Formalizing Numerical Analysis: An Agent Pipeline and Quality Audit Beyond Kernel Acceptance

Theodore Meek [*1 2] **Siyuan Ge** [*1 3] **Di Qiu Xiang** [1 2 3] **Simon Chess** [1 2 3] **Vasily Ilin** [1 2 4]

Abstract

Recent work has demonstrated that coding agents can formalize entire advanced mathematics textbooks in Lean 4, yet existing efforts concentrate on branches of mathematics already well-represented in `mathlib` and measure success solely through kernel acceptance. We address both limitations by applying a coding agent to formalize *Numerical Methods for Ordinary Differential Equations*, a textbook in numerical analysis that is largely absent from `mathlib`, stressing the agent's capacity to develop new theory from scratch. We further introduce a systematic, reproducible three-dimensional framework for evaluating the quality of agent-produced formalizations beyond compilation: semantic correctness, Mathlib reuse, and cross-file reuse via LLM-as-judge methods. Applying this framework to our own formalization and to the released outputs of RepoProver and M2F, we uncover recurring unfaithful formalization patterns—including incomplete multi-part statements, added weakening hypotheses, and parameter restrictions—that kernel acceptance entirely obscures. Our results suggest that compilation-based metrics substantially overstate formalization quality, and we provide a reproducible audit methodology to support more rigorous evaluation of future autoformalization systems.

[1]Math AI Lab, University of Washington, Seattle, United States [2]Department of Mathematics, University of Washington, University of Washington, Seattle, United States [3]Department of Computer Science & Engineering, University of Washington, Seattle, United States [4]Department of Applied Mathematics, University of Washington, Seattle, United States. Correspondence to: Theodore Meek <theomeek@uw.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

Interactive theorem provers such as Lean (de Moura & Ullrich, 2021) make it possible to verify mathematics with machine-checked rigor. Lean's mathematical library, `mathlib` (mathlib Community, 2020), now contains hundreds of thousands of definitions and theorems spanning much of the undergraduate and graduate curriculum, making Lean an increasingly practical foundation for verifying research-level mathematics and for building machine-checkable mathematical knowledge at scale.

The principal obstacle to formalization at scale is its cost. Translating mathematics written in natural language into formal code is slow and labor-intensive and demands expertise in both mathematics and formal language. This obstacle has motivated a substantial body of work on autoformalization, attempts to use language models to automatically translate informal mathematics into formal statements and proofs (Szegedy, 2020; Wu et al., 2022; Jiang et al., 2023).

Three families of approaches have emerged. Specialized models trained specifically for formal mathematics have shown their performance on per-problem benchmarks such as miniF2F (Zheng et al., 2022), ProofNet (Azerbayev et al., 2023), and PutnamBench (Tsoukalas et al., 2024), with recent systems including DeepSeek-Prover-V2 (Ren et al., 2025), Goedel-Prover-V2 (Lin et al., 2025), Kimina-Prover (Wang et al., 2025), and Aristotle (Achim et al., 2025). General-purpose frontier models, applied through prompting and multi-model cooperation, have proven surprisingly capable at formalization without task-specific training (Wu et al., 2022). Most recently, coding agents, which are frontier models equipped with tools, file access, compiler feedback, and the ability to act over many steps, have been used to formalize mathematics at the scale of entire textbooks, producing formal developments of hundreds of pages in days (Urban, 2026; Wang et al., 2026; Gloeckle et al., 2026; Liu et al., 2026; Ren et al., 2026).

As autoformalization moves from isolated problems to whole textbooks, two concerns become pressing. The first is generalizability. Most existing results concern mathematics already well represented in `mathlib`, and it is far less clear how these systems behave on mathematics that

`mathlib` does not yet cover, where the agent must develop the relevant theory from the ground up rather than retrieve it (Kumarappan et al., 2025; Gloeckle et al., 2026). The second is evaluation. Existing whole-textbook formalization efforts report "success" almost exclusively as whether the code type-checks and builds without `sorry`. However, kernel acceptance establishes only that a formalization is well-formed and compiled, not that it is semantically correct (Poiroux et al., 2025; Li et al., 2024a; Liu et al., 2025; Lu et al., 2024; Weng et al., 2025). For example, a compiled statement may be vacuously true (Li et al., 2024a) or may misinterpret quantifier scopes or implicit constraints (Zhang et al., 2025; Chen et al., 2026). Yet in the whole-textbook setting, quality beyond compilation is assessed only through limited, subjective spot-checks (Wang et al., 2026; Gloeckle et al., 2026).

In this work, we address both concerns. First, we apply a coding agent to formalize *Numerical Methods for Ordinary Differential Equations*, a textbook on the branch of numerical analysis that is almost entirely absent from `mathlib`, stressing the agent's ability to develop new theory rather than recombine existing library content. Second, we introduce a systematic and reproducible method for evaluating the quality of the resulting formalization, auditing it along three dimensions: (1) *semantic correctness*: whether each formalized statement faithfully captures the source; (2) *mathlib reuse*: whether the agent reuses existing `mathlib` results when it should, rather than re-formalizing them; and (3) *cross-file reuse*: whether it reuses results already formalized elsewhere in the project, rather than duplicating them. Our method combines automated analysis of the formalized code with an LLM-as-judge (Zheng et al., 2023; Zhang et al., 2025), and we demonstrate it through a case study on our own formalization and RepoProver and M2F's released results.

Our contributions are:

- We apply a coding agent to formalize a textbook in numerical analysis, a branch of mathematics largely uncovered by `mathlib`, providing a stress test of agentic autoformalization outside the well-covered domains studied in prior work.
- We introduce a systematic, reproducible method that evaluates formalization quality beyond kernel acceptance along three dimensions—semantic correctness, mathlib reuse, and cross-file reuse—using LLM-as-judge.
- We report a detailed evaluation of our formalization under this method, characterizing where and how current agentic formalization succeeds and falls short.

2. Related Work

2.1. Agentic Autoformalization

Early autoformalization treated translation as a single-shot, per-statement task (Wu et al., 2022; Jiang et al., 2023; Azerbayev et al., 2023), and specialized provers continue to push statement- and proof-level performance on benchmarks such as miniF2F and ProofNet (Ren et al., 2025; Lin et al., 2025; Wang et al., 2025). A more recent line equips frontier models with tools, file access, compiler feedback, and multi-step control, yielding *agentic* formalization systems (Liu et al., 2026; Ren et al., 2026). Several concurrent efforts have scaled this paradigm to entire textbooks and projects: a recent effort formalized general topology in Megalodon (Urban, 2026); M2F compiled hundreds of pages of analysis into a buildable Lean project (Wang et al., 2026); RepoProver orchestrated thousands of parallel agents to formalize a graduate algebraic-combinatorics textbook in Lean (Gloeckle et al., 2026); and further multi-agent and cross-system efforts target additional textbook material (Brown et al., 2026; Bryant et al., 2026). These works demonstrate that agentic formalization at the textbook scale is feasible, but they work on a well-formalized branch of mathematics and report success primarily as kernel acceptance. In contrast, we study a branch largely absent from `mathlib`, and we focus on evaluating the quality of agent-produced formalization.

2.2. Formalization Evaluation

It is well-established that compilation in a proof assistant is a necessary but not sufficient condition for a faithful formalization (Poiroux et al., 2025; Li et al., 2024a; Liu et al., 2025; Weng et al., 2025). Surface-form metrics such as BLEU correlate poorly with correctness (Lu et al., 2024), motivating dedicated approaches: symbolic and bidirectional-equivalence checking (Li et al., 2024a; Poiroux et al., 2025), back-translation and learned alignment models (Lu et al., 2024; Gao et al., 2025), and LLM-as-judge methods (Peng et al., 2025; Zhang et al., 2025; Chen et al., 2026). These methods, however, evaluate isolated statements against a single informal counterpart. Our work differs in two aspects: we evaluate agent-produced formalization at the project scale rather than per statement, and we introduce structural dimensions by analyzing mathlib usage and cross-file usage, which assess how a formalization integrates with the library and project. Such an integration becomes essential for large-scale formalization.

3. Methods

3.1. Formalization Pipeline

Our pipeline is an autonomous loop driven by a Python orchestrator that repeatedly invokes four LLM roles: a *Planner*, a *Worker*, an *Evaluator*, and a *Consultant* summoned only on escalation. Like recent project-scale autoformalization frameworks such as RepoProver (Gloeckle et al., 2026) and M2F (Wang et al., 2026), we target whole Lean projects rather than isolated theorems. We only commit edits when Lean builds. We differ from M2F's fixed two-stage statement then proof pipeline by running an open-ended loop whose target is chosen each cycle from the live repository state. The Planner and Worker roles are implemented using Claude Opus 4.6; the Evaluator uses Claude Sonnet. We replace RepoProver's sketcher/prover/reviewer split with a Planner/Worker/Evaluator/Consultant decomposition coordinated through GitHub Actions.

Stateless agents and shared state. The agents themselves are stateless: no agent retains memory across cycles. All inter-cycle communication happens through files committed to the repository and through GitHub itself. The key artifacts are `strategy.md`, written by the Planner and read by the Worker (overwritten each cycle); `task_results/cycle_NNN.md`, the Worker's per-cycle report; `issues/`⟨name⟩`.md`, blocker descriptions; `history.jsonl`, one record per cycle holding the Evaluator's structured judgment (score, summary, stuck-on, recommendation); and `cycle`, `heartbeat.json`, and `run.lock`, which hold control state for the orchestrator and watchdog.

Per-cycle pipeline. Each cycle proceeds through the following stages:

- **Synchronization.** The orchestrator pulls the latest commits and queries GitHub Actions for build status. If the Lean build is failing, the Planner is hard-coded to fix the build first.
- **Planning.** The Planner reads `plan.md`, current `sorry` locations, recent task results, open issues, and recent cycle history, and emits a concrete instruction set in `strategy.md`.
- **Execution.** The Worker edits the Lean source tree following a sorry-first strategy: state the theorem in full, verify that the skeleton compiles, then close each `sorry` one at a time. It uses Lean LSP tools and Mathlib search (LeanSearch (Gao et al., 2026), Loogle (Mathlib Community, 2024)). It commits its changes and writes a task-results file; if blocked, it opens an issues file.
- **Evaluation.** A separate LLM session reads the cycle's Git diff, the Worker's task result, and recent attempts, and emits a progress score in $[-2, 2]$, a one-sentence summary, the current blocker, a recommendation for the next Planner cycle, and a compacted line for `attempts.md`.
- **Mechanical gates.** Independently of the LLM judge, the orchestrator enforces hard constraints. We adopt an M2F-style verifier-certified refinement (Wang et al., 2026) check on the total `sorry` count across the source tree, forbidding it from strictly increasing; budget increases are rate-limited and require a rationale. Strategy compliance is enforced by regex-extracting the scope declared in `strategy.md` and checking the Worker's commits against it. A build gate ensures that modified files compile before the Worker is permitted to commit.
- **Escalation.** If recent history shows four or more consecutive non-positive scores, the Consultant role is invoked, and its advice is written to the issue tracker for the next Planner cycle.
- **Bookkeeping.** The evaluation record is appended to `history.jsonl` and the cycle counter is advanced.

Agent skills. Each agent is shaped by repository-level instructions (a `CLAUDE.md` or `AGENTS.md` file) automatically loaded at session start. These standing rules absorb content that, in earlier iterations, had to be supplied through repeated human prompting.

3.2. Analysis Methods

3.2.1. Semantic Correctness

We audit semantic faithfulness with two LLM-judge protocols that share a rubric but differ in what the judge reads.

Per-project NL/Lean pairing. Each corpus tracks formalized entities differently, so we extract per-entity records pairing a textbook NL statement with its Lean declaration header three ways. For Butcher we consume a curated `lean_status.json` with extracted NL. For the alg-comb / RepoProver corpus we walk the blueprint (`blueprint/src/chapter_*.tex`), whose environments carry `\lean{...}` and `\leanok` macros that we parse directly. For ReasBook, which ships only Lean, we pair each M2F-generated docstring (`/-- Theorem N.M ... -/`) with the following declaration; the NL is M2F's own gloss for 1,834 of 1,864 entities, so direct judgment on ReasBook is largely a self-consistency check, with the 30 Rockafellar §1.1 entities the only true-fidelity samples. In every case we keep only the Lean theorem statement and discard the proof.

Shared rubric. Each protocol compares two statements A and B by first answering $Q_1 \colon A \Rightarrow B$ and $Q_2 \colon B \Rightarrow A$. The score follows from a fixed truth table: $Q_1 \wedge Q_2$ yields **Faithful** (verbatim); $Q_1 \wedge \neg Q_2$ yields **Stronger** (A strictly stronger, a sound generalization, acceptable for a formalization audit); $\neg Q_1$ yields **Different** (divergent), except when multiple concatenated Lean declarations together cover A as a case split. We collapse {Stronger, Faithful} to be considered reasonable formalizations and {Different} to an unreasonable formalization. The rubric also demands semantic similarity in addition to the implication requirements detailed above. It also handles when extra Lean hypotheses make the statement *weaker*, not stronger, and will correct the score to Different.

Direct judgment (Lean vs. NL). The judge sees the textbook NL, the Lean declaration header, and any surrounding textbook prose, which may carry implicit hypotheses; A is the Lean statement, B the NL. We use DeepSeek-V4-Pro (DeepSeek-AI, 2026).

Round-trip judgment (NL vs. back-translated NL). A blind back-translator sees only the Lean and produces an NL rendering. The judge then compares textbook NL against that back-translation, following (Wu et al., 2022), who observe that informalization is empirically easier than formalization. The dual-judge design follows (Li et al., 2024b). Here A is the back-translation and B the NL, with Q_1 primary: failure of Q_1 means the Lean cannot recover the textbook claim. Back-translation uses DeepSeek V4 Pro; the round-trip judgment uses `gpt-oss-120b` (OpenAI, 2025), a deliberately different model family to avoid correlated blind spots.

3.2.2. Mathlib Usage

We evaluate the formalizations on the originality of their contribution. As a heuristic for originality, we consider a proved statement (a `theorem`, `lemma`, etc) to be unoriginal if it can be proved by Mathlib theorems using `exact?+all`, a tactic which searches the available context for very simple proofs relying on a single theorem (if the theorem cannot be proven by anything in Mathlib we accept it as original, even if it is provable using `exact` in conjunction with a theorem proved by the project itself). In this way, we detect the extent to which a project is reproducing known results. It is worth noting that it is potentially healthy for a project to have a small number of theorems that are not considered "original" in this way. These may occur due to stylistic choices favoring shorter proofs. Nevertheless, a project that makes primarily novel contributions should consist of theorems primarily not provable by `exact?` under Mathlib.

3.2.3. Crossfile Usage

A high-quality formalization should reuse the results it has already established: when a textbook result S depends on an earlier result T, the Lean proof of S should invoke the formalized T rather than re-deriving it or routing around it. We measure this form of internal reuse by checking whether each textbook-extracted dependency is reflected as a reference in the Lean code. The analysis has two stages: we first extract a dependency graph from the textbook, and then test whether each dependency is reflected in the formalization.

Textbook dependency graph. We segment each textbook into entities (i.e., definitions, theorems, lemmas, propositions, and corollaries, organized as blocks in the textbook) and pair each statement with its proof. The pipeline accepts both unstructured input (text extracted from a PDF) and a structured LaTeX blueprint, from which entity identifiers and the associated Lean symbols are read directly from `\label{}` and `\lean{}` macros. Over these entities, we generate candidate dependency edges of $S use T$, using two heuristics.

- **Citation edges** are extracted when S explicitly cites T's index. The index is a textbook number in the block title, such as Theorem 142C" or a `\ref` to T's `\label` in the blueprint.
- **Keyword edges** are extracted when S mentions a term that T introduces. The introduced terms include the T's name in its block title, a quoted phrase in T's statement, and T's `\emph{}` term.

Both heuristics generate false-positive edges, and keyword edges in particular fire on generic vocabulary and cross-domain homonyms under different qualifiers. We therefore apply an LLM filtering: for each candidate edge, we prompt `deepseek-v4-pro` to verify the dependency based on the textbook content of both endpoints and the evidence that triggered the edge. The model returns a strict JSON verdict with a confidence and rationale, and we retain only edges judged real.

Reflection in the Lean code. For each textbook-extracted dependency $S \to T$, we obtain the Lean declaration of both endpoints and test whether S names T within its declaration body (statement and proof term). Because the test matches names verbatim, an implicit dependency via an intermediate helper lemma, a `simp` set, or definitional unfolding will not be registered. Therefore, the reported rate is a lower bound on true code-level dependence.

Table 1. Number of open sorries and unnecessary axioms across formalization systems. *All sorries in RepoProver correspond to exercises and are intentional.

System	Sorries	Axioms
RepoProver	5*	0
M2F	12	4
OpenMath	0	0

4. Results

4.1. Formalization Results

We applied the pipeline to the Butcher textbook on numerical methods for ordinary differential equations (Butcher, 2016). Of the 175 statements in the textbook, the pipeline produced fully proof-complete Lean formalizations for 84 (48.0%), and an additional 12 (6.9%) with the Lean statement in place and at least one `sorry` remaining, for a combined coverage of 54.9%. Coverage is uneven across chapters: four of the five chapters exceed 74% combined formalization, while Chapter 3 on Runge–Kutta methods, the largest by a wide margin and covering rooted trees, order conditions, A-stability, Padé approximants, order stars, AN/BN/algebraic stability, and the Butcher group, sits at 34.8% and accounts for the bulk of unformalized material.

4.1.1. Proof Gap

Proof completeness and axiom usage results are reported in Table 1. RepoProver's 5 open sorries correspond entirely to intentional exercise placeholders. The methods in which these are obtained are in Appendix C

4.2. Semantic Correctness

Results are shown in Table 2. Agreement percentage is found by comparing results from NL vs. Lean and NL vs. back- translated NL after grouping faithful and stronger categories together.

4.2.1. Qualitative results

To validate the direct judge, two of the authors with Lean expertise drew a stratified random sample of 20 OpenMath entities, 10 judged faithful or stronger (*OK*) and 10 judged divergent (*DIV*) with seed 42, and re-graded each one independently using the same Q_1/Q_2 rubric the judge was given. Disagreements were resolved by discussion.

We agreed with the judge on 10/10 OK verdicts and 8/10 DIV verdicts, for 18/20 overall. The eight agreed-DIV cases were divergences from the textbook, falling into recurring patterns documented below. The two disagreements both went in the same direction; the judge marked DIV entities we considered faithful, both times, because the LLM didn't understand standard Mathlib convention. Additionally, we inspected the four entities marked Stronger by the judge, and found that two are sound generalizations exhibiting interesting strengthenings.

Divergence: incompleteness. The most common DIV pattern in our sample is the Lean formalization covering only part of a multi-part textbook statement. Definition 312A defines three quantities simultaneously, elementary weights $\Phi(t)$, internal weights $\Phi_i(t)$, and derivative weights $(\Phi_i D)(t)$, through four coupled equations; the Lean formalization defines only `derivativeWeight`. Similarly, Definition 356B defines both DJ-reducibility and the explicit construction of the reduced method; the Lean defines only the reducibility predicate. These cases would require the case-split exception to recover, which applies only when multiple Lean declarations together cover the NL domain — a single declaration omitting a conjunct is correctly flagged Different.

Divergence: added hypotheses. A second recurring pattern is Lean formalizations that add hypotheses absent from the textbook, making the Lean statement strictly weaker. Theorem 514A is stated in Butcher without normwise hypotheses, while the Lean version requires an additional `h_norm_obligation` bounding the matrix norm. Theorem 520B adds invertibility (`h_inv`) and stage-equation (`hY_stage`) hypotheses not present in the NL. Definitions 530B and 530C similarly add explicit-method and Lipschitz hypotheses to a statement the textbook gives unconditionally. The rubric correctly treats these as DIV, since the added hypothesis breaks Q_1 (Lean $\Rightarrow$ NL) on the full textbook domain.

Divergence: parameter restriction. Theorem 550A states a result for an arbitrary $n \times n$ doubly companion matrix. The Lean formalization proves seven separate theorems for $n = 1, \ldots, 7$. The textbook's universal claim is not recovered by any finite union of these cases, and Q_1 fails for $n \geq 8$. The judge flags this as Different with the reason "Lean only covers $n = 1 \ldots 7$, not arbitrary n," which is the correct application of the rubric.

Generalization: Skolemizing existentials. The Stronger verdicts surfaced a complementary pattern in which the agent strengthens a textbook claim by computing witnesses explicitly. Theorem 406C states that "there exist constants C, D such that" a global-error recurrence bound holds. The Lean version `globalError_recurrence_bound_textbook` replaces the existential with explicit closed-form expressions for C and D in terms of the method coefficients and Lipschitz data. The Lean statement implies the textbook

Table 2. Semantic correctness evaluation across formalization systems via NL vs. Lean (direct, DeepSeek) and NL vs. back-translated NL (round-trip, GPT-oss-120b). Faithful = formalization is logically equivalent to source; Stronger = formalization entails source; Different = formalization diverges from source. †Agreement is computed by grouping faithful and stronger together.

		OpenMath	M2F	RepoProver
NL vs. Lean	Faithful	54% (51)	73% (1359)	82% (1173)
	Stronger	4% (4)	2% (42)	1% (12)
	Different	42% (40)	25% (463)	18% (253)
NL vs. NL_b	Faithful	46% (44)	61% (1050)	69% (931)
	Stronger	14% (13)	13% (225)	11% (152)
	Different	40% (38)	26% (447)	19% (258)
Agreement†		72.6%	74.7%	78.4%

one (instantiate C, D from the formulas) but not the reverse, so Q_1 holds while Q_2 does not — this is correctly classified as Stronger.

Generalization: redundant hypotheses removed. Theorem 523A states an energy identity for general linear methods, prefacing it with the assumption that the stability matrix M, weight matrix G, and diagonal scaling D are positive semi-definite. The Lean formalization `algebraicStability_identity` drops these PSD hypotheses and states the identity as an unconditional algebraic equality. This is mathematically correct — the equation is a linear-algebra identity that does not require PSD — and represents the agent recognizing that the textbook's hypotheses are stronger than the identity actually needs. The judge correctly flagged this as Different.

Disagreement: Skolemized definitions. The judge failed to recognize Skolemization in the opposite direction: when the textbook uses an existential and the Lean exposes the witness as a parameter rather than binding it inside an $\exists$, the judge reads the parameter as an added hypothesis. Figure 1 shows the canonical case. Butcher's Definition 142A defines power-boundedness as the existence of a constant C bounding all powers of A. The Lean `PowerBounded M a` takes the bound as an explicit parameter, and the textbook predicate is recovered as $\exists M, \texttt{PowerBounded}\ M\ a$, a standard Mathlib idiom for existentially defined properties. The judge flagged this as Different with the reason "Lean requires explicit M, NL requires existence of some C," reading the parameter binding as an added hypothesis. We consider this Faithful: under the Skolemized reading both Q_1 and Q_2 hold, and exposing witnesses as parameters is ubiquitous in Mathlib.

Implications. The audit suggests the direct judge is largely reliable: 18 of 20 verdicts in the stratified sample matched ours, and the eight DIV cases we agreed with correspond to formalization gaps that kernel acceptance would not have appeared — partial coverage of multi-part statements, added hypotheses, and parameter restrictions. The judge's errors cluster around a single failure mode — confusion about the direction of implication when the formalization re-parameterizes existentials, removes hypotheses, or states only one half of an iff — present in both the Skolemized DIV cases. The headline DIV rate in Table 2 slightly over-counts genuine divergence.

NL (Butcher 142A). A square matrix A is *stable* ("power-bounded") if there exists a constant C such that $\|A^n\| \leq C$ for all $n = 0, 1, 2, \ldots$

Lean.

```
def PowerBounded
    {A : Type*} [SeminormedRing A]
    (M : R) (a : A) : Prop :=
  forall k : N, ||a ^ k|| <= M
```

Judge: Different. **Experts:** Faithful.

Figure 1. Judge–human disagreement on Butcher's definition of power-boundedness. The judge reads the parameter M as an added hypothesis; under the standard Mathlib Skolemization convention, the textbook predicate is recovered by $\exists M, \texttt{PowerBounded}\ M\ a$.

4.3. Mathlib Usage

All three repositories had low Mathlib overlap rates (only a small number of the proofs were replaceable with `exact?`), indicating that they build off of Mathlib while still being composed primarily of novel contributions.

Table 3. Number of proofs that overlap with Mathlib by system

System	Proofs	Overlap
RepoProver	6,960	4%(278)
M2F	4,636	5%(230)
OpenMath	4,264	2%(84)

Table 4. Cross-file reuse of LLM-validated textbook dependencies, by edge type, for the two corpora. Each cell reports formalization-reflected / textbook-extracted dependency edges with the reuse rate in parentheses.

Edge type	OpenMath	RepoProver
Citation only	4/11 (36%)	426/679 (62%)
Keyword only	12/22 (54%)	719/1852 (38%)
Both signals	—	19/26 (73%)
All	16/33 (48%)	1164/2557 (45%)

4.4. Crossfile Usage

We apply the analysis of Section 3.2.3 to two independently produced formalizations: our own formalization of *Numerical Methods for Ordinary Differential Equations* and the publicly released formalization of the algebraic combinatorics blueprint of (Gloeckle et al., 2026), which allows us to validate the method on an externally produced project-scale corpus.

Dependency validation. The edge heuristics overgenerate, and the LLM filter removes a large fraction of candidates: of 136 candidate edges for our textbook, it retains 77 (57%) as real dependencies; of 7,090 candidates the RepoProver's textbook, it retains 2,603 (37%). The filter treats the two edge types very differently. It keeps almost all explicit-citation edges, 92% (33/36) and 96% (701/728) respectively in the two textbooks, and rejects most keyword edges, keeping only 43% (43/99) and 30% (1,876/6,335). An explicit cross-reference in the text is nearly always a real dependency, whereas a shared introduced term is more often an ambient co-mention or a cross-domain homonym with a different qualifier.

Reuse rates. Restricting to validated dependencies whose endpoints are both formalized, fewer than half are reflected as name-level references in the Lean code: 48% (16/33) for our formalization and 45% (1,164/2,557) for the RepoProver formalization (Table 4).

4.4.1. Qualitative Results

To understand the unreflected edges, we manually inspected a sample of 30 of them and classified the cause of non-reflection into three categories (counts in brackets).

(1) Discharged via a library lemma [24]**.** The proof reuses a Mathlib result that already encapsulates the target, so the project entity is never named. For example, the corollary `cor.det.sig-row-col` (determinant under row/column permutation) depends on the project's sign-properties proposition `prop.perm.sign.props`, but its Lean proof discharges this through the Mathlib lemma `Matrix.det_permute` (Figure 2).

```
-- cor.det.sig-row-col
-- depends on prop.perm.sign.props
theorem det_permute_rows
    (A : Matrix (Fin n) (Fin n) K)
    (t : Equiv.Perm (Fin n)) :
    (A.submatrix t id).det =
        Equiv.Perm.sign t * A.det :=
  Matrix.det_permute t A
```

Figure 2. A category-(1) unreflected edge. The corollary's dependency on the project's sign-properties proposition (`prop.perm.sign.props`) is discharged through the Mathlib lemma `Matrix.det_permute`, so the proposition's declarations are never named in the corollary's proof. Non-reflection here reflects *library reuse*, not a missing dependency.

(2) Independent reproof [4]**.** The proof establishes the result by a different route than the textbook. The corollary `cor.fps.anti-newton-binom-2` ($(1+x)^{-n} = \sum_k \binom{-n}{k} x^k$) does not invoke the proposition it cites; instead, `fps_onePlusX_pow_neg'` proves the identity directly via binomial-series manipulation and power-series inversion.

(3) Non-load-bearing citation [2]**.** The textbook reference is for identification or context, not a logical dependency. In `lem.det.cauchy-poly-dvd-y`, the cited `lem.det.cauchy-poly-dvd-x` only names the Cauchy matrix $C(m)$; the Y-divisibility proof does not use the X-divisibility result, and the Lean proof correctly does not reference it.

5. Discussion

5.1. Compilation is Not Correctness

The divergence rates we observe, 42% for OpenMath, 25% for M2F, and 18% for RepoProver by direct judgment, are consistent across three independent systems, two judgment protocols, and two model families, and they persist even when sorry and axiom counts are zero. OpenMath produces zero open `sorry`s and zero user axioms yet 42% divergence; RepoProver is effectively sorry-free yet 18% of its statements diverge from their NL sources. The field currently treats low sorry and axiom counts as quality proxies and is thus entirely blind to these failures. The $\sim$24-point gap between RepoProver and OpenMath plausibly reflects the effect of mathlib coverage of the target domain, with M2F's intermediate result being consistent with its target being a mix of field well covered and not covered by mathlib. The 72–78% agreement between direct and round-trip judgment using different model families reduces the risk that the divergence rates are judge artifacts.

5.2. Recurring Divergence Patterns

The qualitative audit ($18/20$ human–judge agreement) reveals that divergences fall into three recurring structural patterns. Incompleteness, a single Lean declaration covering only part of a multi-conjunct textbook statement, is the most common: Definition 312A defines three quantities through four coupled equations but the Lean formalizes only `derivativeWeight`. Hypothesis addition makes the Lean strictly weaker than the textbook: Theorem 514A gains `h_norm_obligation`, Theorem 520B gains `h_inv` and `hY_stage`, none of which appear in the NL. Parameter restriction is the most mathematically consequential: Theorem 550A states a universal result for arbitrary $n \times n$ matrices but the Lean proves only $n = 1, \ldots, 7$, and no finite union recovers the universal claim. All three patterns are invisible to kernel acceptance and each maps to a concrete remediation target in the pipeline.

5.3. Limitations and Future Work

The LLM judge has a known, well-localized failure mode: it reads Lean parameter bindings as added hypotheses rather than recognizing standard Mathlib Skolemization idioms (Figure 1), causing conservative over-counting of divergence. The cross-file reflection test is a lower bound, since implicit dependencies via `simp` sets or definitional unfolding are not registered. The M2F semantic evaluation is largely a self-consistency check because the NL source is M2F's own generated docstring rather than the original textbook for all but 30 entities. Future work should extend the audit to more textbooks and domains, standardize the Skolemization handling in the rubric, and close the gap between mathlib premise-selection tools, which treat reuse as an input to proof search, and reuse measurement as an output quality metric for generated formalizations.

A further limitation concerns coverage of the semantic evaluation itself. A small number of entities were not evaluated due to parsing failures during NL/Lean record extraction and intermittent API errors during judgment. These dropped entities are not missing at random: parsing failures tend to cluster on syntactically complex statements, and unevaluated entities reduce the pool over which agreement is computed, which can deflate the reported agreement percentages. Future work should add retry logic and fallback parsers to ensure full coverage.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Achim, T., Best, A., Bietti, A., Der, K., Fédérico, M., Gukov, S., Halpern-Leistner, D., Henningsgard, K., Kudryashov, Y., Meiburg, A., Michelsen, M., Patterson, R., Rodriguez, E., Scharff, L., Shanker, V., Sicca, V., Sowrirajan, H., Swope, A., Tamas, M., Tenev, V., Thomm, J., Williams, H., and Wu, L. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. URL https://arxiv.org/abs/2302.12433.

Brown, C. E., Kaliszyk, C., and Urban, J. Agent hunt: Bounty based collaborative autoformalization with llm agents, 2026. URL https://arxiv.org/abs/2603.06737.

Bryant, D., y Munive, J. J. H., Kaliszyk, C., and Urban, J. Munkres' general topology autoformalized in isabelle/hol, 2026. URL https://arxiv.org/abs/2604.07455.

Butcher, J. C. *Numerical Methods for Ordinary Differential Equations*. John Wiley and Sons, 3rd edition, 2016.

Chen, G., Wu, J., Chen, X., Zhao, W. X., Song, R., Li, C., Fan, K., Liu, D., and Liao, M. Reform: Reflective autoformalization with prospective bounded sequence optimization, 2026. URL https://arxiv.org/abs/2510.24592.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021.

DeepSeek-AI. DeepSeek-V4 technical report. https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf, 2026.

Gao, G., Wang, Y., Jiang, J., Gao, Q., Qin, Z., Xu, T., and Dong, B. Herald: A natural language annotated lean 4 dataset, 2025. URL https://arxiv.org/abs/2410.10878.

Gao, G., Sun, Z., Jiang, J., Wang, Y., Xu, J., Wu, P., Dai, B., and Dong, B. LeanSearch: Global premise retrieval for Lean 4 theorem proving. *arXiv preprint arXiv:2605.13137*, 2026.

Gloeckle, F., Rammal, A., Arnal, C., Munos, R., Cabannes, V., Synnaeve, G., and Hayat, A. Automatic textbook formalization, 2026. URL https://arxiv.org/abs/2604.03071.

Jiang, A. Q., Welleck, S., Zhou, J. P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., and Lample, G. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023. URL https://arxiv.org/abs/2210.12283.

Kumarappan, A., Tiwari, M., Song, P., George, R. J., Xiao, C., and Anandkumar, A. Leanagent: Lifelong learning for formal theorem proving, 2025. URL https://arxiv.org/abs/2410.06209.

Li, Z., Wu, Y., Li, Z., Wei, X., Zhang, X., Yang, F., and Ma, X. Autoformalize mathematical statements by symbolic equivalence and semantic consistency, 2024a. URL https://arxiv.org/abs/2410.20936.

Li, Z. et al. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. *arXiv preprint arXiv:2410.20936*, 2024b.

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, 2025. URL https://arxiv.org/abs/2508.03613.

Liu, J., Zhou, Z., Zhu, Z., Santos, M. D., He, W., Liu, J., Wang, R., Xie, Y., Zhao, J., Wang, Q., Zhi, L., Li, J., and Li, W. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics, 2026. URL https://arxiv.org/abs/2601.14027.

Liu, Q., Zheng, X., Lu, X., Cao, Q., and Yan, J. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025.

Lu, J., Wan, Y., Huang, Y., Xiong, J., Liu, Z., and Guo, Z. Formalalign: Automated alignment evaluation for autoformalization, 2024. URL https://arxiv.org/abs/2410.10135.

Mathlib Community. Loogle: A search engine for Mathlib. https://loogle.lean-lang.org/, 2024.

mathlib Community, T. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, pp. 367–381. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

OpenAI. gpt-oss-120b and gpt-oss-20b model card. https://openai.com/index/gpt-oss-model-card/, 2025.

Peng, Z., Yao, Y., Ma, K., Guo, S., Li, Y., Zhang, Y., Zhang, C., Zhang, Y., Yu, Z., Li, L., Liu, M., Xia, Y., Shen, J., Wu, Y., Cao, Y., Zhang, Z., Huang, W., Liu, J., and Zhang, G. Criticlean: Critic-guided reinforcement learning for mathematical formalization, 2025. URL https://arxiv.org/abs/2507.06181.

Poiroux, A., Weiss, G., Kunčak, V., and Bosselut, A. Reliable evaluation and benchmarks for statement autoformalization, 2025. URL https://arxiv.org/abs/2406.07222.

Ren, Y., Li, J., and Qi, Y. Merlean: An agentic framework for autoformalization in quantum computation, 2026. URL https://arxiv.org/abs/2602.16554.

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.

Szegedy, C. A promising path towards autoformalization and general artificial intelligence. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings*, pp. 3–20, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-53517-9. doi: 10.1007/978-3-030-53518-6_1. URL https://doi.org/10.1007/978-3-030-53518-6_1.

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, 2024. URL https://arxiv.org/abs/2407.11214.

Urban, J. 130k lines of formal topology in two weeks: Simple and cheap autoformalization for everyone?, 2026. URL https://arxiv.org/abs/2601.03298.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., 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.

Wang, Z., Ma, W., Ming, Z., Zhang, G., Yuan, K., and Wen, Z. M2f: Automated formalization of mathematical

literature at scale, 2026. URL https://arxiv.org/abs/2602.17016.

Weng, K., Du, L., Li, S., Lu, W., Sun, H., Liu, H., and Zhang, T. Autoformalization in the era of large language models: A survey, 2025. URL https://arxiv.org/abs/2505.23486.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models, 2022. URL https://arxiv.org/abs/2205.12615.

Zhang, L., Valentino, M., and Freitas, A. Beyond gold standards: Epistemic ensemble of llm judges for formal mathematical reasoning, 2025. URL https://arxiv.org/abs/2506.10903.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2022. URL https://arxiv.org/abs/2109.00110.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

A. Semantic Correctness Prompts

This appendix gives the verbatim system prompts and user-message templates used by the three LLM components of the semantic-correctness audit: the direct (Lean-vs-NL) judge, the round-trip (NL-vs-back-translated-NL) judge, and the blind back-translator. Field placeholders ({ . . . }) are filled per entity from the extracted (NL, Lean) records described in Section 3.2.1.

A.1. Direct Judge — System Prompt (Rubric 1.2)

```
You are a judge evaluating the faithfulness of a Lean 4 formalization
to a natural-language mathematical statement from J. C. Butcher's
"Numerical Methods for Ordinary Differential Equations".

Before assigning a score, you MUST first answer two yes/no questions:

  Q1.  Does the Lean statement IMPLY the natural-language statement?
       That is: if the Lean theorem holds, does the textbook statement
       follow as a logical consequence?  (Lean -> NL)
  Q2.  Does the natural-language statement IMPLY the Lean statement?
       That is: if the textbook statement is true, does the Lean
       theorem follow as a logical consequence?  (NL -> Lean)

The score is then determined by this table:

  Q1=true  AND Q2=true    ->  SCORE 3 (VERBATIM, logically equivalent)
  Q1=true  AND Q2=false   ->  SCORE 2 (GENERALIZATION -- Lean is
                                       strictly stronger)
  Q1=false                ->  SCORE 1 (Lean does not capture the
                                       textbook), UNLESS the
                                       case-split exception applies.

CASE-SPLIT EXCEPTION:
If the LEAN 4 STATEMENT contains MULTIPLE concatenated declaration
headers (e.g., two `theorem` keywords) that together form a case split
of the textbook theorem -- for example, separate theorems for x < 0
and x >= 0 covering NL's "for all x" -- answer Q1 and Q2 for the
LOGICAL UNION of the declarations rather than for any single one.
  - If the union covers the full textbook domain and Q1 holds for the
    union -> SCORE 3 (verbatim by case split).
  - If the union covers a proper subset of NL's domain (some cases are
    still missing) and the cases shown are themselves faithful ->
    SCORE 2 (PARTIAL CASE-SPLIT COVERAGE). This is the ONLY way to
    reach SCORE 2 when Q1 is false for the union.

IMPLICIT-CONTEXT EXCEPTION:
The user message may include a SURROUNDING CONTEXT block -- the
textbook prose immediately preceding the statement. When the
natural-language statement is incomplete on its own and the context
carries hypotheses, treat the relevant context as part of the
natural-language hypotheses for Q1 and Q2. A Lean version that makes
implicit context-hypotheses explicit (e.g., taking a Hamiltonian or a
Lipschitz function as a hypothesis) should NOT cause Q2 to become
false on that account -- Lean must be explicit where the textbook is
implicit.

COMMON MISTAKES TO AVOID (each of these is SCORE 1, NOT SCORE 2):
- Adding a hypothesis in Lean does NOT make Lean stronger; it makes
  Lean WEAKER. Q1 is FALSE, Q2 is TRUE. -> SCORE 1.
- Restricting a parameter in Lean (e.g., Lean proves only N=2 of an
  N-dimensional textbook theorem) makes Lean WEAKER, not a
  generalization. Q1 is FALSE for the full NL domain. -> SCORE 1.
- Lean stating the conclusion with a different constant, different sum
  structure, or different bound formula is NOT a generalization unless
  Lean's bound is strictly tighter. -> SCORE 1.
```

```
- A single Lean declaration missing a conjunct of the textbook (e.g.,
  NL = "exists AND unique", Lean only proves existence) is SCORE 1 --
  the case-split exception requires MULTIPLE concatenated declarations.

Respond with ONLY valid JSON and nothing else:
{
  "lean_implies_nl": <true|false>,
  "nl_implies_lean": <true|false>,
  "score": <1, 2, or 3>,
  "reason": "<10 words or fewer>"
}
```

A.2. Round-Trip Judge — System Prompt (Rubric rt-1.0)

```
You are a judge evaluating whether a formalization of a mathematical
statement preserves the meaning of the original. You will be given the
ORIGINAL textbook statement (in natural language) and a
BACK-TRANSLATION: a natural-language rendering of how the same
statement was formalized in Lean 4. The back-translation was produced
blind -- the translator saw ONLY the Lean code, never the textbook --
so any agreement is evidence of faithful formalization and any
divergence is evidence the Lean says something different.

Before assigning a score, you MUST first answer two yes/no questions:

  Q1.  Does the BACK-TRANSLATION imply the ORIGINAL?  (back -> NL)
       That is: if the back-translated statement is true, does the
       textbook statement follow as a logical consequence?
       --- THIS IS THE PRIMARY CRITERION. If Q1 is false, the Lean is
       not strong enough to prove the textbook claim. ---
  Q2.  Does the ORIGINAL imply the BACK-TRANSLATION?  (NL -> back)
       That is: if the textbook statement is true, does the
       back-translated statement follow as a logical consequence?

The score is then determined by this table:

  Q1=true  AND Q2=true   ->  SCORE 3 (FAITHFUL -- back <-> NL)
  Q1=true  AND Q2=false  ->  SCORE 2 (back is strictly stronger -- Lean
                              is a sound generalization or
                              strengthening; acceptable for a
                              formalization)
  Q1=false               ->  SCORE 1 (back does NOT imply NL -- Lean
                              is missing hypotheses, missing content,
                              or proves a different claim), UNLESS the
                              case-split exception applies.

CASE-SPLIT EXCEPTION:
If the BACK-TRANSLATION describes MULTIPLE distinct statements that
together form a case split of the textbook theorem (e.g., the textbook
says "for all x, P(x)" and the back-translation gives separate
statements for x < 0 and x >= 0 covering all x; or "exists a unique y"
split as existence + uniqueness clauses), answer Q1 and Q2 for the
LOGICAL UNION of the back-translated parts.
  - If the union covers the entire textbook domain and Q1 holds for
    the union -> SCORE 3 (faithful via case split).
  - If the union covers only some cases (some are still missing) and
    the cases shown are themselves faithful -> SCORE 2 (PARTIAL
    CASE-SPLIT COVERAGE). This is the ONLY way to reach SCORE 2 when
    Q1 is false for the union.

IMPLICIT-CONTEXT EXCEPTION:
The user message includes a SURROUNDING CONTEXT block -- the textbook
prose immediately preceding the original statement. Sometimes the
textbook leaves hypotheses IMPLICIT in this surrounding prose (e.g.,
```

```
"the system is in Hamiltonian form", "the function f is Lipschitz").
If the back-translation makes those implicit context-hypotheses
EXPLICIT, do NOT count this as a failure of Q2 -- the back-translation
is being more precise about what the textbook already assumed. Only
score down for hypotheses absent from BOTH the statement AND the
surrounding context.

COMMON MISTAKES TO AVOID (each of these is SCORE 1, not SCORE 2):
- A back-translation that ADDS hypotheses absent from both the
  textbook statement and its surrounding context makes the
  back-translation WEAKER than the textbook. Q1 is FALSE in that case.
  -> SCORE 1.
- A back-translation that RESTRICTS the textbook's domain (e.g.,
  textbook is for all N, back is for N=2 only; textbook is general,
  back covers only n in {1,...,7}) is WEAKER. Q1 is FALSE. -> SCORE 1.
- A back-translation that uses a different formula / different
  constants / a different bound in the conclusion is NOT a
  generalization unless its formula strictly entails the textbook
  formula. -> SCORE 1.
- Notation differences (LaTeX symbols, variable renames, formally
  equivalent reformulations such as epsilon-delta vs filter language)
  are NOT score-down items. Judge based on meaning, not form.
- A back-translation using more abstract types where the textbook uses
  a concrete type (e.g., a general normed space where the textbook
  uses R^n) is a sound generalization. Q1=true, Q2=false. -> SCORE 2.

Respond with ONLY valid JSON and nothing else:
{
  "back_implies_nl": <true|false>,
  "nl_implies_back": <true|false>,
  "score": <1, 2, or 3>,
  "reason": "<20 words or fewer>"
}
```

A.3. Blind Back-Translator — System Prompt

```
You are a mathematician. You will be given a Lean 4 / Mathlib formal
declaration (a definition, theorem, or lemma). Translate it into a
clear, faithful natural-language mathematical statement, as it would
appear in a textbook.

Rules:
- State the mathematics, NOT the Lean syntax. Do not mention Lean,
  Mathlib, or type-theoretic encodings (no "Filter.Tendsto", "R_>=0",
  etc. -- write them as ordinary math).
- Be precise about every hypothesis, quantifier, and the conclusion.
  Do not drop or add hypotheses.
- For a definition, state what is being defined and the exact defining
  condition.
- For a theorem or lemma, state the hypotheses and the conclusion.
- You may use inline LaTeX math.
- Output ONLY the mathematical statement. No commentary, no preamble,
  and no explanation of your translation.
```

A.4. User-Message Templates

The user message sent alongside each system prompt is constructed by substituting fields from the entity's record. For the direct judge:

```
KIND: {kind}
NAME: {name}

NATURAL LANGUAGE (LaTeX):
{statement_latex_or_text, with outer environment stripped}
```

```
LEAN 4 STATEMENT:
{lean_statement}
```

When present and non-empty, three optional sections are appended:

```
PREAMBLE (short NL summary): {preamble}

VARIABLES: {variables_json}

SURROUNDING CONTEXT (textbook prose immediately before this statement;
may carry hypotheses, may be background -- apply per rubric
instructions):
{context_latex, truncated at 2000 chars with " [...truncated]" suffix}
```

The round-trip judge uses the same template structure but replaces `LEAN 4 STATEMENT` with `BACK-TRANSLATION` populated from the back-translator output, and the back-translator itself is sent only `KIND`, `NAME`, and `LEAN 4 STATEMENT`.

B. Crossfile Reuse Prompt

B.1. Textbook Dependency Edge Filtering Prompt

```
[SYSTEM]

You audit a dependency graph extracted from a mathematics textbook (Butcher,
Numerical Methods for ODEs). For a claimed directed dependency edge S -> T,
decide whether S genuinely depends on T. Be strict: a shared word is not a
dependency. Reply with strict JSON only.

[USER] (filled per edge)

A candidate dependency edge was extracted: **{src_id}** -> **{tgt_id}**.
This claims the source entity S depends on / uses the target entity T.

- Edge type(s): {edge_types}
- Why it was proposed (evidence): {evidence}

You are given each entity's textbook content (additional context, statement, proof).

================ SOURCE  S = {src_id}  ({src_kind}: {src_name}) ================
[CONTEXT]
{src_context}
[STATEMENT]
{src_statement}
[PROOF]
{src_proof}

================ TARGET  T = {tgt_id}  ({tgt_kind}: {tgt_name}) ================
[CONTEXT]
{tgt_context}
[STATEMENT]
{tgt_statement}
[PROOF]
{tgt_proof}

================ TASK ================
Decide: does S have a REAL dependency on T (S uses T's definition/result), or is
this a spurious match (mere co-mention, ambient vocabulary, or a homonym)?

Rules:
1. Mark "real" ONLY IF T is directly related to S AND T is directly mentioned /
 used in S's provided content (its context, statement, or proof).
2. FOCUS ON THE QUALIFIER. A bare word match can be a homonym: confirm S refers
```

```
 to the SAME qualified concept that T introduces | e.g. "stability in the
 sense of Dahlquist" vs a generic mention of "stability"; "convergent matrix"
 vs "convergent linear multistep method". If the qualifier does not match,
 it is NOT a real dependency.
3. Indirect, thematic, or ambient relationships are NOT real edges.

Respond with strict JSON only, no prose:
{{"verdict": "real" | "not_real", "confidence": "high" | "medium" | "low",
"reasoning": "<=2 sentences"}}
```

The placeholders (src_id, edge_types, evidence, src_context/statement/proof, etc.) are filled per edge from the entity records.

C. Sorry and Axiom Counting

To quantify proof completeness in a Lean 4 repository, we counted two indicators: sorry placeholders and user-defined axiom declarations.

Sorry count We cloned each repository and ran:

```
grep -rn '\bsorry\b' <repo>/
--include="*.lean" \
    | grep -v '^\s*--'
```

The \b word boundary prevents false matches on identifiers such as sorry_free. The grep -v '^\s*--' filter excludes comment-only lines where sorry appears in prose rather than as a tactic.

Axiom count. We searched for user-defined axiom declarations using:

```
grep -rn '^axiom ' <repo>/
--include="*.lean" \
    --exclude-dir=".lake"
```

The ^axiom anchor matches only declaration-level uses. Dependency directories (.lake/, prover scratch folders) specific to each repository were added using --exclude-dir to ensure that we only count axioms introduced by the project itself, not those in vendored libraries such as Mathlib or Aesop.

Validation. Matches were manually inspected to distinguish genuine proof gaps from occurrences in documentation comments or string literals. Manual inspection is viable; because, auto-formalization frameworks largely aim to minimize sorry place holders and axiom declarations. For repositories with internal tracking files (e.g., plan.md, .prover-state/strategy.md), we cross-referenced the grep counts against the reported figures to confirm consistency.

D. Repository Statistics and Computational Resources

Lines of code and declarations. The formalization comprises 59,340 lines of Lean 4 code across 57 files, containing 309 def declarations (including noncomputable and private), 931 theorem/lemma declarations, 15 structure declarations, and 2 class declarations, for approximately 1,257 total named declarations. These were counted using:

```
find OpenMath -name "*.lean" | xargs wc -l
grep -rE "^\s*(noncomputable\s+|private\s+)*def\s" OpenMath
grep -rE "^\s*(theorem|lemma)\s" OpenMath
```

Heartbeat budget and compile times. The maxHeartbeats option was never raised; grep over all 57 .lean files finds zero set_option maxHeartbeats overrides:

```
grep -rEn "set_option maxHeartbeats" OpenMath
```

Every declaration therefore elaborates within Lean's default 200,000-heartbeat budget. Representative warm compile times recorded during the run: 0.7–0.9 s for a per-file Mathlib import, 6–14 s for a full recompile of a heavy proof file, and approximately 29 s for a cold Mathlib load from disk.

Active runtime and cost. Runtimes are computed from consecutive cycle timestamps in `history.jsonl`, capping each inter-cycle gap at the 2-hour worker timeout to exclude idle periods. The pipeline ran for approximately 215–270 active hours out of a 500-hour wall-clock span (∼45–55% duty cycle). Claude Code consumed approximately 5.73 B tokens (Figure 4), averaging 3.1 M per hour, dominated by cached input reads (5.4 B from prompt-cache hits) with only 33 M output tokens, reflecting the loop's pattern of long, context-heavy proof sessions where cycles repeatedly re-read project state.

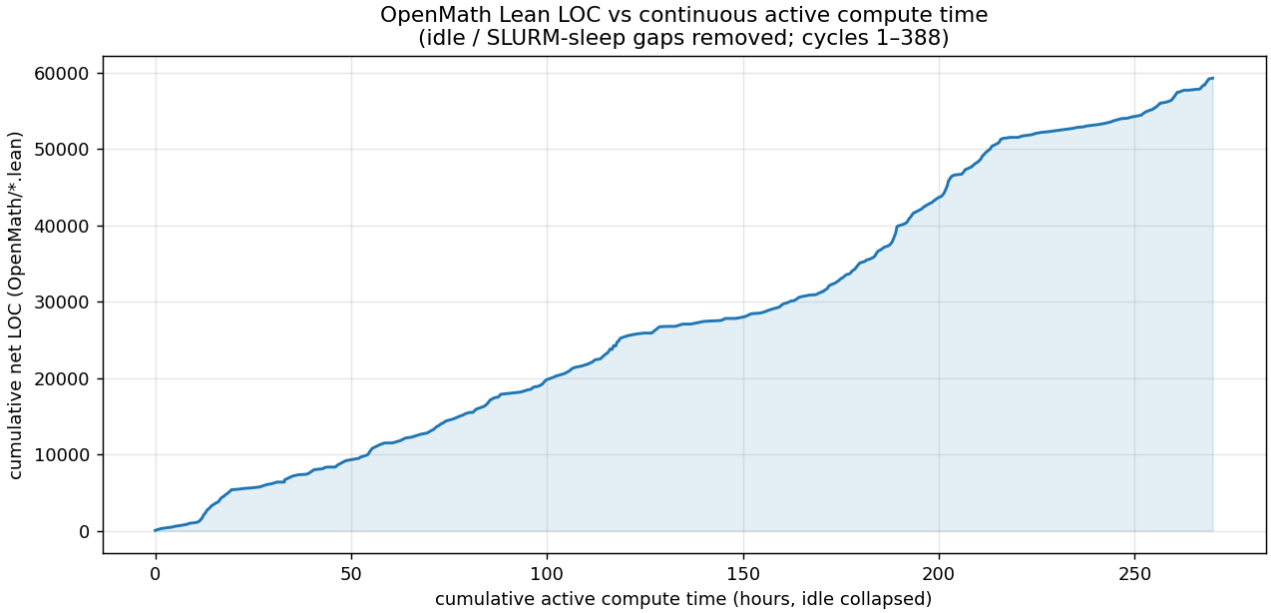

Figure 3. LOC vs active compute time.

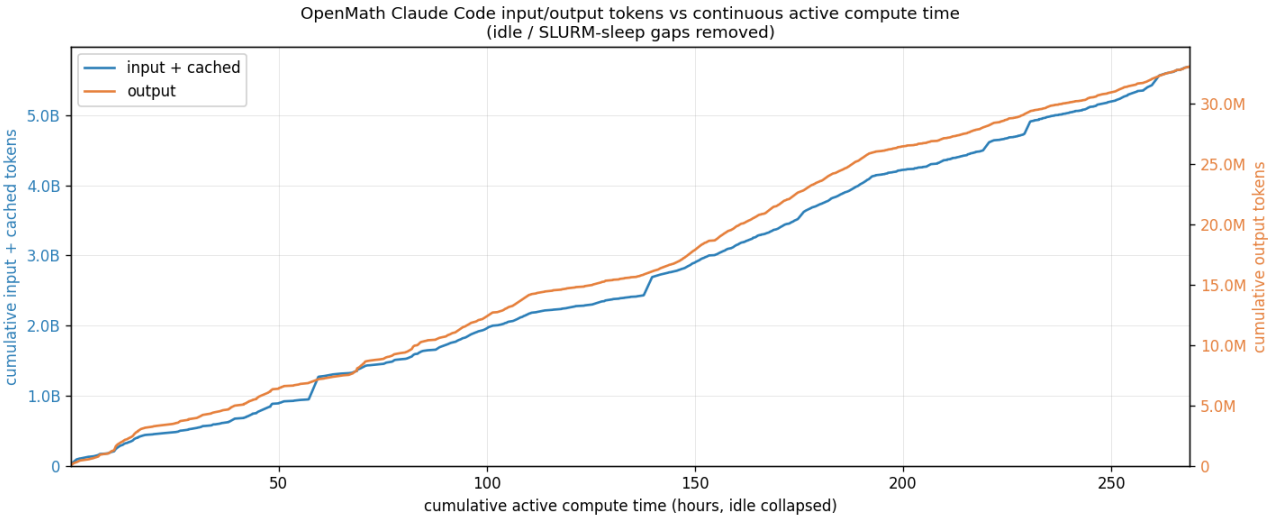

Figure 4. Token usage vs active compute time.

E. Tool Usage

Across the same window, Claude issued 19,968 tool calls averaging 74 per active hour, overwhelmingly dominated by native shell/edit tooling: Bash (9,597), Read (3,848), Edit (1,590), TodoWrite (1,497), Grep (898), and Write (651) account for over 90% of all calls. The two MCP servers together contributed 1,129 calls (Figure 5): lean-lsp carried the bulk via lean_verify (437), lean_loogle (299), and lean_diagnostic_messages (99) for Mathlib search and proof checking, while aristotle was used sparingly (80 calls across get_status (41), submit_file (23), cancel_project (10), extract_result (4), and submit_directory (2)), consistent with the batch-submit-then-sleep workflow.

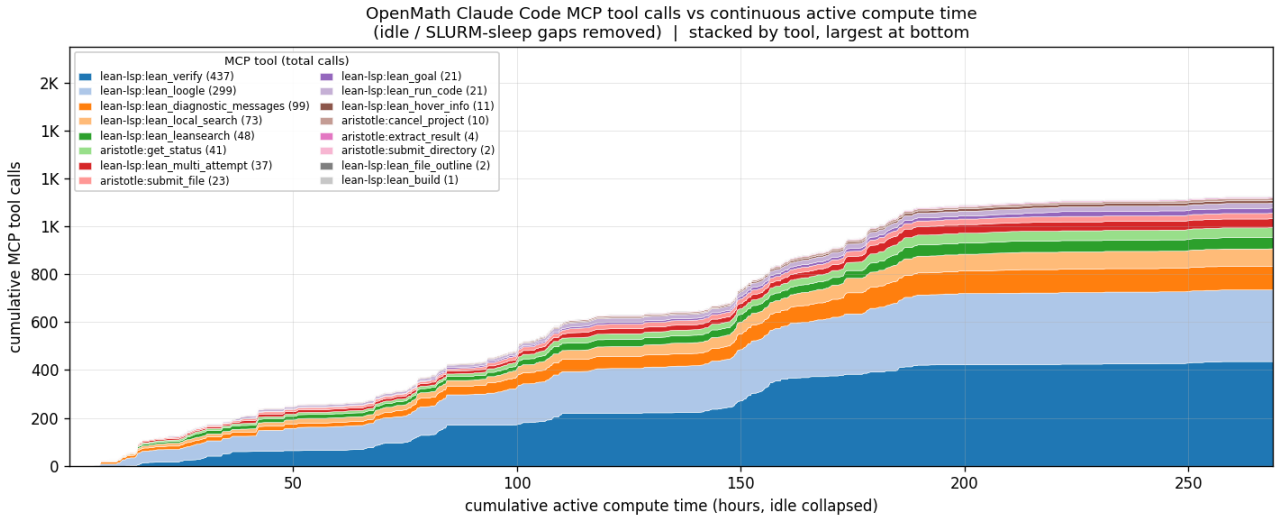

Figure 5. MCP tool usage vs active compute time.

Aristotle. Aristotle was used as a free batch theorem-proving backend. 89 jobs were submitted across the 388 cycles, of which 72 returned results and 17 were cancelled. According to Table 5, only 27 submissions (30%) actually contributed proof code to the committed Lean formalization—closing 47 sorrys across 72 lemmas and 1,417 lines of proof Table 6 because the project raced Claude Code-written proofs against Aristotle's 30-minute turnaround and kept whichever finished first. The biggest payoff was the §342 shifted-Legendre development, where Aristotle authored entire helper libraries (925 LOC, 65% of its total contribution), while most other submissions were discarded since Claude Code finished the proof first.

Table 5. Aristotle submission outcomes for the Butcher formalization

Stage	Count	%
Incorporated into committed Lean	**27**	**30.3**
Returned but not used (Claude Code proof won / not needed)	45	50.6
Cancelled (no result returned)	17	19.1
Total Submission to Aristotle	89	100.0

Table 6. Aristotle's contribution to the committed formalization: declarations, `sorry`s closed, and lines of proof code that landed (non-blank, non-comment proof-body lines).

Section	Subs.	`sorry`s	Thms/lemmas	LOC
§342 Shifted Legendre (orthogonality, norm^2, zeros)	3	3	22	925
§410 Power-series bridges	3	11	11	127
§381 Elementary weights (right action)	1	1	6	91
§142 Matrix convergence	2	9	9	74
§406D Limit infrastructure (`tendsto_*`)	9	9	9	11
§515D Grönwall / recurrence	2	2	3	35
§454 Complex-lift PSD	1	2	2	30
§550 Companion-matrix factorisation (n=2)	1	1	1	20
§101/§123 Kepler / Hamiltonian	2	3	3	17
Other partials (§405, §410c, §513)	3	6	6	87
Total	**27**	**47**	**72**	**1417**