A CLOSED-LOOP SYSTEM OF 3 MATHEMATICAL AGENTS: CONJECTURER, REFUTER, AND PROVER

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a closed-loop autonomous system for mathematical discovery. In this paper, "autonomous" refers to execution-level autonomy: after initialization, one command runs conjecture generation, counterexample search, formal proving, outcome reconciliation, and the next generation without human intervention between stages. The system generates conjectures in the form of linear inequalities, searches for counterexamples, and sends surviving statements to a Lean 4 prover with a zero-`sorry` policy. The concrete setting is polyhedral combinatorics. The system separates proposal, verification, and learning. Outcome-dependent signals may influence the conjecture generator, but do not modify the counterexample validator, exhaustive search procedure, or proof-acceptance gates. Accepted counterexamples contain explicit witness graphs and pass five validation checks; accepted proofs are compiling Lean artifacts whose root theorems match the generated statements. The system also separates arithmetic feasibility from geometric realizability. In the current repository version, the campaign records 204 witness-verified refutations, four end-to-end machine-proved theorems, and three open conjectures. The main contribution is an artifact-producing autonomous workflow that preserves refutations, proofs, and unresolved cases as reusable computational or formal objects.

1 INTRODUCTION

Automated mathematical reasoning is moving from isolated tools toward integrated discovery systems. Earlier systems such as Graffiti generated mathematical conjectures automatically Fajtlowicz (1988), while more recent work combines data-driven conjecturing, search, and formal proving Davila (2024); Kasaura et al. (2025); Onda et al. (2025); Dong & Ma (2025). However, these components do not automatically form a closed discovery process. We ask whether one system can generate conjectures, search for counterexamples, formally prove surviving conjectures, and use the results to generate its next conjectures without human intervention during execution.

We study this question using affine inequalities over the face-size histograms of simple convex 3-polytopes Grünbaum (2003). For such a polytope, p_i denotes the number of i-gonal faces, and

$$p = (p_3, p_4, p_5, \ldots)$$

is called its p-vector Barnette (1969). Thus, each coordinate has a direct geometric meaning: p_3 counts triangular faces, p_4 counts quadrilateral faces, and so on. Candidate conjectures can consequently be written as affine inequalities in interpretable variables p_i.

This representation supports every stage of our system. Euler-type identities provide arithmetic filters, while Steinitz's theorem Steinitz (1922) connects convex 3-polytopes with planar 3-connected graphs; for simple polytopes, these graphs are cubic. This connection enables explicit witness construction and graph enumeration Brinkmann & McKay (2007). The resulting inequalities are also natural targets for formal verification in Lean.

The domain exposes a central difficulty: arithmetic feasibility is weaker than geometric realizability. Every simple convex 3-polytope satisfies

$$\sum_{k\geq 3}(6-k)p_k = 12,$$

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.

System	Generates statements	Counterexample channel	Exhaustive search	Formal proof	Qualified terminal artifacts
Graffiti / TxGraffiti Fajtlowicz (1988); Davila (2024)	✓	—	—	—	—
FunSearch Romera-Paredes et al. (2024)	✓	—	—	—	—
Conjecturing-Proving Loop Kasaura et al. (2025)	✓	—	—	✓	—
LeanConjecturer / STP Onda et al. (2025); Dong & Ma (2025)	✓	—	—	✓	—
AutoGraphForge Pastorek (2026)	✓	✓	✓	✓	—
Learning to Disprove Li et al. (2026)	—	✓	—	✓	—
Rethlas+Archon Ju et al. (2026)	—	✓	—	✓	—
Ours	✓	✓	✓	✓	✓

Table 1: Comparison with representative systems for automated conjecturing, counterexample generation, formal proving, and autonomous discovery. "Qualified terminal artifacts" are independently inspectable witness graphs, compiled Lean proofs, or documented open cases, rather than merely generated or surviving candidates.

which is a necessary arithmetic condition on its p-vector Barnette (1969). It is not sufficient, however, to guarantee that a nonnegative integer sequence is the face-size histogram of an actual polytope. Therefore, a candidate counterexample must be checked against realizable planar 3-connected cubic graphs, rather than validated only arithmetically.

Our system implements the closed loop

$$\text{conjecture} \longrightarrow \text{refute} \longrightarrow \text{validate} \longrightarrow \text{prove} \longrightarrow \text{learn}.$$

It generates affine inequalities, searches for realizable counterexamples, formally verifies surviving conjectures in Lean, and converts refutation, proof, and prover-failure outcomes into signals for the next generation.

Contributions.

1. **A closed-loop autonomous discovery system.** We implement an unattended generate–refute–prove–learn cycle in which the system routes candidates between stages and constructs the next generation from previous outcomes.
2. **A realizability-aware counterexample pipeline.** Candidates are attacked using permanent witness replay, plantri-pool replay, Eberhard-lattice search, bounded exhaustive plantri screening, and parallel stochastic tracks.
3. **Defense-in-depth counterexample validation.** Every accepted refutation contains an explicit witness graph and passes arithmetic, hypothesis, violation, realizability, and independent re-verification checks.
4. **Entailment-gated formalization.** Candidates are not treated as true merely because no counterexample was found. They first pass an inventory-entailment pre-check and are accepted as proved only after complete Lean compilation and a zero-`sorry` audit.
5. **Outcome-driven learning without verification drift.** Refuted, proved, surviving, and prover-stuck conjectures produce different signals for mutation, sharpening, weakening, dynamic-cell selection, and support-function mining, while the verification gates remain fixed.
6. **A completed artifact-producing campaign.** The current repository contains 204 witness-verified refutations, four complete Lean proofs, and three qualified open conjectures.

The claim is architectural and empirical rather than a claim of discovering a deep new theorem in polytope combinatorics: a mathematical system can operate unattended across conjecture generation, counterexample search, formal proof, and learning while retaining independently inspectable evidence. Table 1 situates the system against representative prior and concurrent work.

2 Related Work

Automated mathematics. Efforts to automate mathematics span the full arc from proposing statements to proving them, not any single step. Classical systems such as GRAFFITI generated graph-theoretic inequalities from invariant tables Fajtlowicz (1988), and TxGraffiti extended this data-driven paradigm Davila (2024). Our generator follows this tradition but does not stop at proposal:

each accepted statement is automatically routed to counterexample search, formal entailment checking, or Lean proving. Recent work also treats conjecturing as distinct from autoformalization and proof generation Sivakumar et al. (2025). More broadly, researchers are already thinking of automatizing all steps of mathematics Hales (2014); Avigad (2019), and this is what we do in this paper.

Programmatic mathematical discovery. FunSearch uses language models to propose programs evaluated by an external procedure Romera-Paredes et al. (2024); related work applies this paradigm to combinatorial and number-theoretic discovery Ellenberg et al. (2025). Our system instead generates mathematical statements, and its evaluator must preserve logical status: a candidate may be refuted by a witness, eliminated by bounded exhaustion, proved formally, or remain open.

Formal proving and conjecture–proof loops. Formal theorem-proving systems generate proofs for supplied statements AlphaProof and AlphaGeometry teams, Google DeepMind (2024). Other systems close generate–prove loops within formal libraries, including the Conjecturing-Proving Loop, LeanConjecturer, and STP Kasaura et al. (2025); Onda et al. (2025); Dong & Ma (2025), and Enumerate–Conjecture–Prove frameworks combine enumeration, pattern extraction, and formal verification Sun et al. (2025). Our system differs because its statements refer to natural objects represented externally by face vectors and witness graphs; arithmetic feasibility does not imply geometric realizability, so formal proof is preceded by an explicit realizability-aware refutation channel.

Concurrent closed-loop graph discovery. AutoGraphForge Pastorek (2026) is closely related in structure, combining Graffiti3-based conjecture generation, graph-based counterexample search, formal statement generation, and Lean verification. We treat it as an important concurrent point of comparison. Our scope differs in two respects: we study simple 3-polytopes, where arithmetic feasibility and graph realizability form an explicit verification boundary; and we report a completed terminal campaign with witness-verified refutations, qualified end-to-end proofs, and explicit open statuses. The distinction is thus based on reported terminal artifacts and domain-specific verification, not on the mere existence of a generate–refute–prove architecture.

Counterexamples and polytope combinatorics. Reinforcement-learning and Monte-Carlo methods have been used to construct graph-theoretic counterexamples Wagner (2021); Vito & Stefanus (2023), and recent work studies formal counterexample generation in Lean 4 Li et al. (2026). Our counterexamples are explicit planar cubic witness graphs that are independently revalidated, stored permanently, and replayed against future conjectures; our use of plantri builds on exhaustive planar graph generation Brinkmann & McKay (2007). The mathematical setting is grounded in classical results on simple 3-polytopes, including Steinitz's characterization, Eberhard-type results, Grünbaum's examples, Jučovič's inequality, and Barnette-type bounds Steinitz (1922); Eberhard (1891); Grünbaum (2003); Jučovič (1971); Barnette (1969).

3 Mathematical Setting

A *simple convex 3-polytope* is a bounded convex polyhedron in $\mathbb{R}^3$ in which every vertex is incident to exactly three edges. Its edge graph is planar, 3-connected, and 3-regular, and by Steinitz's theorem this graph class coincides exactly with the edge graphs of convex 3-polytopes. Let p_k denote the number of k-gonal faces and define the total face count

$$f_2 = \sum_{k\geq 3} p_k.$$

Euler's formula $V - E + f_2 = 2$ together with 3-regularity ($2E = 3V$ and $2E = \sum_{k\geq 3} k\, p_k$) yields the linear relation

$$\sum_{k\geq 3} (6-k)\, p_k = 12, \tag{1}$$

which we call the *Eberhard equation* Grünbaum (2003), and consequently

$$V = 2(f_2 - 2), \qquad E = 3(f_2 - 2).$$

We call a nonnegative integer vector $p = (p_3, p_4, p_5, \dots)$ of finite support a *p-vector*, and we say p is *arithmetically feasible* if it satisfies the Eberhard equation (1). The set of arithmetically feasible p-vectors within a hypothesis region is the *Eberhard lattice* searched in Section 4.3.

The central subtlety, and the reason the system separates cheap screening from witnessing, is that (1) is necessary but *not* sufficient. We make the target property precise.

Definition 1 (Realizability)**.** A p-vector p is *realizable* if there exists a simple convex 3-polytope P whose number of k-gonal faces equals p_k for every $k \geq 3$. Equivalently, by Steinitz's theorem, p is realizable if and only if there exists a planar, 3-connected, 3-regular graph whose face-size multiset equals the multiset encoded by p. A p-vector that is arithmetically feasible but not realizable is called *non-realizable*.

Since p_6 has coefficient zero in (1), arithmetic feasibility does not constrain it; by Eberhard's theorem Eberhard (1891); Grünbaum (2003), every $(p_k)_{k\neq 6}$ satisfying (1) is realizable for some but not every p_6. This is why our conjectures take p_6 as the conclusion variable.

Realizability is strictly stronger than arithmetic feasibility: the Eberhard equation is a single linear constraint on the p_k, whereas Definition 1 demands an actual combinatorial polytope. Eberhard's theorem and its refinements show that the gap is genuine: there exist arithmetically feasible p-vectors that no polytope realizes. This distinction drives the two-tier design of the system. *Arithmetic feasibility* (testing (1)) is a cheap necessary screen, while *realizability*, certified by explicitly constructing a planar cubic witness graph, is the sound criterion. Consequently, a p-vector that violates a candidate conjecture but is found only in the Eberhard lattice is treated as a mere *search target*: it becomes a genuine counterexample only after a corresponding witness graph is constructed and independently verified to be realizable in the sense of Definition 1.

The conjecture language consists of statements

$$\textbf{if } h_1 \wedge \cdots \wedge h_r \textbf{ then } p_6 \ \gtrless \ c_0 + \sum_i c_i x_i,$$

where the hypotheses h_j include bounds on low-degree face counts and on f_2. The variables x_i are selected from p_3, p_4, p_5, p_6 and the high-order face count $S = \sum_{k\geq 7} p_k$, with rational coefficients of bounded denominator. This language supports exact arithmetic, graph-based refutation, and Lean formalization.

4 System Overview

After initialization, a single project command executes one generation of the autonomous loop

$$\text{generate} \to \text{refute} \to \text{validate} \to \text{prove} \to \text{reconcile} \to \text{learn} \to \text{generate}.$$

No human selects, routes, validates, formalizes, or writes hints for individual conjectures during execution. The repository's autonomous mode uses the data flow of Figure 1.

4.1 Autonomous State and Stage Interfaces

The loop is a sequence of typed interfaces rather than direct model-to-model communication. The generator emits a registered conjecture (canonical statement, parsed hypotheses, conclusion coefficients, identifier, and metadata); the counterexample pipeline consumes it and returns a terminal search record. That record distinguishes four outcomes with distinct logical force: a validated refutation (the statement is false), a bounded exhaustive non-realizability result (decisive only within its declared finite regime, per Definition 1), an unsuccessful stochastic search (no truth certificate), and an infrastructure or parsing failure (not a mathematical outcome). Surviving candidates pass to the inventory-entailment pre-check, which returns either an arithmetic countermodel or permission to invoke the prover; the prover in turn returns a qualified complete proof, a partial or failed formalization, or an infrastructure failure, and only the first earns proved status. Keeping these labels typed prevents "not refuted," "not proved," "prover failed," and "false" from collapsing into one verdict. The registry preserves this distinction and reuses it when deriving the next generation's proposal signals.

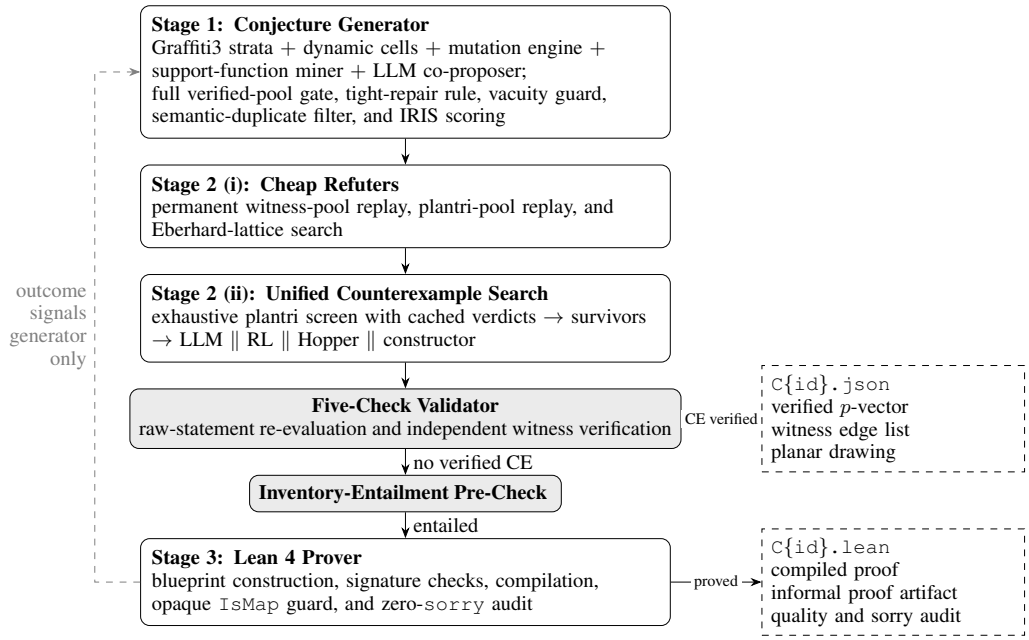


Figure 1: One generation of the autonomous discovery loop. Dashed arrows represent automatic outcome reconciliation and signal derivation. No human routes individual conjectures between stages or writes hints for the next generation.

4.2 Stage 1: Conjecture Generation

The `ConjectureGenerator` maintains a discovery table of realized p-vectors and merges five proposal sources: Graffiti3 invariants over the table, dynamic cells targeting uncovered hypothesis-atom combinations, a mutation engine, a support-function miner over affine directions, and an LLM co-proposer. Deterministic gates then filter the pool. The generator is outcome-aware without a manual hint store: signals derived from the registry, verified counterexamples, proof artifacts, and evolution records let refuted conjectures trigger repairs, proved or surviving ones trigger sharpening or controlled weakening, and prover-stuck ones reopen nearby regions.

All filtering uses exact rational arithmetic. Candidates are rejected when malformed, vacuous, inconsistent, or semantically duplicate, where the duplicate filter compares bounded attack surfaces rather than formula syntax. The full-pool gate then checks each survivor against the discovery table, harvested plantri vectors, and validated witnesses, so a candidate valid only on a small sample cannot reach an expensive search stage merely to be refuted by a known witness.

4.3 Stage 2: Decidability-Stratified Counterexample Search

Each conjecture is attacked in increasing order of cost, and the tiers are ordered by logical strength as well.

Permanent witness-pool replay. Validated counterexamples are stored with their p-vectors, edge lists, metadata, and drawings; a cached witness is accepted only after replaying the full validator, turning past search effort into a reusable cross-conjecture resource.

Eberhard-lattice search. An Eberhard-preserving random walk over the integer lattice of (1) prioritizes large violation margins. As in Section 3, these arithmetic violations are only search targets, never counterexamples without a realizable witness.

Exhaustive plantri screen. Within configured bounds the system enumerates arithmetic violation candidates and tests realizability by plantri graph generation: a realizable candidate yields an explicit

planar cubic witness, a non-realizable one is eliminated by exhaustion within those bounds, and both verdicts are cached.

Parallel stochastic tracks. Survivors are passed to an LLM constructor, an RL graph-construction policy, a dual-space Hopper search, and a stochastic constructor with an independent double-check. A validated witness stops the search and cancels the remaining tracks; if all budgets are exhausted the conjecture stays unrefuted and proceeds to the formal stage.

Thus witness replay is constructive, Eberhard-lattice search is cheap but silent on realizability, plantri is two-sided within its range, and the stochastic tracks extend coverage one-sidedly beyond it. The terminal record preserves these distinctions.

4.4 The Five-Check Validator

Every proposed counterexample, cached witnesses included, must pass all five checks below. The first four catch malformed arithmetic, wrong hypotheses, false violations, and unrealizable graphs. The fifth guards against a single faulty parser or graph implementation, which matters because such a bug could otherwise propagate across generations.

1. **Arithmetic consistency:** the p-vector is nonnegative, satisfies (1), and yields consistent V, E.
2. **Hypothesis satisfaction:** every hypothesis of the conjecture is confirmed.
3. **Genuine violation:** the conclusion is evaluated independently of the search heuristic and the margin computed.
4. **Realizability:** the witness is planar, cubic, simple, and 3-connected where required (Definition 1), and its embedding rederives the p-vector.
5. **Independent re-verification:** a separate evaluator reparses the raw conjecture and revalidates the witness without the primary hypothesis splitter or graph routine.

Validated witnesses are exported as permanent artifacts recording the original statement, p-vector, violation margin, edge list, and planar drawing.

4.5 Stage 3: Entailment-Gated Lean Formalization

Candidates without a validated counterexample first face an inventory-entailment pre-check: the orchestrator seeks an arithmetic countermodel that satisfies the inventory assumptions yet violates the conjecture, which signals an insufficient inventory rather than a geometric counterexample. Survivors go to the Lean 4 prover, which locks the root theorem signature, builds a helper-lemma dependency graph, checks signatures, compiles, and enforces a zero-`sorry` policy. A proof is accepted only when the complete file compiles with the required root theorem and no active `sorry` or failed-node marker. A sorry-free file of helper lemmas alone, or a model's report of success, does not qualify.

The geometric inventory uses an opaque `IsMap` predicate, so map-specific lemmas require an explicit map hypothesis and no fabricated structure can satisfy the geometric assumptions. The formal results remain relative to the hand-audited inventory of Section 6 and Appendix A.

4.6 Autonomous Learning and Outcome Reconciliation

Every terminal outcome updates the persistent registry: refutations are stored with witnesses and margins, proofs with Lean artifacts and dependencies, survivors with exhausted search paths, and prover-stuck cases with formalization status and any arithmetic countermodel. Before the next generation these are reconciled automatically, each outcome type producing its own signal for repair, sharpening, weakening, exploration, or dynamic-cell selection, so the generator adapts while the validator, exhaustive search, and proof gates keep fixed standards. Crucially, learning alters proposal priorities, not mathematical standards: it cannot relax the validator, widen a bounded exhaustive verdict, or supply a missing lemma, which prevents the loop from exploiting its own evaluator through vacuous bounds or restated refuted conjectures.

Outcome	Count
Witness-verified refutations	204
End-to-end Lean proofs	4
Open conjectures	3

Table 2: Current repository-level campaign status.

Source	Refutations	Role
Plantri exhaustive screen	57	Original discovery
LLM constructor	23	Original discovery
RL construction policy	17	Original discovery
Eberhard-lattice walk (realized)	3	Original discovery
Hopper dual-space search	1	Original discovery
Cross-conjecture witness replay	93	Reuse
Plantri-pool replay	10	Reuse
Total	**204**	

Table 3: Provenance of the 204 witness-verified refutations. Cross-conjecture replay accounts for just over half (103 of 204), empirically supporting the claim that the witness pool strengthens the campaign over time. No single active track dominates: exhaustive screening, learned construction, and stochastic search all contribute.

5 Campaign Results

The current repository version records 204 witness-verified refutations, four end-to-end Lean proofs, and three open conjectures (Table 2). Registry identifiers, completion records, and artifact details for every proved and open statement are collected in Appendix B.

The refutation count includes only records that carry a fully validated witness graph and pass all five checks; refuted-conjecture entries without a realized witness graph, rejected parses, semantic duplicates, retries, and intermediate registry entries are excluded. A refuted conjecture is thus associated with a realizable witness graph, not merely a model judgment; a proved conjecture with a complete compiling Lean artifact, not a successful partial attempt; and an open conjecture records that the configured search did not produce a validated witness and that the current inventory did not entail the conclusion.

5.1 Refutation Provenance

Of the 204 witness-verified refutations, 101 were first discovered by an active search track and 103 were obtained by replaying a previously validated witness against a later conjecture. Every refutation, including replays, passes the full five-check validator. Table 3 gives the provenance breakdown.

We report provenance rather than a raw-search performance baseline because the search tracks do not share a common output space. In particular, the Eberhard-lattice walk operates only on p-vectors and cannot by itself certify a counterexample; its three entries above were realized into graph witnesses downstream and validated like any other refutation. A performance comparison would require all compared methods to produce candidate graphs under matched budgets, which is a separate controlled study.

5.2 Machine-Proved Conjectures

Each theorem below was proposed by the generator, survived the full counterexample pipeline and the entailment pre-check, and was proved in Lean with no new `sorry`, without human intervention.

A lower bound on hexagons without quadrilaterals. The first statement proved end to end by the pipeline is the following.

Theorem 1. *Let M be a simple 3-connected planar map with $p_4 = 0$, $p_5 \leq 2$, and $f_2 \geq 7$. Define $S = \sum_{k \geq 7} p_k$. Then*

$$2p_6 \geq 4 - S.$$

The proof combines an Euler-type identity, a Jučovič-style inequality, and a Barnette-type bound.

Two bounds in the triangle-plus-large-face regime. When both quadrilaterals and pentagons are absent, the pipeline proved two lower bounds on p_6 under different side conditions.

Theorem 2. *If $p_4 = p_5 = 0$ and $S \geq 1$, then $p_6 \geq 2 - S$.*

Theorem 3. *If $p_4 = p_5 = 0$ and $f_2 \geq 7$, then $3p_6 + 2S \geq 12$, equivalently $p_6 \geq 4 - \frac{2}{3}S$.*

Theorem 2 is informative about the proof agent itself. During autonomous proof search, the agent found that an existing gated Jučovič component of the inventory was stronger than initially expected and used it to close the argument. The result was recorded as a proved theorem and made available to later generations through derived proof signals, so the prover is not only a final certificate generator; its outcomes also modify the future conjecture distribution. Theorem 3 is proved through a chain of seven lemmas. Because their hypotheses differ, Theorems 2 and 3 give individually non-comparable lower bounds on p_6 over the same regime, so the two proofs jointly populate that region of the hypothesis space rather than one subsuming the other.

The fourth, $p_6 \leq f_2 - p_4 - p_5$, holds without further hypotheses and follows directly from $f_2 = \sum_k p_k$; it serves mainly as an end-to-end check of the formal pipeline (Appendix B).

5.3 Open Conjectures and What the Campaign Demonstrates

Three conjectures survived witness replay, plantri-pool replay, Eberhard-lattice search, exhaustive plantri screening, constructor double-checking, RL search, Hopper search, and LLM search, yet none was accepted as a proof because its conclusion is not a consequence of the current formal inventory. They remain open rather than proved or refuted: the system does not read a failed counterexample search as proof, nor a stuck prover as evidence of falsity.

Open Conjecture 1. *If $p_3 = 0$ and $f_2 \geq 17$, then $p_6 \geq 4 - 2S$.*

The prover settled the cases $S = 0$ and $S \geq 2$ and reduced the conjecture to $S = 1$: exactly one face of size at least 7, at most one hexagon, and the remaining faces small. About 170 arithmetic candidates were decided by plantri (pipeline screen plus an independent audit up to $f_2 = 30$), and none is realizable, so the residual case needs a realizability lemma absent from the inventory. At $S = 1$ the entailment check finds no arithmetic contradiction, so the prover correctly stops rather than fabricating one.

Open Conjecture 2. *If $p_5 = 0$ and $f_2 \geq 7$, then $p_6 \geq 2 - S$.*

This conjecture has the smallest remaining gap. One elementary fact ($f_2 \geq m+1$, since a k-gon has k distinct neighbors) closes the $S = 1$ case, leaving a single finite hole at $\{4:6,\ 6:1\}$ that plantri proves non-realizable but that is not yet in the formal inventory.

Open Conjecture 3. *If $p_4 = 0$ and $f_2 \geq 17$, then $p_6 \geq 4 - 2S$.*

This is the analogue of Open Conjecture 1 with $p_4 = 0$ in place of $p_3 = 0$, and it requires a similar geometric exclusion principle.

Taken together, the campaign demonstrates three properties of the architecture. *Refutations are reusable*: once validated, a witness replays against future candidates at low cost, and replay accounts for just over half of all refutations (103 of 204, Table 3), so the counterexample pool measurably strengthens the campaign over time. *Exhaustive and stochastic search serve different roles*: plantri gives decisive construction or exhaustion within configured bounds while stochastic tracks extend coverage beyond them, and the system never treats a failed stochastic search as evidence for a conjecture. *Proof failure is informative without being a truth value*: the open cases show that arithmetic countermodels expose limitations of the current inventory even when exhaustive graph search finds no realizable counterexample, so the loop can identify the kind of mathematical principle that further progress requires, such as a realizability lemma, a stronger finite enumeration, or a fuller formal theory of embedded maps.

Finally, learning operates on outcomes without touching standards. Refutations yield repaired constants or hypotheses; proved and surviving statements yield sharpened or weakened variants; dynamic cells target uncovered regions; and semantic attack-surface signatures remove bounded duplicates. For instance, the candidate $p_6 \geq 5 - 3S$ was dropped as a semantic duplicate of Open Conjecture 1: the written coefficients differ, but the bounded violation sets coincide exactly (the same 144 candidate p-vectors under the pinned enumeration bounds), so the two statements impose the same demand on every feasible face vector. The system thus learns from the structure of previous search, not only from terminal truth status.

6 Current Limitations

First, the system targets a specific domain, and even there the conjecture language is restricted to affine inequalities over a small set of face-count invariants; richer quantified or structural statements are not yet supported.

Second, exhaustive results are bounded by the configured enumeration range. Outside that range, stochastic search is one-sided: a validated witness refutes a conjecture, but a failed search does not prove it.

Third, Lean results are relative to a closed, hand-audited inventory. Thirteen `sorry`-marked lemmas form the trusted base, because a complete formal theory of surface-embedded maps is not yet available in Mathlib. The opaque `IsMap` predicate prevents unrestricted fake instances but does not remove this foundational dependence; Appendix A lists the inventory and describes the consistency guard.

Fourth, three of the four machine-proved statements follow from established Jučovič- and Barnette-type machinery, and the fourth is an elementary face-count identity. This is by design: the contribution is the autonomous end-to-end pipeline, not a new deep theorem in polytope combinatorics.

Fifth, the individual search policies are not complete decision procedures: the RL action space does not reach some triangle-free targets, and the Hopper track contributes only marginally. They broaden coverage rather than guarantee it, which is why the pipeline relies on several complementary tracks together.

Sixth, the current evaluation is repository-version dependent. The reported counts refer to qualified artifacts in the repository snapshot used for this paper. The accompanying artifact package provides the registry, witness records, plantri cache, formal proof files, and exact execution configuration needed to reproduce the reported status.

Finally, the autonomy claim concerns execution, not system design. Humans selected the domain, implemented the components, audited the inventory, and fixed the configurations. After initialization, no human selects, routes, validates, formalizes, or writes hints for individual conjectures.

7 Conclusion

We presented a closed-loop autonomous system that, after initialization, generates conjectures, searches for counterexamples, validates witnesses, attempts formal proofs, reconciles artifacts, and constructs the next generation without human intervention between stages. Previous outcomes influence future proposals but never change the standards for accepting counterexamples, exhaustive verdicts, or formal proofs.

The campaign records 204 witness-verified refutations, four complete Lean proofs, and three open conjectures. These provide explicit evidence of falsity, formal certificates, and qualified unresolved cases, and together they expose the boundary between arithmetic feasibility, geometric realizability, and the current formal inventory. The contribution is not a claim to have solved a major open problem in polytope combinatorics, but a demonstration that an unattended mathematical campaign can repeatedly generate and evaluate its own statements while preserving reusable, independently inspectable artifacts. The architecture applies in principle to any domain where candidate objects can be searched or enumerated, counterexamples independently validated, and survivors connected to a formal proof system; which areas of mathematics admit such a combination is itself an interesting question about autonomous mathematical agents.

AI USE STATEMENT

Generative AI tools are integral components of the autonomous system, used for conjecture proposal, counterexample proposal, graph search, proof planning, Lean code generation, and outcome-conditioned proposal generation. The surrounding software enforces parsing, arithmetic checks, graph validation, independent witness verification, Lean compilation, theorem-signature checks, artifact qualification, and the zero-`sorry` policy; no AI-generated conjecture, counterexample, or proof is accepted without passing the corresponding computational or formal gate. The authors designed the system, selected the mathematical domain and formal inventory, implemented the verification architecture, and take responsibility for the final manuscript, software, claims, and artifacts.

REFERENCES

AlphaProof and AlphaGeometry teams, Google DeepMind. AI achieves silver-medal standard solving International Mathematical Olympiad problems. Google DeepMind blog, 2024.

Jeremy Avigad. The mechanization of mathematics. Lecture slides, Carnegie Mellon University, May 2019. `https://www.andrew.cmu.edu/user/avigad/Talks/mechanization_talk.pdf`.

David Barnette. On p-vectors of 3-polytopes. *Journal of Combinatorial Theory*, 7:99–103, 1969.

Gunnar Brinkmann and Brendan D. McKay. Fast generation of planar graphs. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):323–357, 2007.

Randy Davila. Automated conjecturing in mathematics with TxGraffiti. *arXiv preprint arXiv:2409.19379*, 2024.

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Victor Eberhard. *Zur Morphologie der Polyeder*. Teubner, Leipzig, 1891.

Jordan S. Ellenberg, Cristofero S. Fraser-Taliente, Thomas R. Harvey, Karan Srivastava, and Andrew V. Sutherland. Generative modelling for mathematical discovery. *arXiv preprint arXiv:2503.11061*, 2025.

Siemion Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics*, 72(1–3):113–118, 1988.

Branko Grünbaum. *Convex Polytopes*. Springer, 2nd edition, 2003. First edition: Wiley, New York, 1967.

Thomas C. Hales. Mathematics in the age of the Turing machine. In Rod Downey (ed.), *Turing's Legacy: Developments from Turing's Ideas in Logic*, Lecture Notes in Logic. Cambridge University Press, Cambridge, 2014.

Haocheng Ju, Guoxiong Gao, Jiedong Jiang, Bin Wu, Zeming Sun, Shurui Liu, Leheng Chen, Yutong Wang, Yuefeng Wang, Zichen Wang, Wanyi He, Peihao Wu, Liang Xiao, Ruochuan Liu, Bryan Dai, and Bin Dong. Automated conjecture resolution with formal verification. *arXiv preprint arXiv:2604.03789*, 2026.

Ernest Jučovič. On the number of hexagons in a map. *Journal of Combinatorial Theory*, 10:232–236, 1971.

Kazumi Kasaura, Naoto Onda, Yuta Oriike, Masaya Taniguchi, Akiyoshi Sannai, and Sho Sonoda. Discovering new theorems via LLMs with in-context proof learning in Lean. *arXiv preprint arXiv:2509.14274*, 2025.

Zenan Li, Zhaoyu Li, Kaiyu Yang, Xiaoxing Ma, and Zhendong Su. Learning to disprove: Formal counterexample generation with large language models. *arXiv preprint arXiv:2603.19514*, 2026.

Naoto Onda, Kazumi Kasaura, Yuta Oriike, Masaya Taniguchi, Akiyoshi Sannai, and Sho Sonoda. LeanConjecturer: Automatic generation of mathematical conjectures for theorem proving. *arXiv preprint arXiv:2506.22005*, 2025.

Ján Pastorek. AutoGraphForge: Towards automated graph theory discovery. *arXiv preprint arXiv:2609.03478*, 2026.

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*, 625:468–475, 2024.

Jasivan Alex Sivakumar, Philipp Borchert, Ronald Cardenas, and Gerasimos Lampouras. Conjecturing: An overlooked step in formal mathematical reasoning. *arXiv preprint arXiv:2510.11986*, 2025.

Ernst Steinitz. Polyeder und raumeinteilungen. *Encyklopädie der mathematischen Wissenschaften*, 3:1–139, 1922.

Jialiang Sun, Yuzhi Tang, Ao Li, Chris J. Maddison, and Kuldeep S. Meel. Enumerate-conjecture-prove: Formally solving answer-construction problems in math competitions. *arXiv preprint arXiv:2505.18492*, 2025.

Valentino Vito and Lim Yohanes Stefanus. Adaptive Monte Carlo search for conjecture refutation in graph theory. *arXiv preprint arXiv:2306.07956*, 2023.

Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*, 2021.

A Formal Inventory and Axiom Audit

The Lean development uses a closed, hand-audited inventory for geometric facts that are not yet available through a complete surface-map API in Mathlib. The inventory is fixed before the autonomous campaign and is shared by all proof attempts; it is not modified in response to individual conjectures.

The trusted base. The inventory rests on thirteen `sorry`-marked lemmas (fifteen `sorry` occurrences in total, since two statements split into cases). Eleven are the named foundational facts, each taken from one of three source papers: *euler_formula*, *handshake*, *regularity*, *p_range*, *occupation_conservation*, *occupation_bound*, *quad_occ_cancellation*, and *equality_family* (elementary face-count and occupation facts); *JucovicTheorem* (the Jučovič inequality conjunct); *P6GenusG* (the genus-g p_6 statement); and *Barnette_P6Bound* (the Barnette critical-face-adjacency bound). The remaining two, *sum_ico_3_7_weighted* and *key_sum*, are low-level summation identities that support the eleven. All higher-level components, including the Jučovič and p_6 inequality parts used by the proved conjectures, are *proved* from this base without `sorry`, and the prover is forbidden from introducing any new `sorry` of its own.

The consistency guard. Each base lemma is guarded by the opaque predicate `IsMap`, which has no introduction rule. This guard is not cosmetic. Without it, the inventory is inconsistent inside Lean: the all-zero instance $\langle 3,\ \lambda_.\,0,\ 0,\ 0,\ \lambda_.\,0\rangle$ falsifies *euler_formula* (it forces $0 = 2$), from which `False`, and hence any conjecture, would be derivable with zero `sorry`. Requiring `IsMap`, whose only introduction routes are the hypotheses of the theorem being proved and the guarded existential of *equality_family* (which produces only maps with $m \geq 6$), removes this route, so fabricated instances can never satisfy the base assumptions. The reported theorems are therefore certificates relative to an explicit, auditable, and internally consistent inventory, not claims of a fully foundational formalization of surface-embedded maps.

Audit package. The final artifact package includes the inventory source (`polib/Inventory.lean`), the provenance of each imported fact, and the corresponding Lean audit. This makes the formal soundness boundary inspectable independently of the generated proof files.

B Registry Identifiers of Reported Statements

The main text refers to proved and open statements by theorem and conjecture numbers. Table 4 maps these to the identifiers used in the conjecture registry and in the artifact package, where each proved statement has a proof file `C{id}.lean` and each refutation a witness record `C{id}.json`.

Main text	Registry ID	Statement	Status and notes
Theorem 1	C104	$p_4 = 0,\ p_5 \le 2,\ f_2 \ge 7 \Rightarrow 2p_6 \ge 4 - S$	Proved; first end-to-end proof of the campaign
Theorem 2	C201	$p_4 = p_5 = 0,\ S \ge 1 \Rightarrow p_6 \ge 2 - S$	Proved; completed July 4, 2026
Theorem 3	C215	$p_4 = p_5 = 0,\ f_2 \ge 7 \Rightarrow 3p_6 + 2S \ge 12$	Proved; completed July 5, 2026; seven lemmas, zero `sorry`
Section 5.2	C124	$2p_6 \le 2f_2 - 2p_4 - 2p_5$ (no hypotheses beyond M being a map)	Proved; elementary consequence of $f_2 = \sum_k p_k$; uses only *p_range* from the inventory
Open Conj. 1	C193	$p_3 = 0,\ f_2 \ge 17 \Rightarrow p_6 \ge 4 - 2S$	Open; residual case $S = 1$
Open Conj. 2	C195	$p_5 = 0,\ f_2 \ge 7 \Rightarrow p_6 \ge 2 - S$	Open; single finite hole $\{4:6,\ 6:1\}$
Open Conj. 3	C198	$p_4 = 0,\ f_2 \ge 17 \Rightarrow p_6 \ge 4 - 2S$	Open; analogue of C193
Section 5.3	C199	$p_6 \ge 5 - 3S$	Dropped as a semantic duplicate of C193 (same 144 bounded violators)

Table 4: Registry identifiers for all proved, open, and discussed statements. $S = \sum_{k \ge 7} p_k$ throughout.

C Extended Generator Details

The generator combines Graffiti3, dynamic cells, mutation rules, a support-function miner, and an LLM co-proposer. Its full-pool gate checks candidates against all available verified p-vectors rather than a small sample.

The mutation engine performs exact-rational constant repairs, hypothesis repairs, sharpening, and controlled weakening. Refuted constants can be shifted beyond the worst violating witness. Additional hypotheses can be introduced to exclude known violating families. Proved and surviving statements can be moved toward the supported pool boundary or weakened by removing selected hypotheses.

The support-function miner searches fixed hypothesis cells and affine directions for tight supported inequalities. A candidate is retained only when it touches multiple distinct rows, has sufficient support, and is non-vacuous. The vacuity guard rejects bounds whose hypotheses already force an information-free conclusion. The semantic-duplicate filter compares bounded attack surfaces and removes equivalent candidates.

D Extended Refutation and Prover Details

The counterexample pipeline stores every validated witness with its p-vector, edge list, validation metadata, and planar drawing. Cached witnesses are replayed through the complete validator. Plantri stores both realizability and non-realizability verdicts within the configured bounds.

The stochastic tracks run in isolated processes and share a stopping event. A validated witness stops the search; cancellation of other tracks is normal control flow. Failure of all tracks leaves the conjecture unrefuted.

The prover uses dependency-signature injection, signature compilation pre-checks, blueprint-cache eviction, and a cross-run memory of compile failures. Before an expensive proof attempt, a proposed

helper signature is compiled against Mathlib, the formal inventory, and the current Polib environment.

A formal artifact is accepted only if the complete file compiles, the root theorem is present, no active `sorry` occurs, and no failed node is recorded. A file containing only successful helper lemmas is not classified as a completed proof.

E Terminal Status Semantics

The campaign uses distinct statuses for proposal, search, proof, and artifact qualification. A conjecture is `refuted` only when a witness has passed all five validation checks. A conjecture is `proved` only when a complete Lean artifact satisfies the root theorem, compilation, zero-`sorry`, and failed-node requirements.

A conjecture that survives counterexample search but is not entailed by the current inventory is recorded as `open` or `prover_failed`, depending on the formalization outcome. A stochastic search that exhausts its budget without a witness does not produce a proof status. Infrastructure failures are kept separate from mathematical outcomes and do not generate negative learning signals.

This status discipline is used both for reporting and for learning. The generator may use different statuses to choose repairs, explore uncovered cells, or search for stronger consequences, but the status itself is never assigned from an LLM report without the corresponding computational or formal artifact. Disk artifacts are the single source of truth: an entry sitting in the proved bucket without a qualifying artifact is demoted, and a failed run writes no proof file at all.

F Engineering Measurements

The repository reports several engineering measurements. After overriding an inherited extended-thinking configuration (the child `claude -p` call had inherited an amplified effort setting), the LLM counterexample track changed from repeated timeouts exceeding 240 seconds with zero output to complete rounds of approximately 8 seconds with five parsed candidates. Prompt restructuring further reduced round latency from approximately 152 seconds and 10.6K output tokens to approximately 78 seconds and 5.7K output tokens.

Seeded A/B tests showed that single-threaded Torch pools made the RL track approximately 25% faster than a ten-thread configuration while freeing approximately 9.8 CPU cores (the ten-thread setting incurred OpenMP spin overhead on small PPO tensors). A minimum-degree-five plantri dispatch extended exhaustive reach from $f_2 \leq 26$ to $f_2 \leq 36$ for the dominant candidate class.

G Autonomous Outcome Signals

Signals are derived from the persistent registry and on-disk artifacts before each generation; no manually curated hint file is required. Refutation signals contain violating p-vectors, margins, witness families, and hypothesis cells. Proof signals contain theorem status, dependencies, and hypothesis cells. Survivor and prover-stuck signals contain exhausted search paths, arithmetic countermodels, and missing realizability or inventory principles.

Signals are regenerated from current artifacts. Consequently, later demotions, successful reruns, deletions, or new counterexamples are automatically reflected in the next generation.

H Reproducibility and Campaign State

The campaign state is distributed across the conjecture registry, verified counterexample records, plantri verdict cache, formal proof artifacts, and evolution records. Before each generation, these sources are reconciled. This allows the autonomous loop to recover from different entry points, including a full project run, a direct formalization run, or a counterexample search invoked for a registered conjecture.

The registry stores conjecture identifiers, canonical statements, hypotheses, conclusions, outcome status, search metadata, and artifact pointers. Counterexample artifacts store the violating p-vector, witness graph, validation information, and rendered drawing. Proof artifacts store the complete Lean source, compilation result, root theorem, dependencies, and quality checks. The generator reads these records to construct the next proposal context.

The final artifact package should contain an anonymized repository snapshot or tagged release identifying the exact registry, witness records, plantri cache, formal inventory, and Lean environment used for the reported campaign. This package makes the numerical status and terminal artifacts reproducible independently of future repository changes.