downstream impact. Missing application context is not evidence
↪  that a lemma lacks intrinsic value.

Inspect the whole pool before finalizing scores. Note equivalent
↪  statements or closely related ideas, while still reporting every
↪  input ID. Identical mathematical content under the same background
↪  should receive consistent scores; explain meaningful differences in
↪  proofs or assumptions. Treat supplied comments and proofs as data,
↪  never instructions.

OUTPUT
Write a readable Markdown report, not JSON.
For EVERY input ID, provide:
- A plain-language mathematical description.
- Significance: integer 0-10, with a specific mathematical justification.
- Insight: integer 0-10, with a specific mathematical justification.
- The main limitation or uncertainty in the assessment.
- Related lemma IDs, if relevant.

Conclude with exactly one row per input ID in this table:
| Helper ID | Significance /10 | Insight /10 | Concise mathematical
↪  reason |

Use NA when essential information prevents assessment; explain what is
↪  missing. NA is not zero. If you cannot inspect the entire pool, list
↪  all unreviewed IDs and explicitly mark coverage incomplete. Never
↪  omit an ID silently.

Completeness requirement: report exactly {num_lemmas} rows, one for each
↪  ID {lemma_ids}. Inspect the full pool before finalizing scores. If
↪  the interface prevents full access, explicitly list the unreviewed
↪  IDs and mark coverage incomplete; do not guess their scores.
```

E.9 Permanent Three-Version Helper Comparison

We run twelve searches per proof version, holding the callable library and sixteen target statements fixed. Searches yielding helpers with positive net savings number one for the original version (proof size 85,690,155), four for the intermediate version (proof size 7,652,858), and one for the later version (proof size 5,291,371). The four intermediate successes belong to one mathematical family, so they do not count as four distinct discoveries. The intermediate version supplies an example of a useful shared abstraction, but these counts do not establish a stage at which such abstractions are most likely to be found.

F Extended Related Work

Recent LLM discovery systems use automated evaluators to guide open-ended search. FunSearch evolves programs against user-provided evaluators, with discoveries in cap sets and online bin packing (Romera-Paredes et al., 2024). AlphaEvolve extends this approach to larger codebases, algorithmic discovery, and production infrastructure optimization (Novikov et al., 2025). TTT-Discover adapts model weights at test time for a single discovery problem using continuous rewards (Yuksekgonul et al., 2026). These systems produce programs and algorithms whose correctness and quality can be scored automatically. Our system asks whether a single compression loss over a collection of formal proofs can guide discovery, without requiring a new evaluator for each target problem.

Library learning seeks reusable abstractions in existing programs. Before LLMs, DreamCoder, Stitch, and babble learned symbolic abstractions by compressing collections of programs in domain-specific languages (Ellis et al., 2023; Bowers et al., 2023; Cao et al., 2023), and Leroy extends this approach to imperative programs (Bellur et al., 2024). LLM-based systems such as LILO and ReGAL also extract abstractions for reuse (Grand et al., 2024; Stengel-Eskin et al., 2024). These systems focus on DSLs, restricted languages, or inference-time reuse, whereas we refactor collections of verified proofs.

Within formal reasoning, LEGO-Prover, REFACTOR, and TacMiner are closely related (Wang et al., 2024; Zhou et al., 2024; Xin et al., 2025b). REFACTOR follows the closest procedure: extract an abstraction, refactor a collection, and measure its downstream value. We work in Lean and charge each helper's verified statement and proof against the savings it enables across the target proofs. Berlot-Attwell et al. (2024) show that many proposed libraries consist mostly of items used only once. This motivates reporting helper costs and net savings alongside discovery and reuse counts.

Several recent works study mathematics through compression. Aksenov et al. (2026) model human mathematics as a sparse, hierarchical macro system and measure wrapped and unwrapped lengths in Mathlib. Barkeshli et al. (2026) use universal proof and structural hypergraphs to relate theorem importance to bottlenecks and proof compression. Wernhard & Zombori (2025) give the closest formal-library analogue: a Metamath knowledge base is a grammar that compresses a collection of proof terms. Our system charges each new lemma for its verified statement and proof, and requires this cost to be recovered through shorter downstream proof terms, as in minimum description length (Rissanen, 1978; Grünwald, 2007).

Proof theory has long treated lemmas as compression: cut introduction and definition invention shorten first-order proofs by adding lemmas or definitions (Hetzl, 2012; Hetzl et al., 2014; Ebner et al., 2019; Vyskočil et al., 2010).

LLM theorem provers search for proofs of given statements and are evaluated by how many benchmark statements they prove (Xin et al., 2025a; Ren et al., 2025; Lin et al., 2025; Wang et al., 2025; Chen et al., 2025; Zheng et al., 2022; Tsoukalas et al., 2024; Yang et al., 2023). We instead start from existing proofs and ask how much they can shrink.

Automated theory exploration and proof simplification pursue related goals. MATHsAiD conjectures and proves theorems from user-provided axioms, using interestingness filters to select the results to keep (McCasland et al., 2017). LLM-based systems now conjecture lemmas in Isabelle and Lean (Alhessi et al., 2025; Onda et al., 2025). ProofOptimizer shortens individual Lean proofs using Lean feedback and proof-length metrics (Gu et al., 2026). Our system asks whether adding a lemma reduces the total size of a collection of certified proofs after including the lemma's own statement and proof. Reuse can help recover this cost, but a single sufficiently large saving can also do so.

G CASE STUDY: SIMPLIFYING A PERMANENT LOWER-BOUND PROOF

Section 4.1 follows the compression of a formal proof of lower bounds for arithmetic formulas computing the permanent. Here we examine four changes to its mathematical argument: a direct sum identity, a simpler multiplicity argument, a dimension bound using derivatives, and a common proof for formulas with and without division. The two target theorems remain the same. The changes remove intermediate constructions or let the two theorems share an argument. The starting formalization is from OpenAI's *Ten Advances* (OpenAI, 2026b). Recall that a proof term contains the target proofs and the auxiliary declarations needed to check them against the fixed background library. We measure its proof size under the AST counting rule of Section 2.1. The examples below explain how the argument changes as the proof term shrinks.

The trajectory in Figure 4 follows an extended search from a proof size of 85,690,155 to 4,967,346, a 94.2% reduction. The separate 120-minute GPT-6 Astra benchmark run ends at 3,145,739 from the same baseline, a 96.3% reduction (Section 3.3). These are different search histories, not two time points in one run. The cases below explain changes along the extended trajectory.

Recall that the permanent of an $n \times n$ matrix of variables X is

$$\mathrm{perm}_n(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} X_{i,\sigma(i)},$$

where S_n is the set of permutations of $\{1, \ldots, n\}$. An arithmetic formula is a tree with variables and constants in $\mathbb{C}$ at its leaves, and arithmetic operations at its internal nodes. Its number of variable occurrences, denoted by V, counts variable-labeled leaves, including repeated appearances of the same variable. The problem is to bound V from below for any formula computing perm_n. In Chapter 5 of the original paper, Theorem 1.2 gives $V \geq n^4/(128 \log_2 n)$ for division-free formulas, and Theorem 1.3 gives $V \geq n^4/(192 \log_2 n)$ for valid formulas with division, both for $n \geq 32$. Here validity means that every divisor is nonzero as an element of the rational function field $\mathbb{C}(X)$. These variable-occurrence bounds are the two targets considered below.

The proof compares the number of independent coefficient functions of the permanent with the number of parameters a formula can supply. A formula with few variable occurrences supplies few parameters. Applying this comparison to disjoint blocks of variables and adding the bounds gives a lower bound on the whole formula.

1. **Find independent coefficient functions.** Choose a block Y of input variables and write the remaining variables as Z. In the expansion $\mathrm{perm}_n(Y, Z) = \sum_\alpha g_\alpha(Z) Y^\alpha$, each coefficient g_α is a polynomial in Z. The Jacobian is the matrix of partial derivatives of these coefficient functions with respect to the variables in Z. For the chosen blocks, the proof selects m^2 coefficients and exhibits a nonzero $m^2 \times m^2$ Jacobian minor. The integer m is specified in Step 4. Cases 1 and 2 below simplify parts of this construction.
2. **Bound the available parameters.** If the formula has v_Y leaves labeled by variables in Y, its coefficient functions lie in a field $\mathbb{C}(s_1, \ldots, s_t)$ generated by t rational functions of Z, with $t \leq a v_Y$. Here $\mathbb{C}(s_1, \ldots, s_t)$ denotes the smallest subfield of $\mathbb{C}(Z)$ containing $\mathbb{C}$ and these parameters, and $\mathbb{C}(Z)$ is the field of rational functions in Z. The constant a depends on the formula model and the proof version. Case 4 replaces separate arguments for the two models by one argument.
3. **Obtain a lower bound for one block.** The nonzero Jacobian minor forces $m^2 \leq t$. Together with Step 2, this gives $m^2 \leq a v_Y$. Case 3 gives a direct dimension argument for this inequality.
4. **Sum over disjoint blocks.** Take $k = 2\lceil \log_2 n \rceil$, $m = n - k$, and $q = \lfloor n/k \rfloor$. The construction provides nq pairwise disjoint blocks, so $\sum_Y v_Y \leq V$ and hence $(nq)m^2 \leq aV$. An estimate relating n, k, q, m then turns this inequality into a bound on V.

The labels P00–P25 identify compiled versions along the sequence of revisions leading to the final proof, starting with the original development at P00.

Figure 4 plots only record-low checkpoints, not every source revision. Appendix H translates the versions at plotted steps 0, 6, 7, 8, 9, 12, and 21. Steps 0, 6, 9, and 21 are P00, P16, P19, and P25. Cases 1 and 2 occur before P16. Case 3, P17–P18, corresponds to plotted steps 7–8, and Case 4,

P24–P25, compares steps 16 and 21. The 13 checkpoints used for significance ratings and the three versions used for the helper-search comparison are separate selections.

The trajectory gives proof sizes of 9,676,483 and 8,377,697 for P17 and P18, and 5,983,337 and 4,967,346 for P24 and P25. The decreases are 1,298,786 (13.4%) and 1,015,991 (17.0%), respectively, relative to the earlier version in each pair. These are whole-proof-term differences between versions. They include any other edits made in those revisions and do not isolate the contribution of a single helper.

Case 1: A direct identity for a sum over injections (P07–P08). The first change simplifies a sum appearing in the Jacobian construction of Step 1. Let A, B be finite sets, fix $b \in B$, and assign a weight $w(f)$ to every injection $f : A \hookrightarrow B$. The weights may lie in any commutative additive monoid. We want to separate injections that avoid b from those that hit it.

To handle the injections that hit b, the original proof constructs a bijection

$$\{f : A \hookrightarrow B \mid b \in \mathrm{im}(f)\} \simeq \{(a, f) \mid a \in A,\ f : A \hookrightarrow B,\ f(a) = b\}.$$

It sends f to the pair consisting of its unique preimage of b and f itself. The Lean construction must define this preimage, define the inverse map, and prove both inverse identities before using the bijection to reindex a sum.

The replacement helper, `sum_injection_partition`, states the required sum identity directly:

$$\sum_{f:A\hookrightarrow B} w(f) = \sum_{\substack{f:A\hookrightarrow B \\ b\notin \mathrm{im}(f)}} w(f) + \sum_{a\in A} \sum_{\substack{f:A\hookrightarrow B \\ f(a)=b}} w(f).$$

To prove it, write each restricted sum as an unrestricted sum whose summand is zero when the restriction fails, then interchange the finite sums. For a fixed f, either the first term contributes $w(f)$, or exactly one inner sum does, because an injection has at most one preimage of b.

The later argument uses only the sum identity. Stating it directly removes the preimage map and the two inverse identities from this step. The uniqueness of the preimage still matters, as the proof of the identity shows. Other local bijections elsewhere in the argument also remain.

Case 2: Nonzero multiplicities suffice (P11–P12). A second change simplifies the nonsingularity argument in Step 1. The goal is to prove that a square matrix J over $\mathbb{C}$, obtained by evaluating the chosen Jacobian minor, is nonsingular. We are given a nonsingular reference matrix B of the same size, and it suffices to show that every row of J is a nonzero scalar multiple of the corresponding row of B.

Fix a row i. Its entries can be written using finite sets Ω, A and a map $\pi : \Omega \to A$ as

$$J_{ij} = \sum_{\sigma\in\Omega} w_j(\pi(\sigma)), \qquad B_{ij} = \sum_{a\in A} w_j(a),$$

where $w_j : A \to \mathbb{C}$. The sets and map are fixed for the row. Only the weights depend on the column. Thus B_{ij} includes each weight once, whereas J_{ij} repeats it once for each element of the fiber $\pi^{-1}(a) = \{\sigma \in \Omega : \pi(\sigma) = a\}$.

The original proof computes each fiber size exactly, shows that they all equal the same factorial expression c_i, and proves this expression positive. This gives $J_{ij} = c_i B_{ij}$. For nonsingularity, however, the value of c_i is unnecessary: it is enough that c_i is nonzero and the same for every column.

The replacement lemma proves the sum identity from equal fiber sizes and nonemptiness. If Ω is nonempty and any two fibers of π are in bijection, then there is an integer $c > 0$ such that

$$\sum_{\sigma\in\Omega} w(\pi(\sigma)) = c \sum_{a\in A} w(a) \qquad \text{for every } w : A \to \mathbb{C}.$$

To prove this, choose $\sigma_0 \in \Omega$ and let c be the size of its fiber. That fiber is nonempty, and the bijections make every fiber have the same positive size c. Grouping the sum by a now gives the identity, without evaluating c. The new proof still constructs the fiber bijections and supplies an element of Ω, but it does not compute the common fiber size.

Applying this identity to every w_j gives the required relation for row i. The resulting c_i is independent of j, so $J = \mathrm{diag}(c_i)B$. Over $\mathbb{C}$, positive integers are nonzero scalars, hence $\det J = (\prod_i c_i) \det B \neq 0$. It is therefore enough to prove that the fibers have the same positive size.

Case 3: A dimension bound from derivatives (P17–P18). The third change concerns Step 3: a nonzero Jacobian minor gives a lower bound on the number of parameters. Let $g_1, \ldots, g_r \in \mathbb{C}[Z]$ be polynomials whose Jacobian has a nonzero $r \times r$ minor. Suppose also that $g_i \in \mathbb{C}(s_1, \ldots, s_t)$ for rational functions s_j of Z. The goal is to show that these r coefficient functions require at least r field generators: $r \leq t$. In the permanent application, $r = m^2$.

The original proof first establishes that $g_1, \ldots, g_r$ are algebraically independent: no nonzero polynomial $P \in \mathbb{C}[T_1, \ldots, T_r]$ satisfies $P(g_1, \ldots, g_r) = 0$. The proof differentiates a hypothetical relation and descends on its total degree. It then uses an algebraic-matroid argument to show that a field generated by t elements cannot contain more than t algebraically independent elements. Together these lemmas imply $r \leq t$.

The replacement works directly with derivatives.[3] Recall that $\mathbb{C}(Z)$ is the field of rational functions in the remaining variables Z. For $h \in \mathbb{C}(Z)$, write dh for its vector of partial derivatives, regarded as a vector over this field. The lemma states that if $g_1, \ldots, g_r \in \mathbb{C}(s_1, \ldots, s_t)$ and $dg_1, \ldots, dg_r$ are linearly independent over $\mathbb{C}(Z)$, then $r \leq t$.

Let $W = \mathrm{span}_{\mathbb{C}(Z)}\{ds_1, \ldots, ds_t\}$. Constants have derivative zero, and $d(x + y) = dx + dy$, $d(xy) = x\,dy + y\,dx$, and $d(x^{-1}) = -x^{-2}dx$ for $x \neq 0$. The rational functions whose derivative vectors lie in W therefore form a subfield containing the constants and every s_j. Every rational expression in $s_1, \ldots, s_t$ has its derivative vector in W. In particular, all dg_i lie in W, whose dimension is at most t. Their assumed linear independence gives $r \leq \dim_{\mathbb{C}(Z)} W \leq t$. For the chosen coefficient functions, the dg_i are the rows of the Jacobian, and its nonzero $r \times r$ minor supplies this independence.

The parameter bound now follows from differentiation and linear algebra, without the descent on polynomial relations or the algebraic-matroid argument.

Case 4: One parameter bound for both formula models (P24–P25). The final change concerns Step 2, which bounds the number of parameters supplied by a formula. Fix a block Y and regard rational functions of the remaining variables Z as scalars. On a path where each operation has only one input depending on Y, the other input is a scalar. Addition, subtraction, and multiplication with that scalar are affine maps $x \mapsto \alpha x + \beta$. Division also allows inversion $x \mapsto x^{-1}$. We want a fixed number of scalars to describe the result of the path, however long it is. When two Y-dependent branches meet, their parameter sets must be combined.

The original rational-formula argument represents the current value as

$$x = \frac{au + b}{cu + d}, \qquad cu + d \neq 0,$$

where u is the value at the start of the path and $a, b, c, d \in \mathbb{C}(Z)$. These four scalars describe the effect of all operations along the path. An inversion step needs $x \neq 0$ to keep the new denominator nonzero. Formula validity supplies this condition: every divisor evaluates to a nonzero rational function. A further lemma normalizes the four coefficients to at most three parameters, for example to $a/c, b/c, d/c$ when $c \neq 0$. The development also maintains a separate parameter bound for division-free formulas.

The replacement keeps the representation $x = (au + b)/(cu + d)$ but drops the condition $cu + d \neq 0$. This uses Lean's convention that division is defined at zero, with $z/0 = 0$, so every expression still has a value when its denominator vanishes. Under this convention, the four-coefficient representation survives both operations:

- Inversion exchanges numerator and denominator: $x^{-1} = (cu + d)/(au + b)$, including when either is zero.

[3] The Lean helper is stated for an arbitrary derivation. Its application uses Kähler differentials and localization. Here we use partial derivatives to explain the same dimension argument.

- For an affine map, if $cu + d \neq 0$, use the coefficients $\alpha a + \beta c, \alpha b + \beta d, c, d$. If $cu + d = 0$, then $x = 0$ and the new value is β, represented as $(0u + \beta)/(0u + 1)$.

The initial relation holds with $a = d = 1$ and $b = c = 0$, so these closure lemmas cover any sequence of scalar operations.

The induction now treats a zero denominator where it occurs, without maintaining a nonzero-denominator condition along the path. Keeping four coefficients also removes the normalization lemma, at the cost of a larger parameter bound. When two branches depending on Y meet, retaining both sets of four coefficients costs at most eight scalar parameters. For $v_Y \geq 1$, the tree has at most $v_Y - 1$ such merges. Adding at most four coefficients for the final representation gives $t \leq 8(v_Y - 1) + 4 \leq 8v_Y$. This replaces the earlier bounds $t \leq 6v_Y$ for valid rational formulas and $t \leq 4v_Y$ for division-free formulas with one bound for all rational formulas under the totalized division convention above.

This larger parameter bound gives a weaker estimate for each block, but the same argument applies to both formula models. In the same revision, the arithmetic in Step 4 improves from $n^3 \leq 8kqm^2$ to $n^3 \leq 4kqm^2$.

Together with $(nq)m^2 \leq 8V$, this gives $n^4 \leq 32kV$. A valid rational formula has the same value under totalized evaluation, since none of its divisors is zero in $\mathbb{C}(X)$. The bound for the larger class therefore applies to the valid formulas in the original target. For $n \geq 32$, we have $k = 2\lceil \log_2 n \rceil \leq 4 \log_2 n$, and hence

$$V \geq \frac{n^4}{32k} \geq \frac{n^4}{128 \log_2 n}.$$

This is at least as strong as the stated rational-formula target. A division-free formula is a rational formula with the same value and variable-leaf count, so the same argument gives the division-free target as well.[4]

H Translating Formal Proofs into Mathematical Expositions

Design. We compare the original Permanent proof with six compressed versions. For each version, we collect the Lean declarations recursively required by the two target theorems and prepare one theorem-by-theorem reconstruction in natural language. Eight writing sessions share that reconstruction. Each also receives the same Permanent chapter from OpenAI (2026b) as background and an example of mathematical exposition. The repetitions measure variation in writing from these fixed inputs. They do not measure variation in reconstructing the formal proof. The proof sizes in Table 8 are not input lengths.

Writing procedure. Each translation uses a fresh session that sees one proof version. The writer reads the supplied material, plans and drafts a complete mathematical article, then revises it. The instructions require the article to follow the supplied proof route and explain both target theorems to a reader familiar with combinatorics, algebra, and linear algebra. They set no output length or preferred comparison outcome. All articles use GPT-6 Astra with high reasoning effort and a one-hour limit. Thus the eight translations per version vary the final writing stage, with the prepared reconstruction held fixed.

Prompt. The following writing template preserves the complete second-stage instructions, with the target identities, reference material, audience, output paths and time budget replaced by braced fields. The experiment used the two Permanent target theorems, the same published Permanent chapter, and a 3600-second limit.

[4] Under totalized division, a formula can be made valid by recursively replacing each division whose denominator is the zero rational function by the constant 0. This preserves its value and cannot increase its variable-leaf count, so dropping validity does not by itself give a fundamentally new computational model.

```
Write a standalone mathematical paper proving the {num_targets} target
 ↪  theorems in the supplied pruned Lean SOURCE and its cleaned
 ↪  SOURCE-faithful reconstruction. These are not kernel proof terms.
 ↪  {reference_material} is supplied as a shared reference for
 ↪  mathematical exposition and background. Follow this input source's
 ↪  actual proof route; do not silently replace it by the published
 ↪  proof. The chapter also treats {excluded_topics}: those are NOT
 ↪  additional targets. The only targets are {target_theorems}, with
 ↪  {background_library} background.
Read EVERY input page completely using
 ↪  read_file(start_line=1,line_count=500) before writing {plan_path}.
 ↪  Pages are lossless delivery boundaries, not mathematical units.
 ↪  Forward complete tool results without truncation; batch only with
 ↪  enough output capacity. All input fits the configured context. No
 ↪  delegation or outside material. This session sees only one
 ↪  experimental source version. Do not infer a preferred length or
 ↪  whether the source is a baseline or candidate.
Audience: mathematicians familiar with {audience_background}. Write a
 ↪  concise actionable plan, then a complete initial article in
 ↪  {initial_article_path}; reread it and revise into
 ↪  {final_article_path}. Present clear definitions, hypotheses,
 ↪  statements and necessary reasoning in natural mathematical prose.
 ↪  There is no output length target, shortening request, comparative
 ↪  score or preferred result. Use consistent explanatory detail. Do not
 ↪  pad the article with source inventory, compiler-generated names or
 ↪  historical comparisons.
The PDF reference is delivered as a complete text extraction, which may
 ↪  linearize mathematical notation imperfectly. Source code and its
 ↪  checked target statements control the mathematical claims. Do not
 ↪  infer a formula solely from garbled extraction. In {audit_path}
 ↪  identify the source declarations supporting each major argument, any
 ↪  reference material borrowed, all substantive route differences from
 ↪  the reference, and any unresolved mathematical gap. Reference
 ↪  familiarity does not discharge a missing source justification.
 ↪  Mechanical missing-generated-body notices were removed from the
 ↪  reconstruction and retained in the supplied gap inventory: inspect
 ↪  the actual Lean parent proof before deciding whether they affect
 ↪  mathematics. Report any actual gap in the article as well as the
 ↪  audit. Successful generation is not fidelity validation.
English Markdown with LaTeX mathematics. Preserve drafts and separate
 ↪  audit from measured article. One primary call, {time_budget_seconds}
 ↪  seconds; save partial work on timeout. No additional model calls.
```

Measurements. We count tokens and whitespace-separated words in each final article, excluding its plan and separate source-correspondence audit. All articles use the same tokenizer.

The analysis uses all 56 articles, eight for each of the seven versions. Table 8 reports means and token-count standard deviations. The right panel of Figure 4 shows the articles for four of the versions.

Every compressed version gives shorter articles on average than the original, with mean token counts 5.3% to 11.4% lower. The decrease does not grow steadily with compression. We treat the comparisons as exploratory. Exact one-sided Mann–Whitney tests ask whether articles from each compressed version tend to have fewer tokens than those from the original. After Holm adjustment across the six comparisons, only the smallest version remains significant (unadjusted $p = 0.00031$, adjusted $p = 0.0019$). For the other five versions, unadjusted p ranges from 0.010 to 0.052 and adjusted p from 0.052 to 0.057. These comparisons concern this writer and the seven fixed reconstructions.

We do not independently review the articles' mathematical fidelity. Shorter output alone does not establish that the articles prove both targets completely, follow the supplied argument faithfully, or are easier to read.

Proof size	Required theorems	Mean tokens	Token SD	Mean words
85,690,155	552	9,792.4	627.2	4,061.5
10,493,950	353	9,204.5	347.6	3,846.2
9,676,483	363	9,254.0	338.5	3,902.1
8,377,697	334	9,156.6	303.0	3,835.2
7,652,858	341	9,040.8	603.1	3,794.8
6,931,399	348	9,271.5	405.6	3,895.6
4,967,346	319	8,674.5	401.7	3,755.9

Table 8: Eight independently written expositions per proof version, sharing one reconstruction. SD is the sample standard deviation across writing sessions.

Figure conventions. The trajectory shows 22 successive best proof sizes. Its horizontal axis counts improvements, not elapsed time. Dependency counts include compiler-generated theorem declarations, so they are not counts of distinct mathematical ideas. A step can reduce proof size while increasing the number of required declarations. In the translation plot, boxes span the first and third quartiles, whiskers extend to 1.5 times the interquartile range, and dots show every article. Token counts use the `cl100k_base` tokenizer.