Proceedings of CONF-APMM 2026 Symposium: Applied Mathematics and Quantitative Modeling
DOI: 10.54254/2753-8818/2026.PU38278

Large Language Models for Mathematical Discovery: Generation, Verification, and Knowledge Provenance

Tengyu Wu

Cogdel School Chengdu, Chengdu, China
wty928292681@hotmail.com

Abstract. Formal mathematics has a data problem, and large language models have become one of the tools applied to it. LeanConjecturer mined 40 seed files from Mathlib and produced 12,289 candidate statements, of which 3,776 resisted the automated tactic used as a filter. AlphaGeometry solved 25 of 30 olympiad geometry problems by letting a language model propose auxiliary constructions whenever symbolic deduction stalled. FunSearch improved on earlier results for the cap set problem by having an evaluator score programs written by a model. Evidence of this kind still supports a narrower claim than mathematical discovery. Five studies are compared in this review: LeanConjecturer, the Conjecturing-Proving Loop, AlphaGeometry, FunSearch, and Unlearning as Ablation. Using literature analysis and comparative case analysis, task objectives, generation mechanisms, verification procedures, and evidential limits are examined side by side. Two findings stand out. External verification is what makes candidate selection and iterative repair function in each system, and each system demonstrates a different slice of the work: conjecture supply, proof repair, geometric construction, program optimization, and, in the case of Unlearning as Ablation, a proposal for auditing knowledge provenance. Correctness, novelty, and independence from previously acquired target knowledge remain three separate questions that require three separate kinds of evidence. A more reliable evaluation framework would combine auditable knowledge boundaries, dependency checking, and expert assessment of mathematical significance.

Keywords: Large language models, Mathematical discovery, Formal verification, Machine unlearning

1. Introduction

Mathematical discovery bundles several activities together: framing problems, guessing conjectures, proving them, deciding which results matter. Large language models (LLMs) produce mathematical text and code; external verification tools then check what they produce. Lean 4 provides an extensible environment for theorem proving and programming, and Mathlib supplies reusable definitions and theorems, so models and mathematical knowledge libraries can interact [1, 2].

MiniF2F evaluates proving capabilities through formalized problems across proof systems, while LeanDojo supports proof search through data extraction, environment interaction, and lemma retrieval [3, 4]. Neither addresses discovery as such. Both start from statements handed to the

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2026.PU38278

system. Discovery requires more: something has to choose candidates worth pursuing, and to place those candidates in relation to what is already known. Studies often report progress through successful proofs or higher scores. Whether such indicators travel all the way to discovery is another matter, and it shapes both task design and the way results are read. Using literature analysis and comparative case analysis, this review examines five representative studies, comparing generation, verification, and feedback mechanisms while keeping mathematical validity apart from knowledge provenance. The cases span conjecturing, proving, construction, and evaluation. Coverage stops there: what follows is a framework for reading the evidence these systems provide, not a survey of the field.

2. Representative studies on mathematical discovery with large language models

2.1. LeanConjecturer: formal conjecture generation

LeanConjecturer was built around a shortage of formal training data. Variable declarations and namespace information are pulled out of Mathlib seed files, Mathlib and Aesop imports are added, and the model writes only Lean theorem statements. Screening then proceeds in sequence: syntax, redundancy against the library, automated proving difficulty. Candidates that survive re-enter later contexts. Keeping generation apart from verification spares the model the job of organizing environment, statement, and proof at the same time [5].

The numbers from the study: 40 seed files yielded 12,289 conjectures. Of those, 10,950 passed syntax, 4,130 cleared the novelty check, and 3,776 could not be proved by the designated automated tactic. The authors also trained a prover on the generated conjectures, which shows how a bigger pool of candidates can feed later proof learning [5].

The screens have limits worth naming. The Lean proof-search tactic "exact?" asks whether existing results prove a candidate outright; "aesop" gauges automated proving difficulty [5]. When both fail, all that has been shown is that the candidate cleared that screen. Truth is not established, and mathematical novelty is not either. Higher proving success on training examples needs independent testing before generalization to new concepts can be claimed. The contribution, then, lies in candidate supply. Sifting out restatements of known results, vacuous statements built on inconsistent assumptions, and combinations of little use would take further filtering, alongside expert judgment of mathematical significance.

2.2. The Conjecturing-Proving Loop and in-context learning

The Conjecturing-Proving Loop (CPL) couples candidate generation, proof attempts, Lean error feedback, and knowledge accumulation. Verified theorems and proofs are appended to later contexts, so proof experience is reused without touching model parameters. The formally published 2026 version of the study is analyzed here [6].

Its target is a known topological result: the intersection of two α-open sets is α-open, approached from definitions of generalized open sets. Every run was allowed 14 million application programming interface (API) tokens, with 20 independent runs per condition. CPL reached the target in five runs. A simple loop that generates statements and proofs together reached it in none. Re-proving generated theorems succeeded 2,106 times out of 2,123 attempts when the accumulated library was available, against 1,935 with only the initial definitions [6].

Two lessons follow, one positive and one cautionary. Verified context genuinely helps later proving. Candidate yield, by contrast, says little about whether a hard target can be reached.

DOI: 10.54254/2753-8818/2026.PU38278

Evidence here is thin, resting on one target. Anonymized definitions, runs that fail without accumulated context, and shifting proof paths describe behavior under the tested conditions; whether related knowledge sat in pretraining is not something this design can settle. Novelty is judged against the specified library and search process, so claims about results new to the mathematical literature still need outside review.

2.3. AlphaGeometry: auxiliary constructions and symbolic deduction

AlphaGeometry pairs a neural language model with a symbolic reasoning engine for Euclidean plane geometry. Consequences are derived from known relations by the symbolic engine. Whenever deduction stalls, the language model proposes auxiliary points or lines, and deduction resumes. Training on automatically synthesized problems and proofs lowers reliance on human proof demonstrations [7].

In the test set used by the study, 25 of 30 International Mathematical Olympiad geometry problems were solved. Learned auxiliary construction and symbolic deduction turn out to be complementary: constructions widen the search space, the rule-based system checks each deduction, and longer proof chains become reachable [7].

What has been achieved concerns given goals and intermediate objects. Competition success rates measure reasoning inside one geometry environment. Open-ended discovery also requires judgments about statement selection and research significance, which such numbers do not cover. Representation, available rules, and the distribution of synthetic data all bound the capability on display. Research problems in algebra or analysis would demand new representations and new verification machinery. Read as method, the lesson is a division of labor: exploratory construction goes to the generative model, rule-governed deduction to the symbolic system, and errors in each component are examined separately.

2.4. FunSearch: evaluator-guided search for construction programs

FunSearch represents mathematical or algorithmic tasks as executable programs. A language model rewrites program fragments; an evaluator checks and scores them; strong examples drawn from a program database become the next prompts. Multiple evolving populations and iterative selection keep search diversity, so invalid candidates drop out while promising constructions improve [8].

Constructions that improved previous results for the cap set problem were found this way, and heuristics beating commonly used baselines were developed on the tested online bin-packing tasks. Because outputs are programs that generate constructions, they can be executed, inspected, and analyzed further [8].

Two conditions govern applicability: a reliable, affordable evaluator, and a program framework expressive enough for useful candidates. Finite constructions can be checked exhaustively or with specialized procedures. Improvements in bin-packing strategies on test distributions raise generalization questions instead. Broader mathematical conclusions need arguments of a comparable kind, so score improvements belong next to verification coverage, which keeps claims inside the scope of the optimization task. Set against the re-proving of known theorems, a verifiable construction that exceeds a publicly established bound is more direct evidence of novelty: existing programming experience gets reused, and the outcome still adds to mathematical knowledge.

DOI: 10.54254/2753-8818/2026.PU38278

2.5. Unlearning as Ablation: assessing knowledge provenance in rediscovery

Unlearning as Ablation treats memorization versus construction as an evaluation question. Target knowledge is unlearned first; the model is then tested on whether it can rederive the result using permitted axioms and tools. Its "forget-closure" covers the target statement, paraphrases, supporting lemmas, and multihop exposure paths, which amounts to an argument that deleting the target text alone controls knowledge leakage poorly [9].

The contribution is conceptual and methodological rather than empirical. Mathematical and algorithmic examples describe procedures that are proposed, not executed, and no systematic experiment on rediscovery after actual unlearning is reported. What remains is a direction for falsifiable evaluation. TOFU (A Task of Fictitious Unlearning for LLMs), a related study, uses fictitious author profiles to assess forgetting quality and retained utility, and offers an operational reference for judging removal alongside capability preservation [10].

Three implementation problems stand out. Completeness of the forget-closure is hard to establish. The line between foundational knowledge and knowledge that supports the target is hard to draw. Capability degradation from unlearning is hard to separate from genuine knowledge loss, since failed rederivation may reflect both; and failing to elicit a target through a limited probe set does not show that its information has been removed. Historical restrictions on external knowledge act on retrieval and tool dependencies, whereas parameter-level unlearning intervenes directly on the model. Different routes call for different audit evidence.

Table 1 collects the outputs, verification methods, and evidential limits of the studies discussed above. The comparison is synthesized for this review.

Table 1. Outputs, evaluation methods, and evidential limits of representative studies

Study	Main output	Verification or evaluation	Evidential limits
LeanConjecturer	Formal conjectures	Syntax and automated tactic screening	Conjectures may be false
Conjecturing-Proving Loop	Theorems and proofs	Lean checking and re-proving	Evidence centers on one target
AlphaGeometry	Auxiliary constructions and proofs	Symbolic geometry deduction	Domain-specific representation
FunSearch	Construction programs	Execution and task scoring	Evaluator-dependent coverage
Unlearning as Ablation	Knowledge-ablation framework	Proposed unlearning and leakage audits	Empirical validation required

3. Synthesis and future research directions

LeanConjecturer, CPL, AlphaGeometry, and FunSearch all constrain generation through external feedback, applied respectively to conjecture screening, proof repair, geometric construction, and program optimization. Unlearning as Ablation asks whether successful outputs support judgments about knowledge provenance [5-9]. Because these systems differ in task, budget, and verification strength, a single "discovery success rate" hides more than it reveals. Cross-system comparisons are more informative when task categories and the specific evidence behind each result are kept in view.

DOI: 10.54254/2753-8818/2026.PU38278

At least three dimensions deserve separate treatment. Mathematical validity asks whether a result holds under explicit assumptions, and formal proofs, finite computations, and empirical tests offer support of different strength. Novelty asks what the contribution is relative to a mathematical library, to training materials, or to the published literature, taking nontriviality and research significance into account. Independence from previously acquired target knowledge asks whether that knowledge could have entered generation through model parameters, retrieval, or tools. A check passed on one dimension licenses a conclusion about that dimension only.

For the rediscovery of historical theorems, an audit can cover historical cutoffs, permitted materials, and proof dependencies, while candidates, repairs, and the lemmas actually used are recorded. Modern formalization tools should restrict the mathematical content exposed to the model, and a match with a known historical result should be evaluated apart from originality. Even when every external dependency complies with the restrictions, knowledge stored in the parameters may still contribute, so conclusions belong to verifiable constructions under constrained conditions.

Future work would benefit from reporting failed candidates along with model-call and verification costs, and from combining tasks across domains, verifiable training or unlearning conditions, and expert assessment so that results become comparable. When feedback mechanisms are compared, model, initial knowledge, and budget should be held fixed while verified proofs, error messages, and retrieved lemmas are examined separately. Library versions, automated tactics, and time limits deserve documentation as well, if labels such as "unproved," "unmatched within the library," and "new to the literature" are to be checkable. Five representative studies are covered here. No systematic survey of all related work is attempted, and no system was reproduced in a common environment. The comparisons clarify methods and evidence; they do not rank model capabilities.

4. Conclusion

Five studies were examined to sketch how large language models contribute to mathematical discovery. Formal conjecture generation widens the supply of candidates. Accumulated verified proofs help later search. Neural–symbolic cooperation and program evaluation enable construction within bounded domains. Knowledge ablation raises the bar for provenance claims about rediscovered results. Taken together, they show where the research value lies: generation paired with reliable feedback. Three things remain separate—successful verification, novelty, independence from previously acquired target knowledge—and each needs its own evidence, so conclusions should stay inside the task and verification scope. The limits of this review are worth stating plainly. Five cases are all that are covered. The systems are not reproduced in a common environment. The related literature is not searched systematically, and computational costs and cross-domain generalization are not compared quantitatively. Future work can join knowledge-boundary audits, verified training or unlearning conditions, replication across domains, and expert assessment to examine correctness, novelty, and provenance together. Publishing failed cases, verification dependencies, and evaluation criteria would make the research easier to audit, give a firmer basis for using mathematical research assistants, and show which conclusions the available evidence actually supports.

References

[1] De Moura, L., & Ullrich, S. (2021). The Lean 4 theorem prover and programming language. In *Automated Deduction—CADE 28* (pp. 625–635). Cham: Springer. https://doi.org/10.1007/978-3-030-79876-5_37.

[2] The mathlib Community. (2020). The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 367–381). New York: ACM.

DOI: 10.54254/2753-8818/2026.PU38278

https://doi.org/10.1145/3372885.3373824.

[3] Zheng, K., Han, J. M., & Polu, S. (2022). MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*. https://mlanthology.org/iclr/2022/zheng2022iclr-minif2f/.

[4] Yang, K., Swope, A., Gu, A., et al. (2023). LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems, 36*, 21573–21612. https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract.html.

[5] Onda, N., Kasaura, K., Oriike, Y., et al. (2025). LeanConjecturer: Automatic generation of mathematical conjectures for theorem proving. arXiv:2506.22005v1. https://arxiv.org/abs/2506.22005v1.

[6] Kasaura, K., Onda, N., Oriike, Y., et al. (2026). Discovering new theorems via LLMs with in-context proof learning in Lean. In *Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA)* (pp. 40–49). Association for Computational Linguistics. https://aclanthology.org/2026.naloma-1.5/

[7] Trinh, T. H., Wu, Y., Le, Q. V., et al. (2024). Solving olympiad geometry without human demonstrations. *Nature, 625*(7995), 476–482. https://doi.org/10.1038/s41586-023-06747-5

[8] Romera-Paredes, B., Barekatain, M., Novikov, A., et al. (2024). Mathematical discoveries from program search with large language models. *Nature, 625*(7995), 468–475. https://doi.org/10.1038/s41586-023-06924-6

[9] Yang, R. (2025). Unlearning as ablation: Toward a falsifiable benchmark for generative scientific discovery. arXiv:2508.17681v4. https://arxiv.org/abs/2508.17681v4

[10] Maini, P., Feng, Z., Schwarzschild, A., et al. (2024). TOFU: A task of fictitious unlearning for LLMs. *Conference on Language Modeling*. https://openreview.net/forum?id=B41hNBoWLo