Open Problems Solved by LLMs?
A Survey of Verifiable Mathematical Discovery

Ioannis Tzachristas [1,2*], **Georgios Tzachristas** [1,3*], **Aifen Sui** [1†]
[1]Huawei European Research Institute
[2]Technical University of Munich, Germany
[3]National Technical University of Athens, Greece

Abstract

Recent years have produced a small but rapidly growing set of results where Large Language Models (LLMs)—usually embedded in a search-and-verification loop—advance the state of the art on problems previously regarded as "open" in the pragmatic sense of lacking a best-known construction, bound, or proof certificate. This paper surveys that emerging line of work with a Big Picture emphasis: *what makes these successes possible, what should count as "solved", and what design patterns generalize?* We (i) propose an evidence ladder for interpreting "LLM solved an open problem" claims, (ii) map mathematical subfields by difficulty dimensions that matter for LLM-based discovery, (iii) curate a timeline of key breakthroughs leading to verifiable discovery systems, and (iv) synthesize the techniques and frameworks—tool use, retrieval, search, and verification—that repeatedly appear in successful case studies. We give particular attention to formal-methods backends common in security and verification contexts, including Linear Temporal Logic (LTL) and Satisfiability Modulo Theories (SMT) solvers, as scalable middle-layer verifiers between lightweight tests and proof assistants. We close with an evaluation and reproducibility checklist aimed at making the next wave of claims easier to trust, reproduce, and build upon, while separating peer-reviewed or certificate-backed results from fast-moving community reports that are useful signals but not yet stable evidence.

1 Introduction

The phrase *"LLMs solve open problems"* is simultaneously exciting and misleading. Exciting, because in a few settings LLM-driven systems have produced artifacts that are *objectively* better than prior best-known ones: new extremal combinatorial constructions, improved numerical bounds, or certified inequalities. Misleading, because "open problem" conflates many different notions of difficulty, and because LLMs almost never act alone. Most credible successes are not *one-shot* "proof writing"; they are *closed-loop discovery* systems that turn an LLM into a proposal generator whose outputs are filtered by an external evaluator.

Mathematics is a particularly revealing arena for this phenomenon. Recent perspective work on generative modeling for mathematical discovery makes a complementary systems-level case for treating learned models as proposal generators in checkable search loops (Ellenberg et al., 2025). Unlike many NLP tasks, math often comes with (or can be engineered to have) strong correctness tests: a candidate construction can be checked against constraints; a numerical bound can be verified by code; a formal proof can be validated by a proof assistant. This makes math a natural testbed for the broader scientific question: *when do generative models become reliable discovery tools?*

Survey scope. We focus on LLM-driven systems that (a) claim progress on problems where the status quo was "best known" or unknown in the literature or community benchmarks, and (b) provide an evaluator or certificate that can in principle be checked independently. We intentionally de-emphasize purely anecdotal "ChatGPT proved X" stories unless they culminate in an externally verifiable artifact.

Contributions.

- **Evidence ladder.** We define tiers of evidence for "solved" and connect them to reproducibility expectations.
- **Hardness map.** We analyze which dimensions make math hard for LLM systems and argue that *verifiability* is the dominant factor in current successes.

*Equal contribution.
†Correspondence: aifen.sui@huawei.com.

Proceedings of The Big Picture v2: Crafting a Research Narrative, pages 10–21

- **Breakthrough timeline.** We summarize key milestones from transformers to tool use to evolutionary search that culminate in verifiable discovery.
- **Technique taxonomy.** We distill recurring frameworks—LLM+search+verifier, retrieval over formal libraries, test-time adaptation—and outline how they interact.
- **Community signals.** We record how non-paper announcements, problem wikis, and social-media discussions should be handled without confusing attention with verification.

How the axes fit together. The figures and taxonomies below are intended as coordinates of one reference architecture, not as independent lists. Figure 1 asks how strong the evidence is; Figure 3 asks whether the target domain admits a cheap verifier; Figure 2 locates that verifier by rigor and cost; and Figure 5 shows how generation, search, and checking interact. The timeline in Figure 4 is therefore best read as a sequence of improvements to components of the same loop, with milestone citations and component labels made explicit in Tables 4–5.

2 What should count as "an open problem solved by an LLM"?

A central aspect of this topic is definitional. In everyday mathematical practice, an "open problem" can mean anything from "nobody knows the exact answer" to "the best known bound might be improvable" to "no proof exists." Moreover, the role of the LLM can range from minor assistance (e.g., drafting exposition) to being the main generator of candidates in an automated loop.

2.1 A practical evidence ladder

Figure 1 summarizes a spectrum of claims. The ladder does *not* rank mathematical importance; it ranks how directly a reader can verify that the system achieved a new result.

We operationalize the ladder as tiers (Table 1). The goal is not to police language, but to make papers easier to compare and reproduce.

2.2 Community-reported claims

A practical complication is that influential claims often appear first on Mathstodon, GitHub wikis, LinkedIn, Hacker News, personal pages, or shared chat transcripts, before a conventional paper exists. We treat these as *community signals*: useful for discovering what problems people are discussing, but not sufficient evidence by themselves. For the community examples below, we used the Erdős wiki and Problem #728 forum as problem-status sources, Mathstodon and Hacker News as discussion fora, and LinkedIn posts as examples of social amplification rather than as independent validators (Terence Tao and contributors, 2026; Erdős Problems community, 2026; Tao, 2026; Hacker News contributors, 2026; Boland, 2026; Hoefler, 2026). Recent examples include the Erdős-problems community wiki, which explicitly labels AI contributions as full, partial, or incorrect; the public discussion of Erdős Problem #728 and related factorial-divisibility questions; and Knuth's "Claude's Cycles" note on decomposing a directed toroidal grid into Hamiltonian cycles (Terence Tao and contributors, 2026; Tao, 2026; Sothanaphan, 2026; Knuth, 2026; Morrison and contributors, 2026). A parallel, more curated example is the GPT-5.2 learning-curve monotonicity case, where a public company post pointed readers to a technical write-up on an open statistical-learning question (OpenAI, 2025). LinkedIn posts and other reposts amplified these examples quickly, but their role in this survey is pointers to the problem and artifact, not independent validation (Boland, 2026; Hoefler, 2026).

2.3 Why verification dominates

Across surveyed systems, the strongest results are those where verification is cheap relative to generation. This echoes earlier work on *verifier-based* scaling (e.g., generating many candidates and selecting with a verifier) in mathematical reasoning benchmarks such as GSM8K and MATH (Cobbe et al., 2021; Hendrycks et al., 2021). In discovery settings, the same idea becomes a closed-loop optimization: generate, evaluate, mutate, repeat.

2.4 Verifier spectrum: rigor *vs.* throughput

Even when a problem is "verifiable," verifiers differ sharply in (i) *soundness/rigor* and (ii) *evaluation cost per candidate*. This trade-off strongly shapes which mathematical areas see early progress: systems gravitate toward regimes where they can test *many* candidates per unit compute while retaining meaningful correctness guarantees.

Figure 2 summarizes a practical landscape. In practice, the main "sweet spot" for present-day open-problem progress is typically *certificates and deterministic checkers* (Tier B/C): they are much more rigorous than heuristic scoring, and far

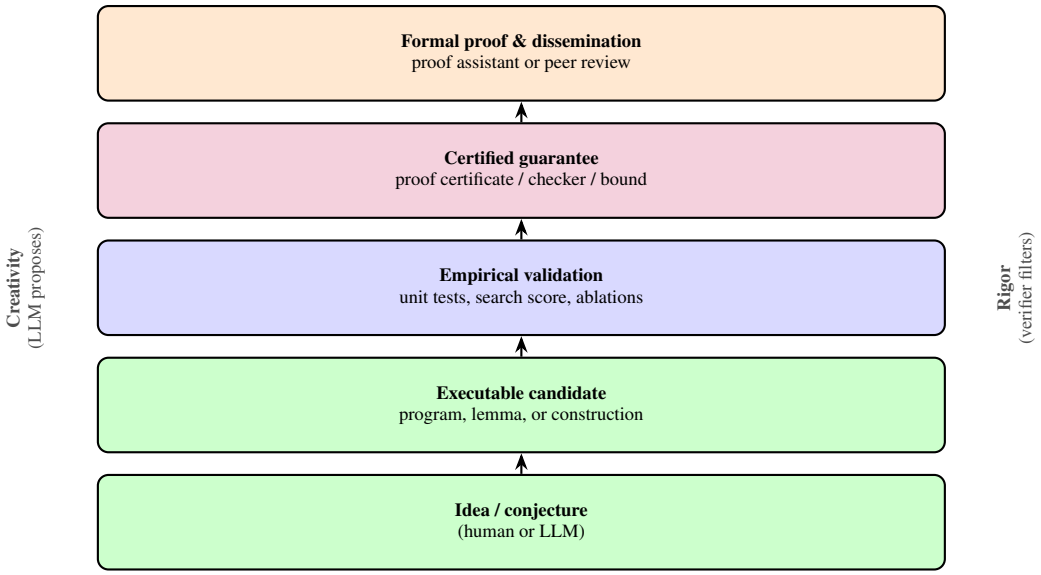


Figure 1: An evidence ladder for interpreting "LLM solved an open problem" claims. The most trustworthy claims reach the upper rungs via formal verification or independently checkable certificates.

Tier	What the paper provides
A (formal)	A formal proof checked by a proof assistant (e.g., Lean/Coq/Isabelle), or a proof-carrying artifact where verification is fully automated.
B (certified)	A machine-checkable certificate (e.g., explicit construction + deterministic checker, inequality certificate, or reproducible code proving a bound).
C (reproducible best-known)	A new best-known result backed by an open evaluator, strong baselines, and ablations; verification may still rely on extensive computation but is independently runnable.
D (suggestive)	Plausible conjectures, heuristics, or partial progress without a verifiable certificate; valuable, but not "solved."

Table 1: Evidence tiers for "open problem" claims. This survey focuses on A–C.

cheaper than full proof-assistant search.

3 Which area of mathematics is "hardest"?

The question "which area of math is the hardest to solve" is underspecified: hardest for humans, for automated provers, or for LLM-based systems? Here we answer it in a way that is actionable for system design: *hardest for current LLM-centric discovery loops*.

3.1 Difficulty dimensions that matter for LLM systems

One useful way to decompose this is:

- **Verifiability:** Is there a cheap, unambiguous checker? (unit tests, constraints, proof assistant)
- **Formalization barrier:** Can the object be represented in code or a formal language without huge overhead?
- **Search topology:** Does improvement require exploring an enormous combinatorial space with sparse rewards?
- **Abstraction depth:** Do solutions require introducing new concepts, definitions, or multi-lemma scaffolding?
- **Data availability:** Is there enough training signal (text, code, formal libraries) aligned with the target domain?

Figure 3 maps common mathematical areas along two dominant axes: verifiability and long-horizon abstraction/search. The takeaway is that what looks "hard" to humans can be comparatively

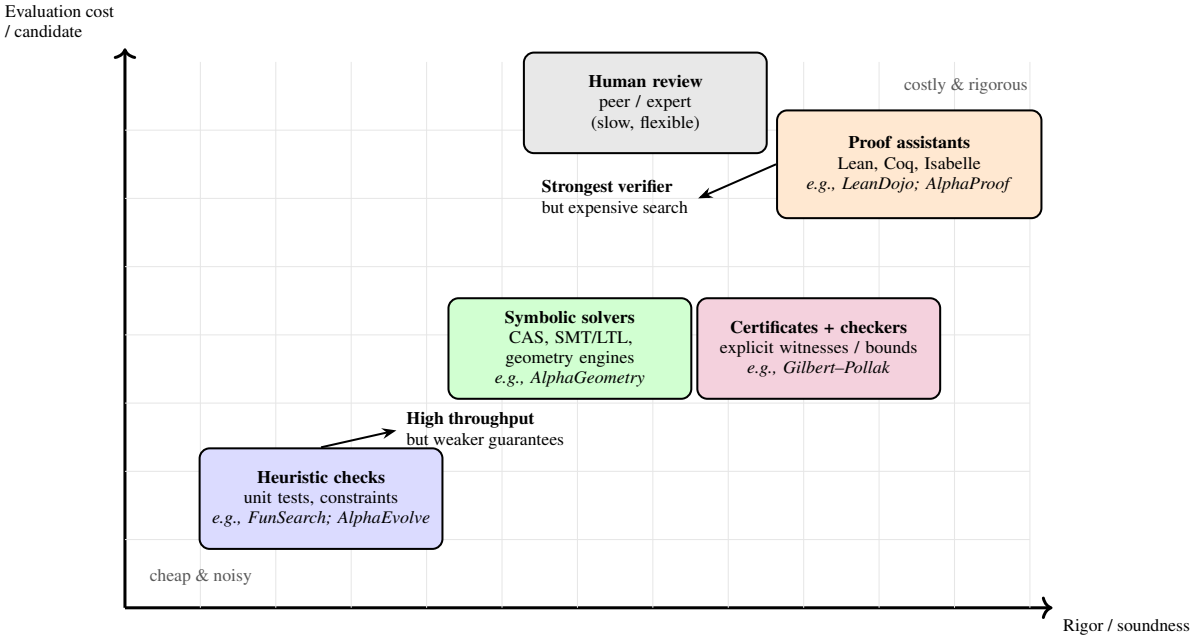


Figure 2: Verifier regimes plotted by approximate evaluation cost and rigor. Different "open-problem" case studies land in different parts of the space; the most scalable successes tend to combine *cheap checking* with *strong guarantees*.

Community signal	**Problem discussed and how we use it**
Erdős wiki / forum / Mathstodon	Problem #728 and related number-theoretic problems; upgrade only when a Lean file, checked write-up, or problem-page status supports the claim.
Knuth page / GitHub repository	Hamiltonian-cycle decomposition of a directed toroidal grid; a public construction plus linked Lean verification is treated differently from a bare claim.
LinkedIn / HN amplification	Useful for noticing fast-moving claims (e.g., Erdős #728, "Claude's Cycles"), but remains Tier D unless it points to a checker, proof, or citable artifact.

Table 2: How non-paper community signals are incorporated without lowering the evidence standard.

accessible to an LLM+verifier loop if the problem admits an objective checker.

3.2 A defensible answer

Under this lens, the most difficult areas for current LLM-based systems are those combining: (i) low verifiability (or extremely expensive checking), (ii) high abstraction depth, and (iii) weak alignment between natural language descriptions and executable representations. This includes large parts of analysis (e.g., PDE regularity), arithmetic geometry, and deep parts of algebraic number theory. By contrast, extremal combinatorics, constructive finite geometry, and some inequality/bounding problems are *comparatively* accessible because they can be posed as "find an object" with a deterministic checker.

4 A timeline of breakthroughs towards verifiable discovery

The emergence of credible "LLM solves open problem" results did not happen at once. It is the product of several strands: transformer scaling (Vaswani et al., 2017; Brown et al., 2020), better prompting and decoding for multi-step reasoning (Wei et al., 2022; Wang et al., 2023), math-specialized LMs (e.g., Minerva) (Lewkowycz et al., 2022), tool-use paradigms (Yao et al., 2023; Schick et al., 2023), and mature verification infrastructure (proof assistants and benchmark tooling).

How to read the timeline. The color of a milestone indicates which part of the closed loop it primarily strengthens: model capability, tool/verifier infrastructure, or the discovery loop itself. For example, chain-of-thought and self-consistency strengthen proposal generation and sampling; ReAct, Toolformer, LeanDojo, and SMT-style backends strengthen the executor/verifier interface; FunSearch, AlphaEvolve, and the Gilbert–Pollak work instantiate the full generate–check–search pattern. This is the link between the timeline, the verifier spectrum, and the framework diagram.

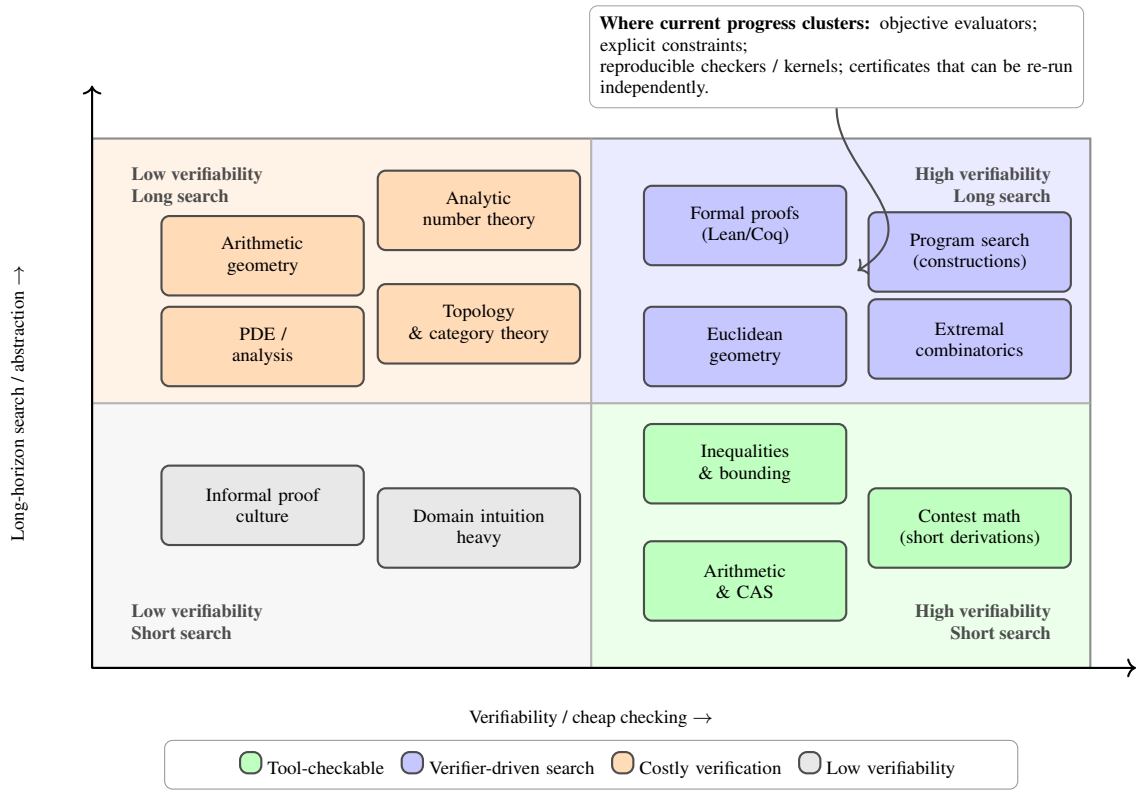


Figure 3: A qualitative hardness map for LLM-based discovery. The regime with *high verifiability* and *long-horizon search* (top-right) is where many credible "open-problem" improvements have appeared so far.

From reasoning to discovery. Chain-of-thought prompting and sampling-based decoding improved multi-step reasoning on math benchmarks (Wei et al., 2022; Wang et al., 2023), and specialized pretraining further improved mathematical competence (Lewkowycz et al., 2022). But benchmark reasoning is still far from open-ended discovery. The shift to discovery required (a) making the target object executable (code, proof assistant, or certificate), and (b) engineering a feedback loop so the model can iterate.

Specialized mathematical LMs. In parallel, several efforts trained or adapted LLMs specifically for mathematics. Minerva demonstrated that large-scale pretraining and math-focused data can substantially improve quantitative reasoning (Lewkowycz et al., 2022). Open models such as Llemma illustrate a complementary direction: releasing weights, training data mixtures, and tooling to enable reproducible research on mathematical LMs (Azerbayev et al., 2023b).

Formal theorem proving accelerates (and supplies stronger verifiers). Recent progress in formal reasoning systems highlights an important point for this survey: proof assistants are not only an evaluation tool but increasingly a *platform* for closed-loop training and search. HyperTree Proof Search introduced a transformer policy coupled with a structured proof search algorithm for neural theorem proving (Lample et al., 2022). Large-scale synthetic-data efforts such as DeepSeek-Prover show how to build formal training corpora in Lean 4 (Xin et al., 2024). AlphaProof demonstrates reinforcement-learning-based formal reasoning at a medal level on IMO problems (paired with AlphaGeometry2) (Hubert et al., 2026). While these systems are not usually presented as "solving open research conjectures," they materially expand the feasibility of Tier A claims.

5 Key techniques and frameworks

Across the literature, LLMs rarely "solve" research problems in a single shot. The most credible progress comes from *closed-loop systems* that turn an LLM into a proposal generator and use an external evaluator to filter, score, and iterate. This section distills the recurring building blocks.

5.1 Core design: generate–check–search

Most successful systems implement some variant of:

> *Generate many candidates; automatically evaluate them; use feedback to*

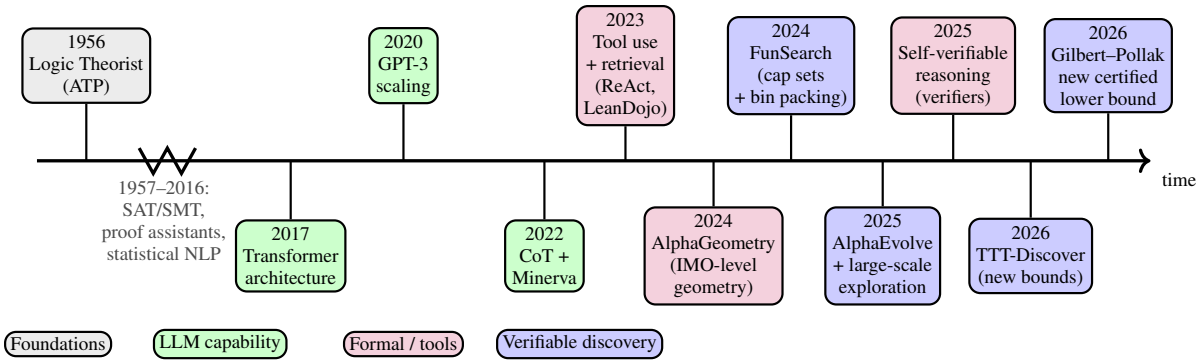


Figure 4: A compressed timeline of milestones leading to verifiable LLM-driven mathematical discovery. "Discovery" systems typically combine an LLM proposal mechanism with automated evaluation loops; Tables 4–5 attach citations and component labels to the milestones.

guide further generation.

At the simplest end, this looks like best-of-N selection with a verifier (Cobbe et al., 2021). At the discovery end, the "verifier" is an executable objective (constraints, tests, certificates, or proof kernels) and the outer loop is an explicit search algorithm.

5.2 Executors and tool use

Tool use makes candidate artifacts *executable*:

- **Program execution / simulators** provide precise feedback (scores, counterexamples) and are central to program-search discovery (e.g., FunSearch) (Romera-Paredes et al., 2024).
- **Symbolic engines** (CAS/SMT/geometry solvers) can act as fast verifiers and structured executors (e.g., AlphaGeometry) (Trinh et al., 2024).
- **Proof assistants** provide the strongest verifiers, enabling fully checkable Tier A claims, but introduce a formalization barrier; recent work increasingly treats them as a platform for learning and search (e.g., LeanDojo; AlphaProof) (Yang et al., 2023; Hubert et al., 2026).

General tool-use paradigms (prompted or learned) such as ReAct and Toolformer help connect LLM reasoning to external actions (Yao et al., 2023; Schick et al., 2023).

5.3 Formal-methods backends: LTL and SMT

Linear Temporal Logic (LTL) is relevant when the candidate object is not a single expression but a trace-producing process: a protocol, strategy, search controller, or transition system. Safety properties ("bad states never occur") and liveness properties ("a desired event eventually occurs") can be stated in LTL; bounded model checking then reduces finite-horizon LTL obligations to SAT/SMT instances and returns counterexamples that are directly useful as feedback to an LLM loop (Pnueli, 1977; Biere et al., 1999).

Satisfiability Modulo Theories (SMT) solvers generalize SAT with theories such as arithmetic, bit-vectors, arrays, and uninterpreted functions, making them a natural middle layer between unit tests and proof assistants (Barrett and Tinelli, 2018; de Moura and Bjørner, 2008). In the framework of Figure 5, an LLM can propose an invariant, lemma schema, constraint encoding, or program fragment; an SMT/LTL backend can then (i) reject it with a model or counterexample, (ii) certify bounded instances, or (iii) simplify the remaining proof obligation before a proof assistant checks the final theorem. This is why SMT/LTL are especially relevant to verifiable discovery: they turn informal mathematical intent into structured, relatively cheap obligations with actionable feedback.

5.4 Search controllers

When naive sampling is insufficient, systems add structure:

- **Evolutionary search** over programs or code edits (FunSearch; AlphaEvolve) (Romera-Paredes et al., 2024; Novikov et al., 2025).
- **Tree search** over structured partial solutions (e.g., proof states in formal math) (Lample et al., 2022).
- **(Test-time) adaptation / RL** when the evaluator provides dense feedback in a specific environment (e.g., AlphaProof; TTT-Discover) (Hubert et al., 2026; Yuksekgonul et al., 2026).

System / paper	Domain	What improved?	Tier	Verifier / artifact
FunSearch (Romera-Paredes et al., 2024)	Constructions	New best-known constructions on checkable problems	C	Program evaluator as fitness; executable code artifacts.
AlphaGeometry (Trinh et al., 2024)	Geometry	IMO-level geometry solving (checkable proofs)	B/C	Symbolic engine + neural guidance; proof traces checkable by solver.
AlphaEvolve (Novikov et al., 2025)	Mixed	Objective-driven rediscovery + improvements	C	Task-specific evaluators; tracked code edits and scores.
Gilbert–Pollak bound (Ke et al., 2026)	Optimization / geom.	New *certified* lower bound (Steiner ratio)	B	Certificate-checking pipeline; independently verifiable bound.
Erdős Problem #728 (Sothanaphan, 2026)	Number theory	Formalized resolution of a factorial-divisibility problem	A	Lean proof attributed to GPT-5.2 Pro + Aristotle; informal write-up and community tracking.
Claude's Cycles (Knuth, 2026)	Combinatorics	Hamiltonian-cycle decomposition for a directed toroidal grid	A/B	Public construction plus linked Lean verification and follow-up variants.

Table 3: Representative case studies illustrating how LLMs (in systems) can yield verifiable progress.

5.5 Retrieval over mathematical libraries

A repeated bottleneck—especially in formal settings—is *premise selection*: retrieving relevant lemmas and definitions. Open infrastructure such as ProofNet (informal↔formal pairs) and LeanDojo (interaction + corpora + benchmarks) makes retrieval and proof-search research more reproducible (Azerbayev et al., 2023a; Yang et al., 2023).

5.6 A reference architecture

Figure 5 summarizes a reusable "discovery loop." Different projects vary in their executors and verifiers, but the overall control flow is stable.

6 Representative case studies

Table 3 lists representative exemplars that reach evidence tiers A–C. The common thread is that the "result" is an independently checkable artifact: code, a certificate, or a formal proof object. For the Gilbert–Pollak entry, we distinguish the classical Steiner-minimal-tree conjecture from the newer verifier-backed lower-bound claim (Gilbert and Pollak, 1968; Ke et al., 2026).

7 Best-practices checklist for verifiable discovery claims

To make "open-problem" claims comparable and reproducible, we recommend reporting:

- **What was open:** precise statement of the target (problem family, n-range, constraints) and the prior best-known baseline.
- **Verifier/evaluator:** complete spec plus code (or proof kernel) needed to check candidates; tests against known instances.
- **Artifacts:** best candidate programs/proofs/certificates and scripts to reproduce the reported numbers.
- **Compute + search budget:** number of evaluations, LLM calls, wall-clock, and search hyperparameters.
- **Robustness + novelty:** reruns with different seeds; adversarial tests; leakage/overlap checks when relevant.
- **Human-in-the-loop disclosure:** manual steps required to reach the final artifact.
- **Community-claim trail:** for social-media or problem-wiki announcements, include the canonical problem page, transcript/checker link, and current status (full/partial/incorrect/pending).

8 Open challenges

Even in the verifiable regime, major challenges remain:

- **Beyond cheap verifiers:** most of mathematics lacks a fast checker; progress may require new in-

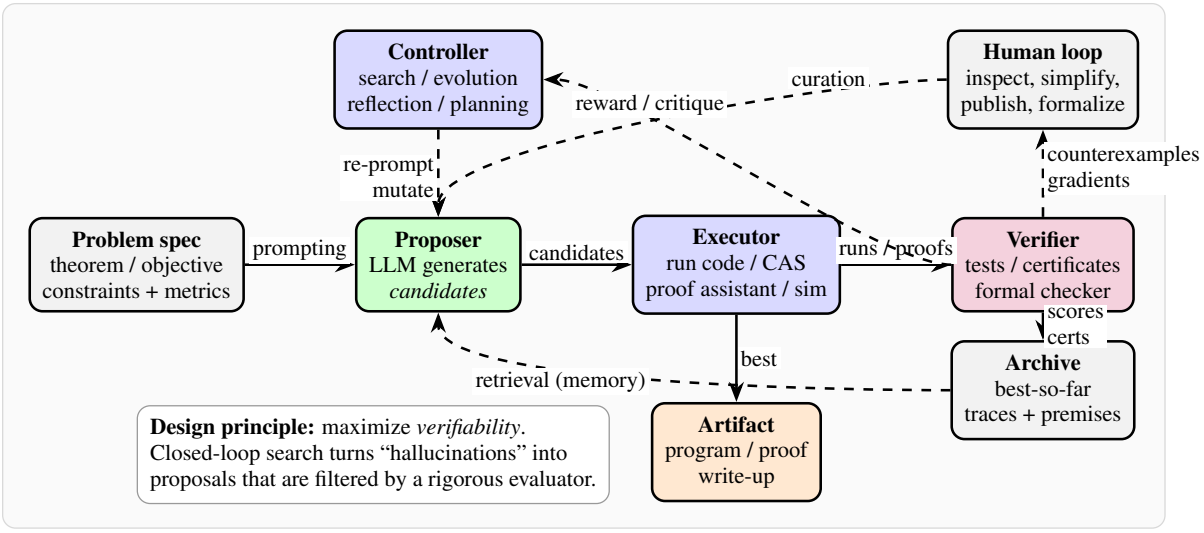


Figure 5: A generic closed-loop framework for verifiable LLM-driven discovery. The strongest "open problem" results appear when the verifier is strong *and* cheap, and when the archive (memory) enables rapid iteration.

termediate representations or hybrid human+AI workflows.

- **From finite-n to theorems:** turning best-known constructions/bounds into general statements (and proofs) is still hard.
- **Novelty and significance:** correctness is not the same as importance; we need better novelty tracking and human-facing inspection tools.
- **Community norms:** clearer standards for attribution, failure reporting, and maintaining open repositories of problems with evaluators.

9 Conclusion

Current systems remain limited, particularly outside domains with strong evaluators. Where strong evaluators exist, however, they can be powerful engines for exploring large search spaces and producing independently checkable improvements. Taken together, these case studies suggest that progress is driven less by "better prose reasoning" and more by *systems design*: executable representations, strong verifiers, retrieval over libraries, and structured search.

Limitations

This survey is a snapshot of a fast-moving area and should not be read as an exhaustive catalog of every community-reported claim. We deliberately emphasize cases with public artifacts, checkers, certificates, or formal proofs, which means that less-verifiable but potentially important mathematical assistance is underrepresented. The evidence ladder also compresses many distinctions: a formally checked result can still depend on problem formalization choices, and a reproducible computational result can still be sensitive to implementation details or search budgets. Finally, several recent examples are tracked through community pages, technical notes, and preprints whose status may evolve.

References

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

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023b. Llemma: An open language model for mathematics. *CoRR*, abs/2310.10631.

Clark Barrett and Cesare Tinelli. 2018. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 305–343. Springer.

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 1999. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer.

Joseph Boland. 2026. ChatGPT 5.2 solves Erdos #728 with Terence Tao. LinkedIn post. Accessed May 12, 2026.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer.

Jordan S. Ellenberg, Cristofero S. Fraser-Taliente, Thomas R. Harvey, Karan Srivastava, and Andrew V. Sutherland. 2025. Generative modeling for mathematical discovery. *CoRR*, abs/2503.11061.

Erdős Problems community. 2026. 728 discussion thread. Erdős Problems forum thread. Accessed May 12, 2026.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. 2025. Mathematical exploration and discovery at scale. *CoRR*, abs/2511.02864.

Edgar N. Gilbert and Henry O. Pollak. 1968. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29.

Hacker News contributors. 2026. Erdős problem #728 was solved more or less autonomously by GPT-5.2 Pro. Hacker News discussion thread. Accessed May 12, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874.

Torsten Hoefler. 2026. Knuth's AI breakthrough: Claude Opus 4.6 solved open math problem. LinkedIn post. Accessed May 12, 2026.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, et al. 2026. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613.

Yisi Ke, Tianyu Huang, Yankai Shu, Di He, Jingchu Gai, and Liwei Wang. 2026. Towards solving the Gilbert–Pollak conjecture via large language models. *CoRR*, abs/2601.22365.

Donald E. Knuth. 2026. Claude's Cycles. Stanford Computer Science technical note. Revised Apr. 14, 2026; accessed May 12, 2026.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. 2022. HyperTree Proof Search for neural theorem proving. *CoRR*, abs/2205.11491.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *CoRR*, abs/2206.14858.

Kim Morrison and contributors. 2026. KnuthClaudeLean: Lean 4 formalization of Knuth's "Claude's Cycles". GitHub repository. Accessed May 12, 2026.

Alexander Novikov, Ngan Vu, 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. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *CoRR*, abs/2506.13131.

OpenAI. 2025. Advancing science and math with GPT-5.2. OpenAI blog. Accessed May 12, 2026.

Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57.

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. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Nat Sothanaphan. 2026. Resolution of Erdős problem #728: A writeup of Aristotle's Lean proof. *CoRR*, abs/2601.07421.

Terence Tao. 2026. Recently, the application of AI tools to Erdős problems passed a milestone. Mathstodon post. Accessed May 12, 2026.

Terence Tao and contributors. 2026. AI contributions to Erdős problems. GitHub wiki. Edited May 10, 2026; accessed May 12, 2026.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale synthetic data. *CoRR*, abs/2405.14333.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. 2026. Learning to discover at test time. *CoRR*, abs/2601.16175.

A Supplementary timeline and explanatory material

The main paper provides a compressed timeline (Figure 4). For completeness, this appendix collects supplementary material in one place: (i) a multi-track visualization of milestones (Figure 6), (ii) a detailed timetable split across two eras (Tables 4 and 5), and (iii) an explanatory diagram summarizing common failure modes in discovery loops and practical mitigations (Figure 7).

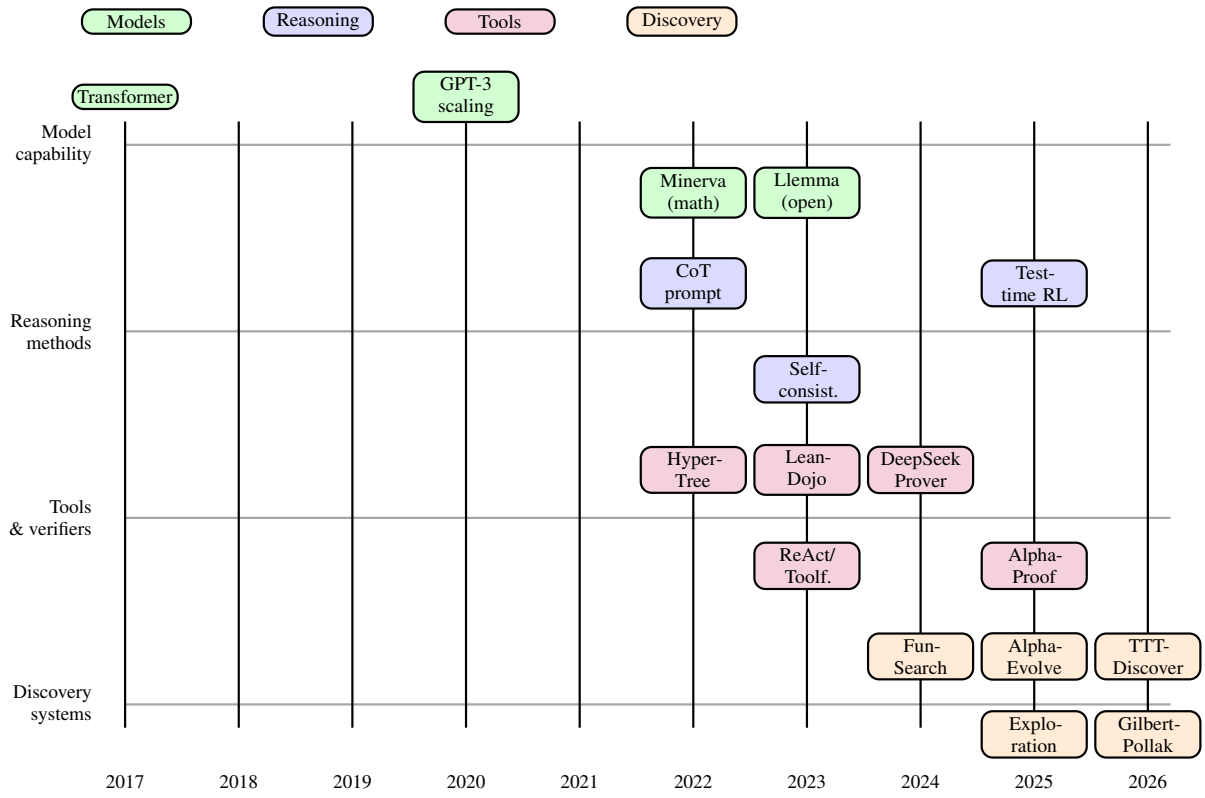


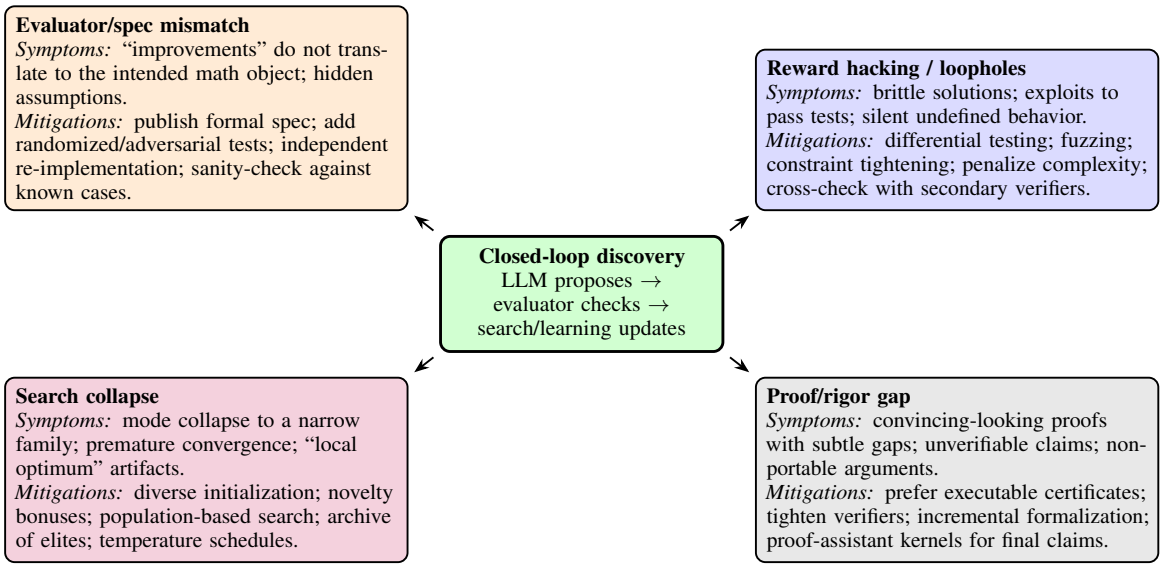
Figure 6: A multi-track timetable of milestones. Each track emphasizes a different ingredient needed for verifiable discovery: model capability, reasoning methods, tool/verifier integration, and discovery systems.

Year	Milestone	Component	Relevance to verifiable discovery
1956	Logic Theorist (ATP)	Foundations	Early template: symbolic search + correctness rules; foreshadows "generate + check" workflows.
1977	Linear Temporal Logic (Pnueli, 1977)	Formal methods	Specifies trace properties such as safety and liveness, which matter when LLM-generated artifacts are protocols or transition systems.
1999	Bounded model checking (Biere et al., 1999)	Verification	Reduces bounded LTL checking to SAT, producing counterexamples that can guide search.
2008/2018	SMT solvers (de Moura and Bjørner, 2008; Barrett and Tinelli, 2018)	Verification	Solvers over arithmetic, bit-vectors, arrays, and uninterpreted functions provide a cheap middle layer between tests and proof assistants.
2017	Transformer (Vaswani et al., 2017)	Models	Scaling-friendly architecture underpinning modern LLMs and in-context reasoning.
2020	GPT-3 scaling (Brown et al., 2020)	Models	Demonstrates few-shot learning and broad competence that later supports mathematical coding/reasoning.
2021	MATH dataset (Hendrycks et al., 2021)	Benchmarks	High-signal benchmark for multi-step symbolic reasoning; drives solver and verifier training.
2021	Training verifiers (Cobbe et al., 2021)	Verification	Establishes verifier-style scaling: sample many candidates, filter with learned/engineered verifiers.
2022	Chain-of-thought prompting (Wei et al., 2022)	Reasoning	Improves long-horizon reasoning via intermediate steps; foundation for later self-critique / reflection loops.
2022	Self-consistency (Wang et al., 2023)	Decoding	Sampling-based aggregation turns stochastic generation into a selection problem (proto-verifier scaling).
2022	Minerva (Lewkowycz et al., 2022)	Math LMs	Math-focused training increases quantitative competence, a key ingredient for program-level discovery.
2022	HyperTree Proof Search (Lample et al., 2022)	Formal search	Structured proof-state search coupled with learning, bridging theorem proving and modern ML.

Table 4: Timetable (foundations): milestones that enabled LLM-based math discovery and verifiable reasoning.

Year	Milestone	Component	Relevance to verifiable discovery
2023	ReAct (Yao et al., 2023)	Tool use	Couples reasoning with actions (tool calls), enabling executable feedback loops.
2023	Toolformer (Schick et al., 2023)	Tool learning	Shows how tool usage can be learned, not only prompted; supports scalable executor integration.
2023	ProofNet (Azerbayev et al., 2023a)	Autoformalization	Paired informal/formal data for evaluating statement translation and proof generation.
2023	LeanDojo (Yang et al., 2023)	Infrastructure	Standardizes interaction with Lean and retrieval; makes premise selection + proof search reproducible.
2023/2024	FunSearch (Romera-Paredes et al., 2024)	Discovery	LLM-guided evolutionary program search surpasses best-known constructions on checkable problems.
2024	AlphaGeometry (Trinh et al., 2024)	Neuro-symbolic	Strong symbolic engine paired with learned guidance solves hard geometry reliably.
2024	DeepSeek-Prover (Xin et al., 2024)	Formal data	Large-scale Lean 4 proof data, improving feasibility of RL/search in formal environments.
2025	AlphaProof (Hubert et al., 2026)	Formal RL	Medal-level formal reasoning; highlights strict verifiers + RL in formal math.
2025	AlphaEvolve (Novikov et al., 2025)	Discovery	Evolves codebases with evaluators; applies beyond math to algorithm discovery and optimization.
2025	Mathematical exploration at scale (Georgiev et al., 2025)	Discovery	Validates exploration at scale (successes + failures) to support scientific norms.
2026	TTT-Discover (Yuksekgonul et al., 2026)	Test-time RL	Learns at test time to optimize one environment; complements evolutionary approaches.
2026	Gilbert–Pollak lower bound (Ke et al., 2026)	Certified progress	Certificate-checking pipeline pushes a certified bound on a long-stagnant problem.
2026	Erdős Problem #728 (Sothanaphan, 2026; Terence Tao and contributors, 2026)	Formal/community	Illustrates the path from community report to Lean-formalized artifact and curated status tracking.
2026	Claude's Cycles (Knuth, 2026)	Formal/community	Public problem note, construction, and linked Lean verification for a Hamiltonian-cycle decomposition problem.

Table 5: Timetable (verifiable discovery era): milestones where LLMs are embedded in evaluator- and certificate-driven loops.



Survey takeaway: scalable discovery requires both *good feedback signals* and *defenses* against optimization pathologies.

Figure 7: Common failure modes in LLM-based discovery loops and practical mitigations.