GRAFFITI3LOOP: Generating and Resolving Formal Machine-Generated Graph Conjectures in a Loop

William Hu [*1] **Michael Douglas** [*1] **Randy Davila** [*2] **Philip Vonderlind** [*3] **Simon Frieder** [*3]

Abstract

We introduce GRAFFITI3LOOP, the first loop that autonomously generates formal graph-theoretic conjectures in Lean 4 and attempts to solve them. The conjectures are machine-generated by GRAFFITI3 from finite invariant tables, automatically translated into typed Lean 4/`mathlib` statements, and checked (where Lean4 versions allow) with Lean-FRO Comparator so that a submitted proof must solve the intended formal challenge rather than merely compile. We use a batch of 330 selected conjectures to ground this loop in concrete outputs, which we call GRAFFITI3BENCH-330. In a zero-tool baseline with three open-weight Lean prover checkpoints, no tested output was accepted; we therefore report a failure-mode analysis covering moved or missing Lean APIs, unsolved goals, missing answer blocks, impermissible proof shortcuts, and malformed proof fragments. By contrast, a tool-using, human-guided agentic triage that ran through GRAFFITI3BENCH-330 produced 42 Lean-checked labels: 17 proofs and 25 explicit disproofs. The appendix contains a commentary on how interesting these are from a mathematical point of view. A further 85 items have computational counterexamples, while 203 remain likely true after bounded counterexample search. As a counterpoint, an agentic baseline without tool use was also run. The benchmark part of this loop is intended to measure AI-for-mathematics systems on machine-originated conjectures that sit closer to mathematical discovery workflows than static exercise corpora, including proof, refutation, and hypothesis-repair settings, paving the way for a new engineering-driven approach to mathematics.

[1]Harvard University [2]First Principles [3]Fields Model Foundation. Correspondence to: All authors (equal contribution): <williamhu@alumni.harvard.edu, mdouglas@cmsa.fas.harvard.edu, randy@firstprinciples.org., philip.vonderlind@aimoprize.com, simon.frieder@cs.ox.ac.uk>.

Preprint. June 26, 2026.

1. Introduction

Formal proof assistants make mathematical claims executable: a proof either checks against a precise statement and trusted kernel, or it does not. This property is attractive for evaluating AI systems for mathematics, where fluent natural-language arguments and plausible proof scripts can hide gaps, type errors, or statement drift. Existing formal-mathematics benchmarks, e.g. PutnamBench (Tsoukalas et al., 2024), are often drawn from human-written libraries, olympiad-style tasks, or curated theorem lists. These datasets are valuable, but they only partially model the research loop in which new conjectures are generated from evidence, filtered for interest, and then proved, refuted, or repaired.

GRAFFITI3LOOP targets this loop. Its inputs are graph-theoretic conjectures produced by GRAFFITI3, a modern member of the Graffiti/TXGRAFFITI family of automated conjecturing systems (Davila, 2026a;b). GRAFFITI3 treats a conjecture as a provisional law supported by a finite invariant table, not as a theorem. Our work introduces appropriate Lean/`mathlib` definitions that facilitate automatic translation of conjectures into Lean 4 and asks theorem-proving systems to (dis)prove the supplied statement.

This setting differs from ordinary proof-completion benchmarks (Tsoukalas et al., 2024; Yu et al., 2025) in three ways. First, the conjectures are machine-originated rather than copied from an existing formal library. Second, ordinary proof-completion benchmarks use graph invariants whose Lean 4 interfaces may be partial, immature, or version-sensitive. Third, some snapshot-true conjectures may require strengthening or refutation, so the benchmark must keep provenance and checking semantics explicit. For proof-generation tasks, we therefore evaluate with Comparator (Lean Prover Contributors, 2026a), using the generated formal statement as a challenge and the model output as a proposed solution. While such conjecturing loops have been studied before (Poesia et al., 2024; Dong & Ma, 2025; Han et al., 2022), the level of mathematics was elementary, as the focus of such earlier work was on gaining deeper understanding of the learning process of an autonomous agent, rather than a production-level system that is capable of producing autonomously expert-level mathematics.

The goal is not merely to add another collection of Lean exercises. A useful AI-for-mathematics system should eventually participate in a discovery workflow: inspect evidence, formulate a plausible statement, prove it, find a counterexample, or suggest a corrected hypothesis. GRAFFITI3LOOP isolates the main verifiable slice of that workflow. The conjecturing system supplies unfamiliar but structured graph statements, while the formal checker supplies an unambiguous success criterion for proof attempts. This lets the loop measure whether a prover can cope with statements that originate outside the usual human-curated theorem-library distribution. The 330 conjectures that make up GRAFFITI3BENCH-330 ground this loop in a concrete, static benchmark, to carry out analyses.

A side results of our investigation is that we obtain a new comparison of base models that solve Lean challenges against tool-based agentic systems on unseen, novel mathematical problems. Prior work (Requena et al., 2026) relied mostly on existing benchmarks, such as PutnamBench, and as such may have been biased as math problems were seen.

Contribution summary.

- A pipeline from GRAFFITI3 theory selection to Lean 4 benchmark items with provenance metadata. (Section 2 and Section 4.)
- A Lean-facing invariant interface separating native `mathlib` concepts, wrappers, benchmark definitions, and deferred items. (Section 3.)
- Zero-tool negative results for three open-weight Lean provers, Goedel-ProverV2-32B DeepSeek-Prover-V2-7B, and KiminaProver-72B, across all 330 conjectures from GRAFFITI3BENCH-330. We provide failure analysis and classify errors. (Section 5.)
- A tool-based, agentic triage, based on Claude Code, was able to resolve 42 conjectures (17 proofs, 25 counterexamples).[1] These results are complemented by a non-tool using run through Claude Code. (Section 6.)
- Commentary on the found proofs and counterexamples. (Section A and B.)

2. From GRAFFITI3 to Benchmark Items

2.1. Automated Conjecturing Background

Graffiti. Fajtlowicz's *Graffiti* was an early automated conjecturing system for graph theory. It maintained a database of graphs and numerical invariants, generated candidate relations among those invariants, and filtered them using the Dalmatian heuristic: conjectures had to be true on the current database and empirically significant relative to prior accepted bounds (DeLaViña, 2005b; Larson & Van Cleemput, 2016).

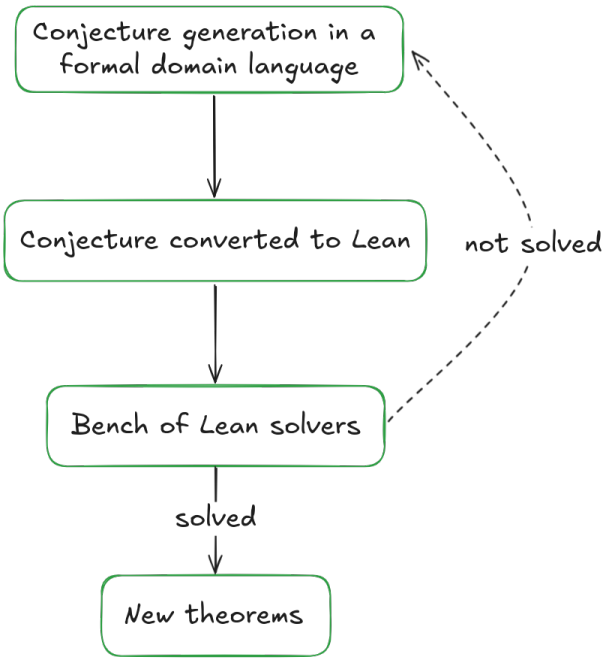

Figure 1. GRAFFITI3LOOP pipeline. GRAFFITI3 selects conjectures from a snapshot invariant table; the benchmark layer translates them into Lean 4 challenges; model completions are checked against the intended formal statement with Comparator.

Graffiti.pc and Sophie. DeLaVina's *Graffiti.pc* adapted this workflow for personal computers and educational use. It made conditional hypotheses more explicit and supported classroom exploration of machine-generated graph conjectures. The Sophie heuristic shifted attention from necessary inequalities to sufficient-condition patterns and structural explanations (DeLaViña, 2005a; DeLaViña et al., 2007).

Polyhedral and optimization systems. GraPHedron uses invariant vectors and convex hulls to identify facet inequalities that are sharp on finite graph corpora; PHOEG extends this style with database-centric support for extremal graph workflows (Mélot, 2008; Devillez et al., 2019). AutoGraphiX takes a complementary optimization-first approach by searching for extremal or counterexample graphs and then extracting conjectural structure from the results (Caporossi & Hansen, 2000).

TxGraffiti and Graffiti3. TXGRAFFITI modernizes the Graffiti tradition around a Python-based conjecturing workflow and has produced graph-theoretic conjectures leading to published work (Davila, 2026a). GRAFFITI3 abstracts this into theory selection over a finite, evolving invariant table

[1] Positive results are shared at: https://github.com/hellofieldsmodel/graffiti3loop

and outputs compact libraries of snapshot-true statements designed for human inspection, counterexample search, and downstream proof work (Davila, 2026b).

Algorithm 1 Benchmark generation from a snapshot invariant table

Require: Snapshot table T, grammar $\mathcal{G}$, conjecture budget b
Ensure: Benchmark set $\mathcal{B}$ of Lean 4 challenges with metadata
Detect the base hypothesis B from predicates true on every row
Build a pruned hypothesis family $\mathcal{H}$ by support thresholding and duplicate-slice removal
for each target invariant $I^\star$ and hypothesis $h \in \mathcal{H}$ **do**
  Generate candidate relations $h \Rightarrow I^\star \,\square\, f$
  Reject candidates falsified on the active table slice
  Apply redundancy and empirical-envelope filters
  Record equality, slack, support, and grammar provenance metadata
end for
Rank survivors with Hazel and keep a budgeted theory library $\mathcal{C}$
Translate selected items to Lean 4 and package them as challenges
return $\mathcal{B}$

2.2. Theory Selection

Let T be a snapshot invariant table over a finite graph set $\mathcal{O} = \{o_1, \ldots, o_N\}$. Columns consist of numerical invariants and Boolean graph predicates. A grammar $\mathcal{G}$ determines admissible hypotheses, expressions, comparisons, and implications, inducing a language $\mathcal{L}(\mathcal{G})$ of candidate conjectures. Given $(\mathsf{T}, \mathcal{G}, b)$, GRAFFITI3 selects a finite theory

$$\mathcal{C} \subseteq \mathcal{L}(\mathcal{G}), \qquad |\mathcal{C}| \leq b,$$

whose members are true on the snapshot and collectively remain compact, readable, and nonredundant.

The numerical columns embed the observed graphs as an invariant cloud. The theory-selection problem is therefore a capacity-control problem over that cloud: the grammar controls which relations can be expressed, the budget controls how many statements can be reported, and the heuristic layer favors conjectures that sharpen an empirical envelope somewhere on the observed objects. The benchmark records this finite-evidence origin because it affects how failures should be interpreted. A conjecture may be hard because it is mathematically deep, because it is false outside the snapshot, or because the Lean interface for the relevant invariant is incomplete.

The generator first detects a base hypothesis B as the conjunction of predicates true on every row. For example, if `connected` and `nontrivial` hold throughout the snapshot, then $B = \texttt{connected} \wedge \texttt{nontrivial}$; if no predicate holds throughout the table, B is `True`. It then builds a pruned family of hypotheses

$$h = B \wedge P_{i_1} \wedge \cdots \wedge P_{i_\ell}, \qquad 0 \leq \ell \leq \ell_{\max},$$

discarding slices with too little support and formulas that define duplicate support sets. Thus, if `regular` and `regular`$\wedge$`cubic` pick out the same observed graphs, the shorter formula is retained as the more readable hypothesis. For each target invariant $I^\star$ and retained hypothesis h, GRAFFITI3 searches conjectures of the form

$$\forall G, \quad h(G) \Rightarrow I^\star(G) \,\square\, f(G), \qquad \square \in \{\leq, \geq, =\},$$

where f is drawn from low-complexity expressions in the remaining invariant columns. The expression grammar includes selected pairwise derived columns such as $\min(X, Y)$, $\max(X, Y)$, $|X - Y|$, and $X \cdot Y$, as well as affine envelopes over feature maps $\Psi(G) = (\phi_1(G), \ldots, \phi_d(G))$. It can also use low-dimensional convex-hull stages: clouds involving the target invariant are converted into facet inequalities and then rewritten with $I^\star$ on the left. Candidate affine or facet bounds are rationalized, filtered for human-readable coefficients, and rechecked on the active table slice before they can enter the theory.

The raw search is intentionally high-recall, so GRAFFITI3 compresses it before reporting a theory library (Davila, 2026b). The first compression step removes redundant specializations: if a bound found under a restrictive hypothesis also holds on a larger slice of the table, the more general statement is preferred. The second step keeps only bounds that improve the empirical envelope somewhere on their slice. For an upper bound this means being smaller than the current retained upper envelope for at least one observed graph; for a lower bound it means being larger somewhere. This is the role played by Dalmatian-style significance filtering in the Graffiti tradition (DeLaViña, 2005b; Larson & Van Cleemput, 2016). Finally, surviving bounds are ordered using tightness and slack information, especially the number of observed graphs on which the proposed bound is attained. In GRAFFITI3 terminology these correspond to Morgan-style redundancy removal, Dalmatian-style envelope improvement, and Hazel-style ordering (Davila, 2026b).

The same table can also be queried in the opposite direction. Once a numerical event has been selected, such as equality in a proposed bound or strict improvement over it, the generator searches for short structural formulas that are true on the observed graphs exhibiting that event. Formally, for an event such as $I^\star(G) = f(G)$ or $I^\star(G) < f(G)$, it searches for a predicate formula p, possibly a conjunction of predicates and negations, satisfying

$$\forall G, \quad \big(I^\star(G) \,\square\, f(G)\big) \Rightarrow p(G), \qquad \square \in \{\leq, \geq, =\}$$

on the snapshot. Thus p is a structural property that appears necessary for the selected event within the finite evidence table; it can later suggest equality cases, counterexample searches, or stronger hypotheses. This is related to the Sophie line of Graffiti-style structural explanation heuristics (DeLaViña et al., 2007; Davila, 2026b). In the present benchmark these annotations are metadata around the proof task, not separate scored tasks.

For GRAFFITI3LOOP, the important semantic point is that snapshot truth does not yet mean theorem. A generated item enters the benchmark as a candidate formal statement with provenance, not as a certified mathematical fact. The present paper evaluates proof generation for supplied Lean 4 statements; counterexample and hypothesis-repair tasks are part of the broader discovery setting and require separate scoring rules.

2.3. Benchmark records

Each selected conjecture is packaged as a benchmark record rather than only as a theorem string. The record stores the generated informal schema, the translated Lean 4 challenge, the target invariant, the hypothesis slice, the right-hand-side expression, and provenance metadata from theory selection, including support, tightness, slack, and the heuristic stage that retained the item. This metadata is not used to give the model extra help in the zero-tool baseline. Its purpose is to make later audits possible: a failed item can be traced back to the invariant table, a missing Lean declaration, a version mismatch, or a genuinely unresolved mathematical task. A compact generation summary is given in Section 2.1.

3. Lean Translation

The generated conjectures were then translated into Lean 4 theorem statements. In this section, we describe some of our modeling and design decisions.

We model graphs using `mathlib`'s `SimpleGraph V` structure. (The mathlib Community, 2020), the type consisting of all undirected simple graphs whose vertices are the elements of the type `V`. Because we only consider finite graphs in this benchmark, we additionally instantiate `V` with the typeclass `[Fintype V]`. We are aware that Mathlib has a library for directed graphs called `Digraph`. However, at the time of writing, Mathlib's theorem base for `Digraph` was significantly less comprehensive than the one for `SimpleGraph`. We leave expanding the library to `Digraph` as future work.

Some of the GRAFFITI3 graph invariants had not yet been defined in `mathlib`. Therefore, we had to implement some missing GRAFFITI3 graph invariants in Lean. To facilitate smooth translation from Graffiti conjectures to Lean 4, we kept the names of the Lean invariants the same as the ones in Graffiti. However, we attempt to reuse `mathlib` theorems where possible. For example, Graffiti has an invariant called `independence_number`. `mathlib` already has a definition for independence number inside `SimpleGraph.indepNum`, so this invariant required no additional implementation

For ease of use, we next attempted to make the output types as specific as possible. Invariants that naturally return integers in a mathematical sense should be integer valued valued in Lean. This appears often for the matrix invariants; for example, the adjacency matrix for a graph is most naturally expressed as an integer matrix. However, there may be cases where a downstream invariant would like to treat the adjacency matrix as a real or complex matrix in order to extract its eigenvalues; in this case, we decree that it is the downstream invariant's responsibility to coerce the integer matrix to its more general form. The full list of invariants and their mathematical definitions are listed in Section C.

4. Evaluation Protocol

4.1. Models

We evaluate three open-weight Lean prover checkpoints: Goedel-Prover-V2-32B (Lin et al., 2026), DeepSeek-Prover-V2-7B (Ren et al., 2025), and Kimina-Prover-72B (Wang et al., 2025). Model weights and configurations are taken from their public releases (Goedel-LM, 2025; DeepSeek-AI, 2025; AI-MO, 2025), and inference is run with vLLM (Kwon et al., 2023). We do not report frontier or proprietary model results in this systematic baseline.

The baseline is deliberately minimal. Each model receives the Lean 4 file for a conjecture and returns one completion inside an answer block. No Lean server, tactic feedback loop, retrieval tool, or self-correction agent is available during generation. This makes the experiment reproducible and isolates direct proof-generation behavior, but it is not intended as the strongest possible configuration for agentic theorem proving.

After generation, completions are cached before checking. The harness extracts the code inside the requested answer block, normalizes imports and variable declarations, prepends the fixed preamble in Listing 1, and submits the resulting challenge/solution pair to Comparator. This normalization reduces failures caused only by duplicated boilerplate while preserving failures caused by missing proof content, invalid syntax, impermissible axioms, or proof of the wrong statement.

Listing 1. Schematic benchmark item.

```
import Mathlib
import TxGraffitiBench.Imports

open scoped BigOperators
```

```
universe u
variable {V : Type u} [Fintype V] [
    DecidableEq V]
```

4.2. Comparator checking

Evaluation uses Comparator (Lean Prover Contributors, 2026a). The original conjecture is supplied as the *Challenge*; the model's extracted completion is supplied as the *Solution*. Comparator checks both sides in a clean Lake workspace, confirms that the solution proves the statement supplied by the challenge rather than a nearby or redefined statement, and rejects impermissible axioms or shortcuts. Backward-compatible export is provided through lean4export (Lean Prover Contributors, 2026b) where supported.

This criterion replaces informal build checks and guards against models attempting to reward-hack towards generating a formal proof that passes the Lean kernel by using hacks rather than correct mathematics. A file that compiles in a model-created environment is not counted as success unless Comparator confirms that it proves the intended statement. Consequently, earlier build-only validity counts are not used in the reported results.

4.3. Version compatibility

The checking stack for the reported experiment used a Lean v4.28 Comparator environment, while several evaluated checkpoints were trained or packaged around older Lean/`mathlib` APIs. The lean4export support used by Comparator has backported compatibility down to Lean v4.15, but earlier releases require additional work. For models closer to Lean v4.9, one can either downgrade the TXGRAFFITI library and Comparator stack or evaluate in the current environment and report version drift as a failure mode. We take the latter approach here, so the reported numbers are not version-normalized scores.

4.4. Prompt and Checking Details

The zero-tool prompt asks the model to return exactly one `<answer>` block containing a complete Lean 4 file with no `sorry`. The prompt template is:

Listing 2. Zero-tool completion prompt template.

```
You are completing a Lean 4 formalization.
    Below is the current partial file:

```lean4
{}
```

Be concise. Do not write any explanation,
    proof sketch, or commentary.
Output a readable, well-formatted Lean 4
    file with sensible line breaks,
consistent indentation, and only the code
    needed to complete the file.

Proof requirement: Provide a complete proof
     that compiles without sorry.
Keep tactics concise. Prefer Mathlib lemmas
    , simp, omega, linarith,
exact, apply, calc, etc. Provide correct
    imports and setup code for
every statement that you use.

Reply with your Lean 4 program inside
    exactly one pair of tags:
<answer>
... complete Lean 4 source ...
</answer>
```

The harness extracts that block, normalizes imports and variable declarations, prepends the fixed preamble in Listing 1, and submits the challenge/solution pair to Comparator. Invalid completions are assigned approximate failure classes by parsing Comparator output and the extracted Lean source. These classes are diagnostic labels, not mathematical judgments about the underlying conjectures.

5. Baseline Results

None of the tested zero-tool completions was accepted by Comparator as a proof of its intended formal challenge. The pass rate for each evaluated model is therefore zero under this protocol. This result is negative but diagnostic: the models fail in different ways, and the failures identify where current open-weight proof generators are brittle on machine-generated graph conjectures. The failure-mode plot is computed from the updated prediction reports containing explicit failure annotations; older build-only archives are not used for these counts.

The dominant failures are consistent with Lean-version drift: completions often reference imports and declarations from older environments, while the benchmark library and Comparator target a newer release. DeepSeek-Prover-V2-7B and Goedel-Prover-V2-32B also frequently produce proof states with unresolved goals. One recurring Goedel pattern is a chain of local `have` declarations restating the target obligation rather than advancing the proof. Kimina-Prover-72B is the only tested model with a recurring class of malformed top-level proof fragments.

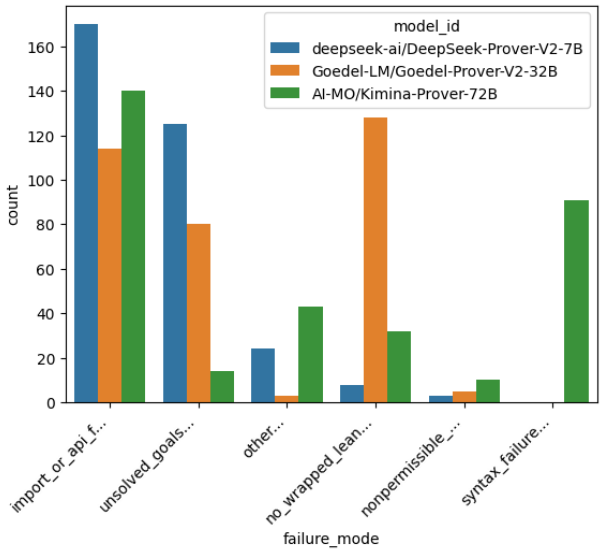

Figure 2. Failure-mode counts for evaluated completions. Since no tested completion passed Comparator, the plot summarizes where the zero-tool baseline breaks down. See Table 3 for full label names.

Listing 3. Representative malformed proof fragment.

```
have h15 : G.degree (Classical.arbitrary V)
    % 2 = 0 := by
  have h10 : forall v : V, G.degree v % 2 =
      0 := by
    intro v
    apply SimpleGraph.IsEulerian.even_degree
        hG v
  apply h10
```

These categories are diagnostic rather than mathematical labels. An import/API failure says that the emitted file did not reach the intended proof obligation in the active environment; it does not say whether the underlying conjecture is true. Similarly, an unsolved-goals failure may reflect missing lemmas, a weak prompt, or genuine proof difficulty. Comparator is important precisely because it prevents these cases from being conflated with success while still allowing the benchmark to report where the proof-generation pipeline broke. Figures 3 and 4 and Section 5.1 report descriptive statistics for completion sizes, which help distinguish missing-output failures from very long but unsuccessful proof attempts.

However, these results should not be read as a ranking of the tested systems or as evidence that generated graph conjectures are beyond LLM theorem provers in general. They show that direct completion from current open-weight checkpoints is not robust enough for this benchmark under a modern Lean environment. A tool-using comparison with Lean feedback is the natural next experiment, since it would separate mathematical difficulty from repairable interface friction.

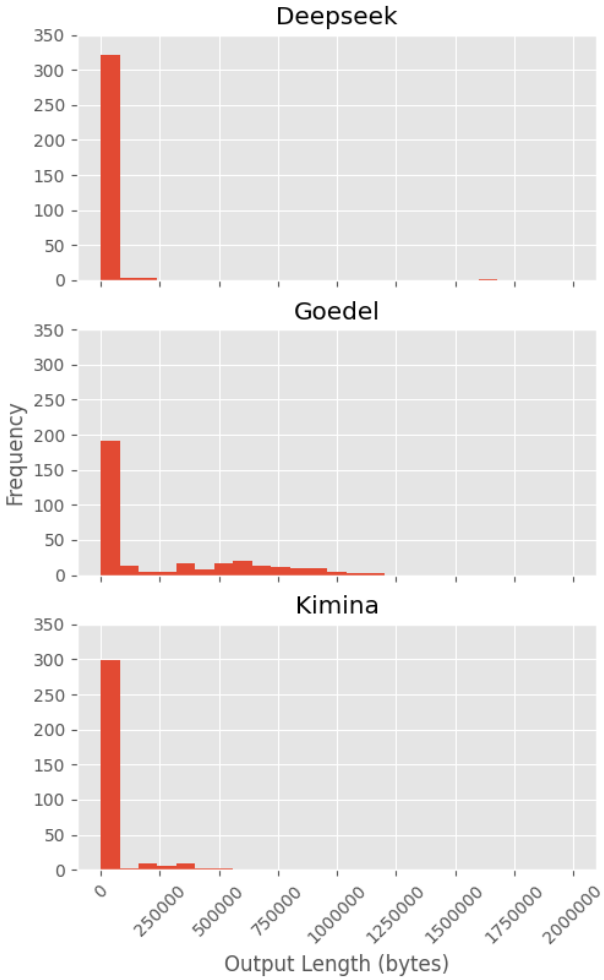

Figure 3. Histograms of completion sizes for the three models.

5.1. Completion-size statistics

6. A human–AI agent baseline

6.1. Agent with tools

The zero-tool baseline isolates direct proof generation, but it is not intended to be the strongest possible theorem-proving configuration. As a contrasting reference point, and to obtain machine-checked labels for part of the batch, we also ran a *tool-using, human-guided agent* over the same items; implementation details are reported in Section 6.3. This setting is not directly comparable to the zero-tool results because it includes compiler feedback, human steering, and iterative file editing rather than a single base-model completion. We include it to show that a nontrivial subset of the benchmark is within reach of an agentic workflow.

Running through 330 conjectures, the agent produced 42 results that pass the same kernel-level discipline used

Model	Shortest	Longest	Average
DeepSeek	442	1,614,528	9,607.5
Goedel	590	1,174,228	243,282
Kimina	131	537,242	29,510

Table 1. Completion-size statistics for the three models. All columns are measured in bytes.

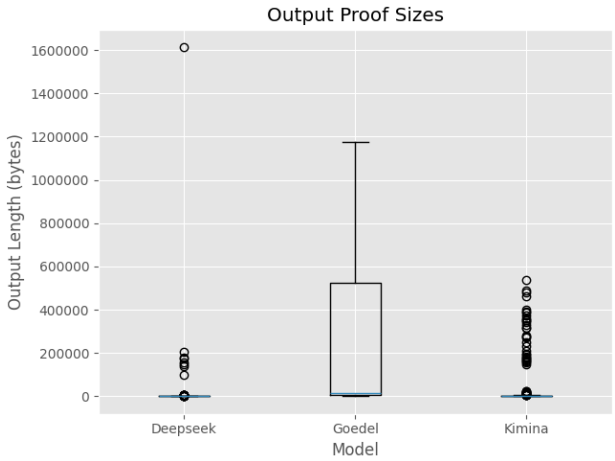

Figure 4. Boxplots of completion sizes for the three models.

in Comparator-style checking: Lean 4 accepts the files, no proof uses `sorry`, and the only axioms reported are `propext`, `Classical.choice`, and `Quot.sound`. These results consist of **17** proofs of likely true items and **25** disproofs, each formalized as a $\neg\forall$ statement discharged by an explicit counterexample. The disproofs come from false items whose invariants are already available through `mathlib` or thin wrappers. The proofs cover tractable cases including bipartite, tree, and degree bounds; an edge double-count for `regular`$\Rightarrow 2\alpha \leq n$; a parity argument; and claw-free or complete-graph cases. Sections A and B give a more detailed triage of the mathematical results.

For completeness, we note here other key ways Claude Code was evaluated in an environment different from the open-weight models due to the fact that Claude could rewrite certain sections of the library. Claude does not import the TxGraffitiBench Lean library; rather, it redeclares certain benchmark invariants (except with differences below); the files otherwise import only `mathlib`. Claude claimed to discover and fix the following two bugs in the Lean library. However, we believe our results are still broadly interpretable even given these differences.

1. First, our existing definition of `has_claw` did not ensure that the four vertices were distinct. However, we do not believe that these changes would drastically affect the results on the open-weight models, due to the fact that 1) not every conjecture involves `has_claw` and 2) the conjectures get *easier* to disprove with our erroneous version of `has_claw`.

2. The typeclass `[DecidableRel G.Adj]` was also added to the definitions of various degree-related properties. We do not think this was a major change since the decidability of propositions could have been assumed using Lean's classical logic. Therefore, we do not consider this difference a bug.

Table 2. Status of the 330 items after triage and agent formalization.

Status	**#**	**Basis**
Formalized in Lean 4	42	kernel-checked: 17 proofs + 25 disproofs
Certain-false, not formalized	85	explicit computational counterexample
Likely-true, unresolved	203	no counterexample ≤ 11 vertices
Total	330	

Table 2 summarizes the resulting labels: 42 items are machine-checked, a further 85 are false with a concrete but not yet formalized counterexample, and 203 remain only "likely true." Two observations are relevant to the benchmark's role as a discovery proxy. First, the first two items in the batch, $\alpha \leq a(G)$ and $\mathrm{residue}(G) \leq \alpha$, recover known bounds for the annihilation number and Havel–Hakimi residue (Larson & Pepper, 2011) Thus the benchmark recovers genuine graph theory, not only overfit inequalities from a finite table. Second, the batch also contains open problems: several items relating the zero-forcing number to α on cubic graphs are instances of TXGRAFFITI's α–Z conjecture, whose claw-free cubic case and asymptotic cubic variants have been studied separately (Davila & Henning, 2020). A benchmark item being unproved by every model can therefore reflect model weakness, statement falsity, a broken Lean 4 definition, or an open conjecture; this is why ground-truth labels and definition audits matter.

6.2. Agent without tools

To probe how much of the agentic result comes from the loop rather than the base model, we re-ran the same 42 machine-checked items—the 17 proofs and 25 disproofs that form the formalized, ground-truth subset of the 330-item batch—as a single-shot, closed-book task. For each item, a fresh `claude-opus-4-8` agent received the natural-language statement and the fixed invariant signatures and had to decide its truth value and emit a complete Lean proof, either of the statement or of its negation, in

one response, with no access to the compiler, proof state, or library search; tool use was prohibited, and the prohibition held across the entire fleet: an audited zero tool calls over all 42 agents. Each output was graded by isolated compilation against the same Lean v4.28.0 / Mathlib v4.28.0 build, counting an item as solved only when its proof term elaborated free of `sorry`. The model decided the truth value correctly for 31 of 42 items, or 74%, but produced a compiling proof for only 2, or 5%: a proof that a graph with $2\alpha = \Delta$ cannot be cubic, since cubicity forces $\Delta = 3$, yielding the impossible equality $2\alpha = 3$; and a refutation of "K_4-free $\Rightarrow n + 3 \leq 5\alpha$" by the triangle K_3, which is vacuously K_4-free yet has $n = 3$ and $\alpha = 1$. The gap between deciding, 74%, and proving, 5%, is the central finding: the failures are overwhelmingly incomplete proofs and incorrect Mathlib lemma names rather than wrong mathematics or wrong directional judgments. The corpus also stresses formalization more than the propositional core suggests—most items require explicitly constructed counterexample graphs with decidable adjacency and computed degree, size, or edge-count invariants, precisely the library fluency that single-shot generation lacks. Because these 42 are exactly the items the tool-using agent had already formalized, this is a same-set ablation: removing the compiler loop and the human collapses the outcome from $42/42$ to $2/42$, while truth-value accuracy stays high. Together, these results locate the binding constraint of single-shot proof synthesis on this benchmark in Mathlib API recall under a compiler feedback loop rather than in mathematical reasoning: the base model largely knows *what* is true and *why*, but cannot reliably name the lemmas that discharge it blind, which is exactly what the loop supplies.

6.3. Reproducibility

The triage and agent baseline were produced in a single interactive session of a command-line coding agent driven by a frontier general-purpose model, with no fine-tuning and no retrieval over a proof corpus. The agent had shell, file-editing, and web-search tools, plus a general-purpose Lean 4 workflow skill that supplies build-first conventions but no domain-specific solver. Unlike the zero-tool baseline of Section 4, this workflow kept compiler feedback in the loop: the agent edited a file, ran the Lean 4 elaborator, read the error, and iterated. It should therefore be interpreted as a tool-using, human-guided workflow rather than a single model-completion baseline.

There was no fixed prompt. The work was steered by natural-language instructions in roughly the following order: recover the benchmark's Lean 4 invariant definitions and build the library; triage all 330 items by counterexample search into a per-item verdict CSV; formalize the easy disproofs, namely false items with `mathlib`-native invariants, as $\neg\forall$ statements instantiated at the triage counterexample and discharged on small explicit graphs by `decide`; formalize the tractable proofs, factoring out reusable lemmas such as a bipartite $n \leq 2\alpha$ bound, a complete-graph toolkit, and an edge double-count; and audit the corrected `claw_free` predicate and the degree-elaboration issue. The artifact's `mods/` directory records the corresponding source-level patch for `claw_free` and a separate dataset patch adding complete graphs used by the counterexample triage. Summing the per-turn processing-time annotations in the session log, the triage and all 42 formalizations took ≈ 1.2 hours of active agent time.

Table 3. Diagnostic failure classes for invalid completions.

Class	Meaning
Import/API failure	Moved imports, renamed declarations, or older namespace structure.
Unsolved goals	Syntactically meaningful code leaves proof obligations unresolved.
Other checking failure	Invalid completion not assigned to a more specific class.
No wrapped Lean	No usable Lean file appears inside the requested answer tags.
Nonpermissible	Comparator rejects an impermissible proof shortcut or axiom use.
Syntax failure	The emitted code is not valid Lean syntax.

7. Limitations and Extensions

Snapshot evidence. GRAFFITI3 conjectures are generated from finite evidence. A benchmark item may be true, false, or dependent on hypotheses that need strengthening. The current proof-generation task asks models to prove the supplied statement, so its pass rate is meaningful only for Comparator-accepted proofs and should not be interpreted as a theorem-only success rate. Counterexample and repair tasks require separate scoring rules.

Lean coverage. Graph theory in `mathlib` is substantial but not tailored to every invariant used by TXGRAFFITI. Some generated conjectures must be held back until their definitions and basic lemmas exist in the benchmark library.

Version fairness. Evaluating older Lean provers in a newer environment measures practical robustness but can understate performance relative to a model's native environment. A version-matched release requires a missing-definition enumeration for each supported Lean/`mathlib` target, including v4.15 and v4.9.

Model scope. The reported baseline covers open-weight prover checkpoints only. Frontier proprietary models and agentic systems with Lean feedback may behave differently and should be evaluated separately.

Lifting these limitations can improve GRAFFITI3LOOP. Fur-

ther, the present experiments evaluate proof generation. The same generated conjecture records can support additional tasks once corresponding verification rules are implemented:

- construct a certified counterexample to a false conjecture;
- minimize a counterexample by vertex count and then edge count;
- characterize equality cases for a valid inequality on a finite universe;
- strengthen a bound or weaken a hypothesis while preserving validity.

8. Conclusion

GRAFFITI3LOOP turns automated graph-theory conjecturing into a formal-mathematics loop. It samples new conjectures from a data-driven discovery pipeline, translates them into Lean 4 challenges, and judges attempted proofs. The first evaluated batch, GRAFFITI3BENCH-330 identifies the main requirements for stronger AI-for-mathematics evaluations: broader invariant coverage, version-matched metadata, and tool-using baselines that can distinguish mathematical difficulty from interface repair and also showcases new conjectures. This work adds to the growing evidence that (certain parts of) mathematics are being transformed into an engineering discipline: large and complex neuro-symbolic systems generate conjectures, working in tandem with tool-based agentic solvers to (dis)prove them. Mathematicians orchestrate and tune these systems, point them to worthwhile theorems, and review and interpret the generated results, reflecting the new workflow performed by the authors of this paper. The 330 conjectures from GRAFFITI3BENCH-330 were an initial set to showcase how such loops, but sets that are two orders of magnitudes larger are possible.

References

AI-MO. Kimina-Prover-72B. Hugging Face model repository, 2025. URL https://huggingface.co/AI-MO/Kimina-Prover-72B. Accessed 21 May 2026.

Caporossi, G. and Hansen, P. Variable neighborhood search for extremal graphs: 1. the AutoGraphiX system. *Discrete Mathematics*, 212(1–2):29–44, 2000. doi: 10.1016/S0012-365X(99)00206-X. URL https://doi.org/10.1016/S0012-365X(99)00206-X.

Davila, R. Automated conjecturing with TxGraffiti. *Annals of Mathematics and Artificial Intelligence*, 2026a. doi: 10.1007/s10472-026-10005-5. URL https://doi.org/10.1007/s10472-026-10005-5. Online first, 14 April 2026.

Davila, R. Graffiti3: Compact theory libraries for automated mathematical discovery. Research Square preprint, 2026b. URL https://doi.org/10.21203/rs.3.rs-8493329/v1. Posted 19 January 2026.

Davila, R. and Henning, M. A. Zero forcing in claw-free cubic graphs. *Bulletin of the Malaysian Mathematical Sciences Society*, 43(1):673–688, 2020. doi: 10.1007/s40840-018-00705-5.

DeepSeek-AI. DeepSeek-Prover-V2-7B. Hugging Face model repository, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-Prover-V2-7B. Accessed 21 May 2026.

DeLaViña, E. Graffiti.pc: A variant of Graffiti. In Fajtlowicz, S., Fowler, P. W., Hansen, P., Janowitz, M. F., and Roberts, F. S. (eds.), *Graphs and Discovery*, volume 69 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 71–79. American Mathematical Society, Providence, RI, 2005a. Proceedings of a DIMACS Workshop, New Brunswick, NJ, USA, 10 November 2001.

DeLaViña, E. Some history of the development of graffiti. In Fajtlowicz, S., Fowler, P. W., Hansen, P., Janowitz, M. F., and Roberts, F. S. (eds.), *Graphs and Discovery*, volume 69 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 81–118. American Mathematical Society, Providence, RI, 2005b. Proceedings of a DIMACS Workshop, New Brunswick, NJ, USA, 10 November 2001.

DeLaViña, E., Pepper, R., and Waller, W. Independence, radius and hamiltonian paths. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):481–510, 2007.

Devillez, G., Hauweele, P., and Mélot, H. PHOEG helps to obtain extremal graphs. In Fortz, B. and Labbé, M. (eds.), *Operations Research Proceedings 2018*, pp. 251–257. Springer, Cham, 2019. doi: 10.1007/978-3-030-18500-8_32. URL https://doi.org/10.1007/978-3-030-18500-8_32.

Dong, K. and Ma, T. STP: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Goedel-LM. Goedel-Prover-V2-32B. Hugging Face model repository, 2025. URL https://huggingface.co/Goedel-LM/Goedel-Prover-V2-32B. Accessed 21 May 2026.

Han, J. M., Rute, J., Wu, Y., Ayers, E., and Polu, S. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=rpxJc9j04U.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Larson, C. E. and Pepper, R. Graphs with equal independence and annihilation numbers. *The Electronic Journal of Combinatorics*, 18(1):P180, 2011. doi: 10.37236/667.

Larson, C. E. and Van Cleemput, N. Automated conjecturing I: Fajtlowicz's dalmatian heuristic revisited. *Artificial Intelligence*, 231:17–38, 2016. doi: 10.1016/j.artint.2015.10.002. URL https://doi.org/10.1016/j.artint.2015.10.002.

Lean Prover Contributors. Comparator: A trustworthy judge for Lean proofs. GitHub repository, 2026a. URL https://github.com/leanprover/comparator.

Lean Prover Contributors. lean4export: Plain-text declaration export for Lean 4. GitHub repository, 2026b. URL https://github.com/leanprover/lean4export.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., and Jin, C. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j4C0nALrgK. ICLR 2026 poster; arXiv:2508.03613.

Mélot, H. Facet defining inequalities among graph invariants: The system GraPHedron. *Discrete Applied Mathematics*, 156(10):1875–1891, 2008. doi: 10.1016/j.dam.2007.09.005. URL https://doi.org/10.1016/j.dam.2007.09.005.

Poesia, G., Broman, D., Haber, N., and Goodman, N. D. Learning formal mathematics from intrinsic motivation. *Advances in Neural Information Processing Systems*, 37:43032–43057, 2024.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL https://arxiv.org/abs/2504.21801.

Requena, B., Letson, A., Nowakowski, K., Ferreiro, I. B., and Sarra, L. A minimal agent for automated theorem proving. *arXiv preprint arXiv:2602.24273*, 2026.

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

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *Advances in Neural Information Processing Systems*, 37:11545–11569, 2024.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Dos Santos, M., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., Lu, J., de Saxcé, H., Bailey, B., Song, C., Xiao, C., Zhang, D., Zhang, E., Pu, F., Zhu, H., Liu, J., Bayer, J., Michel, J., Yu, L., Dreyfus-Schmidt, L., Tunstall, L., Pagani, L., Machado, M., Bourigault, P., Wang, R., Polu, S., Barroyer, T., Li, W.-D., Niu, Y., Fleureau, Y., Hu, Y., Yu, Z., Wang, Z., Yang, Z., Liu, Z., and Li, J. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11354.

Yu, Z., Peng, R., Ding, K., Li, Y., Peng, Z., Liu, M., Zhang, Y., Yuan, Z., Xin, H., Huang, W., et al. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025.

A. Proof triage for agent-formalized true items

This table summarizes the 17 true benchmark items formalized in `Proofs.lean`. We write $n = |V(G)|$, $m = |E(G)|$, $\alpha = \alpha(G)$, and $\Delta = \Delta(G)$, and suppress the shared connected/nontrivial hypotheses unless they affect the argument. The novelty assessment is preliminary: it records whether the item appears to be a standard graph-theoretic consequence or a potentially interesting generated rediscovery.

Table A.1. Mathematical triage of the agent-formalized true benchmark items.

Item	Human-readable statement	Proof idea	Difficulty and novelty triage
NC-5	If G is bipartite, then $n \leq 2\alpha$.	The two color classes are independent, so each has size at most α, and their sizes sum to n.	Elementary; standard textbook fact.
SC-4	If $2\alpha < n$, then G is not bipartite.	Contrapositive of NC-5.	Trivial once NC-5 is known; no independent novelty.
NC-102	Every nontrivial graph has $1 \leq \alpha$.	A singleton vertex set is independent.	Trivial; sanity-check item.
SC-10	If $\alpha \leq 2$, then G is claw-free.	A claw has three pairwise nonadjacent leaves, giving an independent set of size 3.	Elementary; standard observation.
NC-10	If G is a tree, then $m + 1 \leq 2\alpha$.	A tree is bipartite and satisfies $m+1 = n$; apply NC-5.	Elementary consequence; not new.
SC-2	If $2\alpha \leq m$, then G is not a tree.	For a tree, NC-10 gives $m+1 \leq 2\alpha$, hence $m < 2\alpha$.	Contrapositive-style corollary; not new.
SC-3	If $2\alpha < m + 1$, then G is not a tree.	Direct contrapositive of NC-10.	Immediate corollary; no independent novelty.
NC-4	If G is triangle-free, then $\Delta \leq \alpha$.	The neighborhood of a maximum-degree vertex is independent; an edge inside it would form a triangle.	Elementary but useful; classic fact.
SC-20	If $\alpha < \Delta$, then G is not triangle-free.	Contrapositive of NC-4.	Immediate corollary; not new.
NC-32	If G is bipartite and subcubic, then $m \leq 3\alpha$.	The handshake lemma and subcubicity give $2m \leq 3n$; NC-5 gives $n \leq 2\alpha$.	Standard degree-counting consequence.
SC-45	If $2\alpha = \Delta$, then G is not cubic.	Cubic graphs have $\Delta = 3$, but 2α is even.	Trivial parity obstruction; generated but not mathematically novel.
NC-158	If G is bipartite and claw-free, then $m \leq 2\alpha$.	In a bipartite graph, three neighbors of one vertex are pairwise nonadjacent, so claw-freeness forces $\Delta \leq 2$. Hence $m \leq n \leq 2\alpha$.	One of the more interesting generated items; still an elementary structural consequence, likely a rediscovery rather than new.
SC-87	If $\alpha < 2$, then G is regular.	Since G is nontrivial, $\alpha = 1$; every distinct pair of vertices is adjacent, so G is complete and hence regular.	Elementary complete-graph case; not new.
SC-30	If $\alpha < 2$, then G is regular and claw-free.	Same complete-graph argument as SC-87; complete graphs are claw-free.	Elementary corollary; not new.
NC-19	If G is connected, nontrivial, and regular, then $2\alpha \leq n$.	Let S be a maximum independent set. In a d-regular graph with $d > 0$, all $d\|S\|$ edges incident with S go to $V \setminus S$, while $V \setminus S$ can receive at most $d\|V \setminus S\|$ such incidences.	Moderate among this set; standard double-counting lemma, not new but mathematically substantive.
NC-31	If G is cubic, then $3\alpha \leq m$.	Combine NC-19 with the cubic handshake identity $2m = 3n$.	Standard consequence; useful benchmark item but not new.
SC-6	If $m < 3\alpha$, then G is not cubic.	Contrapositive of NC-31.	Immediate corollary; not new.

B. Counterexample triage for agent-formalized false items

The 25 formalized disproofs in `Falsifications.lean`, `Falsifications2.lean`, and `Falsifications3.lean` are $\neg\forall$ statements discharged by explicit small graphs. The table below groups them by the graph that refutes them. This grouping is more informative than listing all 25 rows separately: many false conjectures fail for the same structural reason.

Table B.1. Counterexample triage for the agent-formalized false benchmark items.

Counterexample	Items refuted	Witnessed facts	Mathematical diagnosis
K_3	NC-9, NC-68, SC-9, NC-67, NC-37, NC-44, NC-57, NC-84, NC-112, SC-32	K_3 is connected, nontrivial, K_4-free, subcubic, and regular, with $\alpha = 1$, $\omega = 3$, $\chi = 3$, and $\Delta = 2$.	The hypothesis K_4-free is too weak to control clique number, chromatic number, or degree expressions; the triangle is the minimal obstruction.
K_6	NC-129, NC-131	$n = 6$, $m = 15$, and $\alpha = 1$.	Dense complete graphs immediately refute bounds that try to control order or size only through α.
K_7	SC-158	$n = 7$, $\alpha = 1$, and K_7 contains a K_4, while $3\alpha + 10 < 2n$.	A linear order–independence inequality does not force K_4-freeness.
G_6: triangle with three pendants at one triangle vertex	SC-8, SC-12	$n = 6$, $m = 6$, $\alpha \geq 4$, and the graph contains a triangle.	Sparse graphs with large independent sets need not be bipartite or triangle-free; a small local obstruction survives the global inequality.
G_7: K_4 with three pendants at one clique vertex	SC-35	$n = 7$, $m = 9$, $\alpha \geq 4$, and the graph contains a K_4.	The inequality $m < 3\alpha$ is too weak to force K_4-freeness.

Counterexample	Items refuted	Witnessed facts	Mathematical diagnosis
G_a: hub joined to six vertices, with two extra disjoint leaf edges	SC-84, SC-29, SC-83, SC-86, SC-36	$n = 7$, $m = 8$, $\alpha = 4$, $\Delta = 6$, so $3\alpha = 2\Delta$; the hub has degree 6 and forms a claw.	The equality $3\alpha = 2\Delta$ does not imply subcubicity, claw-freeness, or combined structural restrictions.
G_b: star on seven vertices plus one extra leaf–leaf edge	SC-53	$n = 7$, $m = 7$, $\alpha = 5$, so $3\alpha = 2n + 1$, but the graph is not a tree.	An equality involving n and α can hold after adding a cycle-creating edge; it does not characterize trees.
$G_c = K_7 - e$	SC-132, SC-69	$n = 7$, $m = 20$, $\alpha = 2$, so $m = 10\alpha$; degrees are 5 and 6, so the graph is not regular.	The equality $m = 10\alpha$ does not impose regularity; near-complete graphs give very small counterexamples.
G_d: a K_4 plus three extra leaves adjacent to one clique vertex	SC-114	$n = 7$, $\alpha = 4$, $\Delta = 6$, so $3\alpha = 2\Delta$, and the graph contains a K_4.	The same equality $3\alpha = 2\Delta$ also does not force K_4-freeness.

Overall, the formalized counterexamples look like diagnostic failures of finite-table pattern selection rather than surprising new graph phenomena. The main mathematical lesson is that many generated sufficient conditions are missing local forbidden-subgraph hypotheses: global linear relations among $m, n, \alpha, \Delta, \omega, \chi$ do not by themselves enforce bipartiteness, clique-freeness, claw-freeness, tree structure, or regularity.

C. Invariant vocabulary

The current benchmark snapshot uses the following typed numerical invariant columns.

Table 6. Typed numerical invariant vocabulary used by the benchmark snapshot.

Type	Columns
$\mathbb{N}$	order, size, maximum_degree, minimum_degree, diameter, radius, clique_number, chromatic_number, independence_number, vertex_cover_number, matching_number, triameter, slater, annihilation_number, residue, domination_number, total_domination_number, independent_domination_number, min_maximal_matching_number, zero_forcing_number
$\mathbb{Q}$	harmonic_index
$\mathbb{R}$	spectral_radius, largest_laplacian_eigenvalue, second_largest_adjacency_eigenvalue

Throughout this appendix, $G = (V, E)$ is a finite simple undirected graph, $n = |V|$, and $m = |E|$. For $v \in V$, $d(v)$ is the degree of v, $N(v)$ is its open neighborhood, and $N[v] = N(v) \cup \{v\}$. Distance-based invariants are evaluated on connected rows; $d_G(u, v)$ denotes graph distance and $e(v) = \max_{u \in V} d_G(u, v)$ denotes eccentricity.

Size, degree, and distance. order is n, size is m, maximum_degree is $\Delta(G) = \max_{v \in V} d(v)$, and minimum_degree is $\delta(G) = \min_{v \in V} d(v)$. The diameter, radius, and triameter columns are

$$\mathrm{diam}(G) = \max_{u,v} d_G(u, v), \quad \mathrm{rad}(G) = \min_{v} e(v),$$

$$\mathrm{tri}(G) = \max_{u,v,w} \big(d_G(u, v) + d_G(u, w) + d_G(v, w) \big).$$

Subsets, colorings, and coverings. clique_number is the maximum size of a clique; chromatic_number is the least k admitting a proper k-coloring; independence_number is the maximum size of an independent set; vertex_cover_number is the minimum size of a vertex cover; and matching_number is the maximum size of a matching.

Domination and matching variants. domination_number is $\gamma(G) = \min\{|D| : N[D] = V\}$. total_domination_number is $\gamma_t(G) = \min\{|D| : N(D) = V\}$, on rows with no isolated vertices. independent_domination_number is the minimum size of an independent dominating set. min_maximal_matching_number is the minimum size of a maximal matching.

Degree-sequence invariants. Let $d_1^{\downarrow} \geq \cdots \geq d_n^{\downarrow}$ be the degree sequence in nonincreasing order, and $d_1^{\uparrow} \leq \cdots \leq d_n^{\uparrow}$ the same sequence in nondecreasing order. The slater column is

$$\mathrm{sl}(G) = \min \left\{ k : k + \sum_{i=1}^{k} d_i^{\downarrow} \geq n \right\},$$

and `annihilation_number` is

$$a(G) = \max\left\{k : \sum_{i=1}^{k} d_i^{\uparrow} \le m\right\}.$$

`residue` is the Havel–Hakimi residue obtained by repeatedly deleting the first positive degree r, subtracting 1 from the next r entries, and reordering until only zeros remain.

Zero forcing. Starting from a blue set $B \subseteq V$, repeatedly apply the rule that a blue vertex with exactly one white neighbor forces that neighbor blue. `zero_forcing_number` is the minimum size of a set B that eventually colors all vertices blue.

Harmonic and spectral invariants. `harmonic_index` is

$$H(G) = \sum_{uv \in E} \frac{2}{d(u) + d(v)}.$$

Let $A(G)$ be the adjacency matrix and $L(G) = D(G) - A(G)$ the Laplacian. `spectral_radius` is $\max_i |\lambda_i(A)|$, `second_largest_adjacency_eigenvalue` is $\lambda_2(A)$, and `largest_laplacian_eigenvalue` is the largest eigenvalue of $L(G)$.