Designing an Agentic Tool for AI-Assisted Mathematical Proofs

Integrating Lessons from the “First Proof” Benchmark, the Erdős/Aletheia Case Study, and Kirov’s Workflow

Dietmar Wolz

12 February 2026

Abstract

This document proposes a production-oriented architecture for an agentic tool. It is designed to support AI-assisted mathematical proof development at the research level. The design draws on two complementary sources of insight. First, we analyze Google DeepMind’s Aletheia project, which evaluated 700 Erdős conjectures. This study revealed critical failure modes, including problem misinterpretation and literature identification. Second, we incorporate practical experience from the “First Proof” benchmark [1]. In that study, a structured multi-agent workflow produced proofs of varying quality for ten previously unpublished problems. Based on these findings, we propose a multi-layered orchestration system. This system assigns specialized roles to frontier models based on their observed strengths and incorporates formal verification (Lean) not just as a checker, but as a “gap-finding oracle.” Crucially, the architecture enforces strict statement-proof separation, maintains explicit human-in-the-loop checkpoints, and utilizes iterative “pitfall registries” to prevent recurring formalization errors.

Keywords: math proofs, change budget, multi-AI orchestration, human control, automated reasoning, Lean formalization, pitfall registry.

Contents

1 Introduction

Mathematical proof is among the most demanding cognitive tasks AI systems can attempt. Unlike code generation, where correctness can often be assessed by running tests, mathematical proofs require multi-modal verification: syntactic well-formedness, logical soundness, semantic meaningfulness, and novelty relative to existing literature. Two recent large-scale efforts have illuminated the landscape of challenges.

The Aletheia/Erdős study. Google DeepMind's systematic evaluation of 700 Erdős conjectures [2] revealed a sobering pipeline:

```
700 problems → 200 candidates (AI verifier) → 63 correct → 13 meaningful
```

The dominant failure mode was not logical error but *problem misinterpretation*: 50 of 63 technically correct solutions solved the wrong problem. Literature identification was "the lengthiest and most arduous step," and the risk of "subconscious plagiarism" from training data posed additional concerns.

The "First Proof" benchmark. The "First Proof" benchmark [1] presents ten research–level questions across diverse mathematical fields—from stochastic analysis and symplectic geometry to spectral graph theory and tensor analysis—whose answers had never appeared publicly. By applying a structured multi-agent workflow using predefined prompts (without injecting domain-specific proof steps), we obtained proofs of varying quality for all ten questions. This experience provides practical evidence about which agent roles, prompt strategies, and iteration patterns are effective for research–level mathematics.

This document synthesizes these two sources into a unified architecture. We address both the theoretical requirements (interpretation, literature, verification, novelty) identified by Aletheia and the practical workflow patterns (computational experimentation, meta-prompted synthesis, adversarial review loops) validated by "First Proof".

2 Key Insights and Design Drivers

2.1 From the Aletheia/Erdős Study

1. **Problem misinterpretation dominates.** 68.5% of solutions were fundamentally flawed; of those technically correct, 79% misinterpreted the problem (wrong conventions, trivial interpretations).

2. **Literature search is the bottleneck.** Identifying whether a problem was already solved—sometimes by an "offhand remark" in an unrelated paper—was harder and more time-consuming than generating proofs.

3. **Formal verification is necessary but insufficient.** Harmonic's Aristotle [4] generated formally verified Lean proofs that solved *weaker variants* of the intended problem. Formal correctness does not imply mathematical meaningfulness.

4. **Plagiarism risk is real.** AI systems may reconstruct known proofs from training data without attribution, requiring explicit novelty checking.

2.2 From the "First Proof" Experience

Applying a multi-agent workflow [5] to all ten "First Proof" problems yielded the following practical insights:

1. **Heterogeneous specialization helps (capability-first).** Across the "First Proof" workflow runs, assigning distinct *capability roles* (e.g., experimentation/code optimization, proof planning/LaTeX synthesis, adversarial logical review) improved overall quality compared to running a single agent end-to-end.

2. **Computational experimentation is a critical first step.** For problems involving numerical objects, running computational experiments before attempting proofs both guides the proof strategy and catches false conjectures early.

3. **Meta-prompting dramatically improves proof quality.** Rather than asking a model to "prove X," first asking it to create a *plan* for the proof and then executing that plan produces substantially more coherent results.

4. **Iterative adversarial review is essential.** A single-pass proof is rarely correct. The review loop—where one agent critiques and another revises—catches logical gaps. Early research by Althöfer [3] on human-machine collaboration demonstrated that providing "multiple computer hints" for human selection significantly improves performance; our workflow adapts this by having the Review Agent provide specific "hints" (replacements) for the Synthesis Agent (or human) to accept or reject.

5. **Concrete replacement text accelerates convergence.** Reviews that merely identify problems are less effective than reviews that propose specific replacement text.

6. **A separate code review loop improves experimental reliability.** Computational experiments benefit from their own review cycle to catch numerical bugs before empirical evidence informs proof strategy.

2.3 From AI-Assisted Formalization Practice

Three emerging case studies inform the design of our Formalization Bridge and the broader human-AI collaboration model.

The CLAUDE.md-as-style-guide pattern. Kirov's workflow [12] demonstrates that agent performance improves when rules are accumulated iteratively. Instead of a static prompt, rules ("Style Guide") and errors ("Pitfall Registry") are recorded in living documents (`CLAUDE.md`, `TACTICS.md`). Our architecture adopts this via the **Definition Lockfile** and **Context Bundles**, evolving them through the review process.

Knowledge delegation taxonomy. We distinguish tasks based on the AI's comparative advantage [13]:

- **Basic language/Syntax:** Delegate after bootstrapping.
- **Library Search:** Always delegate; pure upside.
- **Proof Engineering/Decomposition:** Human retains; AI assists.
- **Mathematical Ideas/Strategy:** Human retains; AI proposes candidates.

Due diligence for AI-generated proofs. Tao's checklist [8] warns that proofs which are "suspiciously short" or "prove significantly more than asked" often solve a misstated problem. This necessitates a strict separation between the agent generating the proof and the agent (or human) verifying the statement.

3 Empirical Grounding, Evaluation Protocol, and Reporting

The architectural choices in this document are motivated by two case studies (Aletheia/Erdős and "First Proof"). To avoid over-generalizing from limited trials, we distinguish:

- **Observed in case studies:** behaviors seen in a specific workflow run.
- **Design hypothesis:** an architectural claim suggested by those observations.
- **Evaluation protocol:** a reproducible procedure that can confirm or refute the hypothesis.

3.1 Key outcome metrics

We recommend reporting results using a fixed rubric:

- **Interpretation accuracy:** Did the final proof address the intended statement?
- **Logical soundness:** Are there unresolved gaps after adversarial review?
- **Meaningfulness:** Does the solution avoid trivialization or weaker variants?
- **Novelty:** Is the result new relative to known literature (best-effort check)?
- **Reproducibility:** Can a third party rerun experiments and regenerate drafts/reviews?

3.2 Minimal reporting bundle (per problem)

1. **Interpretation dossier** (Layer 1 output): Definitions, conventions, alternatives.
2. **Literature dossier** (Layer 2 output): Queries issued, candidate papers, decision notes.
3. **Experiment bundle** (Layer 3 output, if used): Code, seeds, parameters, raw outputs.
4. **Proof trace** (Layer 4 output): $P_0, R_0, P_1, R_1, \ldots$ with accept/reject decisions.
5. **Post-verification report** (Layer 5 output): Interpretation match, novelty check, remaining risks.

3.3 Benchmarking, Ablations, and Regression Tests

To keep the system honest under toolchain drift, we evaluate it on a fixed, versioned benchmark suite and publish ablation results (removing components) to quantify each layer's marginal value.

3.3.1 Benchmark suite design

Maintain a suite with diverse problem types and difficulty tiers:

- **Interpretation-stress** problems (many conventions; easy to solve the wrong thing).
- **Literature-heavy** problems (likely already known; subsumption is subtle).
- **Experiment-friendly** problems (finite objects; counterexamples exist).

- **Formalization-friendly** problems (algebraic/combinatorial fragments).
- **Formalization-hostile** problems (analysis/geometry with heavy background).

Each benchmark item includes an expert-adjudicated label set: intended interpretation, key dependencies, and a "known/unknown" status at evaluation time.

3.3.2 Metrics (quality + cost)

Report the rubric metrics (interpretation, soundness, meaningfulness, novelty, reproducibility) *and* resource metrics:

- wall-clock time, token usage per layer, number of review iterations,
- number of retrieval queries, documents screened, and citations used,
- experiment runtime and compute (CPU-hours/GPU-hours), and
- **human** time at checkpoints (minutes).

3.3.3 Ablations

For each benchmark item, run controlled ablations to quantify the marginal value (and cost) of each component:

- Full system (all layers enabled)
- Remove Layer 1 semantic unit tests
- Remove Layer 2 post-solve pass
- Remove Layer 3 experimentation (if applicable)
- Remove change budgets in the review loop
- Remove the claim-ledger hard gate
- Remove the formalization bridge (if applicable)

Publish deltas in both quality and cost; treat large cost increases as a design bug unless accompanied by a measurable reliability gain.

3.3.4 Regression tests

Promote "semantic unit tests" (Layer 1) and "artifact validity checks" (hashes, env.lock, compilation) into automated regression tests that run on every toolchain update.

4 Artifact Formats, Versioning, and Storage Layout

To make the workflow reproducible and auditable, every inter-layer hand-off is an *artifact* with a machine-readable header and a stable storage layout. The rule is:

> **No artifact, no progress.** A later layer may not proceed unless it can reference the exact artifact version (and hash) it depends on.

4.1 Canonical run directory

Each problem run is stored as a content-addressed directory:

```
runs/<problem_id>/<run_id>/
  meta.yaml
  01_interpretation/
    dossier_v1.md
    dossier_v1.meta.json
    unit_tests/
  02_literature/
    literature_v1.md
    queries_v1.jsonl
    decisions_v1.md
  03_experiments/
    env.lock
    code/
    data/
    report_v1.md
    report_v1.meta.json
  04_proof/
    P0.tex
    R0.md
    P1.tex
    R1.md
    ...
  05_post/
    assessment_v1.md
    claim_ledger_v1.csv
    claim_ledger_v1.meta.json
  06_learning/          <-- New: Post-session extraction
    pitfalls_v1.md      (Registry of formalization errors)
    style_rules_v1.md   (Learned constraints)
```

4.2 Artifact header schema (minimal)

Every `*.meta.json` file contains:

- `artifact_type`: one of {interpretation_dossier, literature_dossier, experiment_report, proof_draft, review, claim_ledger, assessment, pitfall_registry, style_rules}
- `problem_id`, `run_id`, `artifact_version`
- `created_at`, `created_by`
- `inputs`: list of dependency artifacts (type/version/sha256)
- `sha256`: hash of the payload
- `repro`: pointers to `env.lock`, seeds, and command lines

4.3 Versioning rules

1. Layer 1 outputs are **immutable**: later layers must not edit `dossier_v1` but instead create `dossier_v2` with a decision note and re-run unit tests.
2. Any change to definitions/conventions forces a **dossier version bump** and invalidates downstream artifacts until revalidated.

3. Every proof draft and review round is preserved: $P_0, R_0, P_1, R_1, \ldots$.

This contract makes the audit trail robust even when models change and “thinking traces” are unavailable.

5 Architecture Overview

The system consists of five layers, each addressing a distinct class of challenges. An **Orchestrator** coordinates all layers, manages the flow of information between specialized agents, and ensures **human** oversight.

```
+====================================================================+
|  MATH PROOF ORCHESTRATOR                                           |
+====================================================================+
|                                                                    |
|  Layer 1: PROBLEM UNDERSTANDING & DISAMBIGUATION                   |
|    * Parse statement, identify conventions, flag ambiguities       |
|    * Output: Versioned Interpretation Dossier                      |
|    * HUMAN CHECKPOINT: confirm interpretation                      |
|                                                                    |
|  Layer 2: LITERATURE & CONTEXT                                     |
|    * Search for existing solutions (semantic, not just keyword)    |
|    * Build theorem dependency graph                                |
|                                                                    |
|  Layer 3: COMPUTATIONAL EXPERIMENTATION                            |
|    * Numerical simulation and counterexample search                |
|    * Code review loop (separate from proof review)                 |
|                                                                    |
|  Layer 4: PROOF GENERATION & ADVERSARIAL REFINEMENT                |
|    * Meta-prompted proof planning                                  |
|    * Iterative review loop with "fixed change budgets"             |
|    * Formalization Bridge: Statement-Proof Separation              |
|                                                                    |
|  Layer 5: POST-VERIFICATION & NOVELTY                              |
|    * Does proof match intended interpretation?                     |
|    * Semantic similarity to known proofs (plagiarism detection)    |
|    * Post-run learning extraction (Pitfalls/Style)                 |
|    * HUMAN EXPERT REVIEW CHECKPOINT                                |
|                                                                    |
+====================================================================+
```

6 Agent Roles and Specialization

6.1 The Experimentalist (Computational Agent)

Role: Computational Agent

Primary responsibilities: Write and optimize experimental code for numerical exploration; search for counterexamples. **Output:** Code, numerical results, visualizations, proof sketch.

6.2 The Architect (Synthesis Agent)

Role: Synthesis Agent

Primary responsibilities: Create meta-prompted proof plans; execute plans to produce formal LaTeX proofs; revise proofs in response to review feedback. **Output:** Structured proof plan, complete LaTeX proof, revised versions.

6.3 The Reviewer (Adversarial Agent)

Role: Review Agent

Primary responsibilities: Scrutinize proofs for logical leaps and errors; propose concrete replacement text (hints) for the Synthesis Agent or human to select. **Output:** Structured review in LaTeX with proposed corrections.

6.4 The Orchestrator

Role: Orchestrator

Primary responsibilities:

- Classify problems and determine which pipeline branches to activate
- Manage human checkpoints (interpretation, strategy, final review)
- Track proof state: definitions in scope, what is proven vs. assumed vs. cited
- Enforce termination criteria for review loops

6.4.1 Orchestrator Policy: Routing, Budgets, and Termination

We recommend representing policy decisions with three controls: risk level ρ (low/medium/high), confidence $c \in [0, 1]$, and an explicit budget B. Here B caps tokens, wall-clock time, and experiment compute.

Blocking vs. non-blocking review findings. A review item is **blocking** if it is (a) a missing lemma needed for correctness, (b) a circular argument, (c) an unjustified inference step that cannot be replaced by a citation, or (d) a mismatch with the Layer 1 definition lockfile.

Termination and escalation. Terminate only if: unit tests pass, no blocking issues remain, and the claim ledger has evidence links for all nontrivial claims; otherwise revise or escalate to a **human**.

Algorithm 1 Orchestrator policy loop (routing + budgets)

```
1: Activate Layer 2 (literature). If existing solution found, switch to verification.
2: if experimentation trigger holds and budget allows then
3:     Activate Layer 3 (experiments) and require env.lock + seeds.
4: end if
5: Draft proof via Layer 4 (plan $\to$ draft).
6: for $i = 0$ to $max\_iterations$ do
7:     Run targeted review (budget=3).
8:     Update claim ledger; missing evidence links are blocking.
9:     if no blocking issues and unit tests pass then
10:        Compute confidence and either finalize or escalate to human.
11:    else
12:        Revise with explicit accept/reject for each review point.
13:    end if
14: end for
```

Confidence computation and calibration. The confidence score c is a *derived* quantity from multiple partially independent checks; it is not a free-form model self-assessment.

Let the following binary/graded signals be recorded in the Post-verification report:

$$\begin{aligned}
s_{\text{int}} &\in \{0,1\} && \text{(interpretation unit tests pass)},\\
s_{\text{rev}} &\in [0,1] && \text{(review convergence)},\\
s_{\text{exp}} &\in [0,1] && \text{(experiment coverage)},\\
s_{\text{form}} &\in \{0,1\} && \text{(formalization success)},\\
s_{\text{lit}} &\in [0,1] && \text{(literature coverage)},\\
s_{\text{axm}} &\in \{0,1\} && \text{(axiom audit passed: standard axioms only)}.
\end{aligned}$$

A simple baseline aggregation is a logistic model

$$c \;=\; \sigma\Big(w_0 + w_{\text{int}}s_{\text{int}} + w_{\text{rev}}s_{\text{rev}} + w_{\text{exp}}s_{\text{exp}} + w_{\text{form}}s_{\text{form}} + w_{\text{lit}}s_{\text{lit}} + w_{\text{axm}}s_{\text{axm}}\Big),$$

where $\sigma(x) = (1+e^{-x})^{-1}$. The weights w should be calibrated on a held-out set of problems with known outcomes (e.g. curated historical problems, synthetic variants, or subsets with expert adjudication), and recalibrated whenever the model/tool stack changes. At minimum, the system should report an empirical reliability curve (calibration plot) over past runs and interpret c as *uncalibrated* until such a curve exists. In the uncalibrated regime, c must be truncated conservatively (e.g. cap at 0.9) and escalation to a **human** checkpoint should be the default.

7 Detailed Layer Specifications

7.1 Layer 1: Problem Understanding & Disambiguation

7.1.1 Layer 1 Deliverable: Interpretation Dossier (versioned artifact)

Layer 1 should output a single, versioned *Interpretation Dossier* that becomes an immutable input. It should include:

1. **Canonical statement:** a cleaned statement with all symbols bound.
2. **Definition lockfile:** a table of definitions and convention choices.
3. **Semantic unit tests:** small checkable consequences (edge cases, sanity bounds) that would fail under common misreadings.
4. **Decision record:** why the chosen interpretation is believed intended.

7.2 Layer 2: Literature Search & Context

Layer 2 produces a *Literature Dossier* that is sufficient for (a) verifying whether the target statement is already known (possibly as a corollary of a general theorem), and (b) extracting a dependency graph of tools that are permitted to be used downstream.

```
+-----------------------------------------------------+
|  LITERATURE MODULE                                  |
+-----------------------------------------------------+
|  BEFORE solving:                                    |
|  * Search for existing solutions (semantic search)  |
|  * Search for closely related results               |
|  * Build dependency graph of relevant theorems      |
|  * Identify useful techniques from related proofs   |
|                                                     |
|  AFTER generating solution:                         |
|  * Compare solution structure to known proofs       |
|  * Flag potential "subconscious plagiarism"         |
|  * Check if result is a corollary of existing work  |
+-----------------------------------------------------+
```

7.2.1 Layer 2 Deliverable: Literature Dossier (versioned artifact)

The dossier must include:

1. **Query log:** all queries issued (timestamped), including reformulations derived from the Interpretation Dossier (symbols, equivalent definitions, standard names).
2. **Candidate set:** a table of candidate sources with a short relevance note and confidence.
3. **Claim coverage matrix:** for each candidate, which parts of the target claim it appears to cover, and which hypotheses it requires.
4. **Extraction notes:** the precise theorem/lemma statements to be relied on, with *hypothesis checks* against the Interpretation Dossier.
5. **Decision note:** one of {*known*, *probably known*, *unclear*, *probably novel*}, with justification and the remaining uncertainty.

7.2.2 Protocol: two-pass literature search

We run literature search twice:

- **Pre-solve pass:** identify prior art and likely techniques before investing in proof generation.
- **Post-solve pass:** re-run search using the *structure* of the obtained proof (keywords from intermediate lemmas, named techniques, normal forms), which often reveals subsumption results.

Algorithm 2 Literature dossier construction (Layer 2)

```
1: Input: Interpretation Dossier $D$ (canonical statement + definition lockfile)
2: Generate query set $Q$ from $D$ (symbols, synonyms, standard names, equivalent formulations).
3: Retrieve candidate documents; record a timestamped query log.
4: Screen candidates; extract theorem statements; build a dependency DAG.
5: for each extracted theorem do
6:     Check hypotheses against $D$; if unclear, mark as unverified dependency.
7: end for
8: Output Literature Dossier: candidate table + coverage matrix + decision note.
```

7.3 Layer 3: Computational Experimentation

This layer activates for problems involving numerical objects. It includes a separate code review loop to ensure that empirical evidence is trustworthy.

7.4 Layer 4: Proof Generation & Adversarial Refinement

7.4.1 Stage A: Meta-Prompted Proof Synthesis

Two-phase prompting: first plan (identify lemmas, dependencies, logical order), then execute the plan to produce the proof draft.

7.4.2 Stage B: Adversarial Review Loop

The loop alternates between adversarial review and targeted revision.

7.4.3 The Concept of Fixed Change Budgets

To prevent "drift"—where correcting one error introduces two new ones—we enforce a **fixed change budget** (e.g., maximum 3 high-impact changes per iteration). This constraint forces the Review Agent to prioritize the most critical logical gaps rather than nitpicking style. This approach draws on early work by Althöfer [3] on human-machine collaboration in games (2004), which showed that having a computer generate "multiple hints" (candidate moves/changes) for a **human** to select was more effective than a single "best" move. In our workflow, the Review Agent proposes specific replacement candidates (hints), and the Synthesis Agent (or the **human** orchestrator) explicitly selects which to accept.

7.4.4 Prompt Templates

Review Agent: "Propose the three most important improvements... provide proposed replacement text." **Synthesis Agent:** "Check the review... explicitly state for each point whether you accept, reject, or modify."

7.4.5 Layer 4d: The Formalization Bridge (Expanded)

This layer translates local proof slices into Lean/Coq. To ensure validity, we enforce strict separation and validation protocols.

Statement-proof separation. The *statement* of the result to be formalized should be produced or verified by a human (or an independent third party), not by the same AI system generating the proof [8]. This guards against the dominant Lean misformalization risk: an AI proving a subtly weaker or different statement than intended.

Lean validation checklist. Following best practices from the Lean community [9, 10], the Formalization Packet must include a validation report confirming:

1. The proof resides in a proper Lean project with pinned `lean-toolchain`.
2. `lake build` completes without errors.
3. `#print axioms` returns only the standard three axioms (`propext`, `Classical.choice`, `Quot.sound`) — no `sorry`, no user-defined axioms, no malicious metaprogramming.
4. A human has confirmed the *statement* matches the Interpretation Dossier.

Pitfall registry. Following Kirov's practice of maintaining a living `TACTICS.md` file [11], the Formalization Bridge maintains a *pitfall registry*: a versioned list of recurring formalization errors (e.g., tactic misuse, type mismatches) discovered across runs. This registry is included in the Context Bundle for formalization tasks.

Round-trip protocol. If formalization fails, we do not simply record "failed". Instead, we extract the minimal missing obligations (type mismatch, missing lemma, incorrect quantifiers) and feed them back as **blocking** review items in the next change-budget iteration. This turns Lean into a gap-finding oracle even when full verification is infeasible.

Algorithm 3 Formalization bridge (slice-level)

```
1: Input: Context Bundle (including Pitfall Registry), proof slice $P_{\text{slice}}$
2: Human/Orchestrator validates formal Statement $S$ against Dossier.
3: Agent generates Proof $P$ for $S$.
4: Run safety scan (Axiom Check, Sorry Check, Unsafe Check).
5: if verified then
6:     Record evidence pointer in Claim Ledger (formal).
7: else
8:     Convert obligations into blocking review items.
9:     Log new error patterns to Pitfall Registry.
10: end if
```

7.5 Layer 5: Post-Verification & Novelty

Checks if the proof matches the intended interpretation, is meaningfully correct (not trivial), and assesses novelty/plagiarism risks.

7.5.1 Novelty and subsumption decision rule

Layer 5 consumes the Literature Dossier and the Claim Ledger and produces a single decision label:

`known / probably known / unclear / probably novel`

The decision rule is conservative:

- If any candidate source plausibly subsumes the target statement, label `probably known` unless hypotheses are definitively incompatible.
- Label `probably novel` only if (i) the post-solve pass found no plausible subsumption, and (ii) the Claim Ledger contains evidence links for all nontrivial claims.

7.5.2 Output: Post-verification report (versioned artifact)

The report must include: (i) interpretation match verdict (unit tests pass/fail), (ii) a list of unverified dependencies (if any), (iii) the novelty label and rationale (with citations to the Literature Dossier), (iv) remaining risk register (what would most likely break under expert scrutiny).

7.5.3 Post-run learning extraction

After each completed run, the Orchestrator extracts reusable patterns:

- **Style rules** that emerged from review-loop friction (analogous to Kirov's iterative `CLAUDE.md` refinement [11]).
- **Pitfall entries** for the Formalization Bridge registry.
- **Prompt template refinements** based on actionable feedback.

These are stored as versioned artifacts in the run directory (`06_learning/`) and incorporated into future Context Bundles.

8 Structured Proof Context Management

8.1 Evidence-linked proof state: the claim ledger

The orchestrator maintains a claim ledger that links each nontrivial statement to an evidence object:

- **Derivation:** Linked to specific lemmas/steps.
- **Citation:** Linked to BibTeX key + location.
- **Computation:** Linked to code hash + seed.
- **Formal:** Linked to Lean/Coq theorem name.

A claim without an evidence link is not allowed to persist into the final draft.

8.2 Context Bundles: Prompt Interfaces Derived from Artifacts

A *Context Bundle* is the canonical prompt payload produced by the orchestrator for any agent action. It is derived mechanically from versioned artifacts, and it is the primary defense against proof-state drift.

8.2.1 Bundle contents (minimal)

Every bundle contains:

1. **Pointer header:** problem_id, run_id, and artifact versions/hashes for dossier, literature, experiments (if any), and the current proof draft.
2. **Definition lockfile excerpt:** only the active definitions/conventions needed for the current goal.
3. **Current local goal:** the lemma/claim being proved *now* (one-screen statement).
4. **Allowed dependencies:** the subset of cited theorems whose hypotheses have been checked.

5. **Open obligations:** blocking review items and unproven ledger claims relevant to the local goal.

6. **Stylistic & Tactic Constraints:** Constraints derived from the "Style Rules" artifact (e.g., equivalent to `CLAUDE.md`). Example: "Do not use ‘induction‘ on Reals"; "Avoid high-level tactics that obscure logical flow."

7. **Pitfall Registry:** Relevant entries from `TACTICS.md` to prevent recurring errors (e.g., "Note: `linarith` treats $1/a$ and a^{-1} as different atoms").

8. **Do-not-do constraints:** e.g. “do not change definitions”; “do not introduce new symbols without adding to lockfile”.

8.2.2 Context compiler

The orchestrator compiles bundles under a token budget by prioritizing: (1) lockfile + goal, (2) blocking items, (3) minimal dependency statements, (4) only then supporting narrative.

Algorithm 4 Context bundle compiler

```
1: Input: artifact pointers, current goal $G$, token budget $T$
2: Include pointer header + lockfile excerpt needed for $G$
3: Include blocking review items and the ledger slice relevant to $G$
4: Include checked dependency theorem statements (not whole papers)
5: Include style constraints (e.g. TACTICS.md content)
6: while tokens used $< T$ and more helpful context exists do
7:     Add the next-highest-priority item (prioritized by relevance to $G$)
8: end while
9: Output: Context Bundle (machine-recorded) + a short “omitted context” list
```

Recording bundles as artifacts makes later audits precise: reviewers can verify that a draft was produced under the exact same locked definitions and obligations.

9 Complete Workflow Specification

```
task: solve_research_math_problem
agents:
  computational: "Role: Computational Agent"
  synthesis: "Role: Synthesis Agent"
  review: "Role: Review Agent"
  orchestrator: "Coordination layer"

phases:
  1_understand:
    agent: orchestrator
    actions:
      - generate_interpretations
      - human_checkpoint: "Which interpretation?"
    output: interpretation_dossier

  2_literature:
    agent: orchestrator
    actions:
      - pre_solve_literature_search
      - build_dependency_graph
    output: literature_dossier
```

```
3_computational_experimentation:
  trigger: numerical_objects
  agent: computational
  actions:
    - create_experimental_code
    - code_review_loop:
        reviewer: review
        author: computational
        max_iterations: 2
    - execute_experiments
  output: numerical_evidence

4a_proof_planning:
  agent: synthesis
  input: [interpretation_dossier, literature_dossier,
          numerical_evidence (if available)]
  actions:
    - meta_prompted_plan
  output: proof_plan

4b_proof_draft:
  agent: synthesis
  input: proof_plan
  actions:
    - execute_plan_to_latex
  output: P_0

4c_adversarial_review_loop:
  max_iterations: 5
  change_budget_per_iteration: 3
  loop:
    - agent: review
      action: targeted_review(P_i, budget=3)
      output: review_R_i
    - agent: synthesis
      action: accept_reject_revise(P_i, R_i)
      output: P_{i+1}
  output: final_proof

4d_formalization_bridge:
  trigger: formalization_feasible
  agent: aristotle (external)
  input: context_bundle_for_slice
  actions:
    - generate_formalization_packet
    - run_safety_scan
    - run_prover
  output:
    status: verified | failed
    obligations: list_of_blocking_items

5a_post_verification:
  agent: orchestrator
  actions:
    - post_solve_literature_search
    - interpretation_match_check
    - novelty_assessment
  output: post_verification_report

5b_learning_extraction:
  agent: orchestrator
  actions:
    - extract_style_rules
```

```
      - update_pitfall_registry
      - refine_prompt_templates
    output: [pitfalls_v1.md, style_rules_v1.md]
```

Listing 1: Integrated workflow

10 Transparency & Audit Trail

Full reasoning logs, citations for every claim, and a complete revision history are required for scientific integrity.

11 Threat Model and Integrity Safeguards

We treat all non-local inputs as **untrusted**: retrieved documents, PDFs, webpages, and even intermediate model outputs. The goal is to prevent (i) solving the wrong problem, (ii) shipping an incorrect proof with high apparent confidence, and (iii) misattribution or plagiarism.

11.1 Threats

T1. Retrieval prompt injection: a retrieved document contains instructions that alter agent behavior.

T2. Citation laundering: the system cites a source that does not support the cited claim.

T3. Training-data reconstruction: the system reproduces a known proof without attribution.

T4. Confidentiality leakage: private notes or unpublished results appear in logs or artifacts.

T5. Toolchain drift: changes in models/tools invalidate earlier calibration and reproducibility.

T6. Formalization misstatement: the AI generates a valid Lean proof of a statement that is semantically different from the intended claim (e.g., weaker hypotheses, trivial variants) [8].

11.2 Safeguards

S1. Instruction isolation: retrieved text is ingested as data only; the orchestrator strips or ignores imperative instructions from sources.

S2. Cite-then-quote discipline: for every cited claim, record the exact theorem statement/location in the Literature Dossier; block claims with missing locations.

S3. Provenance gates: the Claim Ledger is a hard gate: no nontrivial claim may persist without an evidence pointer (derivation/citation/computation/formal artifact).

S4. Redaction policy: logs and artifacts must support redaction of confidential content before sharing.

S5. Version pinning: every run records model identifiers, tool versions, and environment lockfiles (`env.lock`); any change triggers recalibration of confidence.

S6. Statement-proof firewall: Formal statements are produced/verified by a human checkpoint. The axiom audit (`#print axioms`) is a mandatory gate.

These controls are lightweight but materially reduce the most damaging failure modes identified in the case studies.

12 Worked Example: One End-to-End Run

To ground the theoretical architecture in practice, this section provides a concrete walkthrough of a single problem run. It demonstrates how the abstract layers interact to produce specific, versioned artifacts. By examining the Interpretation Dossier, Proof Trace, and Claim Ledger, implementers can visualize the exact granularity required for the audit trail.

12.1 Layer 1 excerpt: Interpretation Dossier (v1)

Field	Content (excerpt)
Statement	Let $G = (V, E)$ be a connected d-regular graph with n vertices. Prove that the second eigenvalue λ_2 of the Laplacian satisfies $\lambda_2 \geq d - 2\sqrt{d-1}$ [6].
Definitions	• **Combinatorial Laplacian:** $L = D - A$ (chosen). • **Normalized Laplacian:** $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$ (rejected).
Ambiguity	The term "spectral gap" typically refers to $\lambda_1(\mathcal{L})$ in Cheeger inequality contexts [?], but to $\lambda_2(L)$ in algebraic connectivity contexts.
Decision	Selected *Combinatorial* because Section 2 mentions integer spectrum properties.
Unit Tests	**Test 1 (Triangle K_3):** $L = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$. Eigenvalues $\{0, 3, 3\}$. $\lambda_2 = 3$. **Test 2 (Edge K_2):** $L = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. Eigenvalues $\{0, 2\}$. $\lambda_2 = 2$.

12.2 Layer 4 excerpt: Proof Trace (Review Loop)

Cycle 0 (Draft P_0):

> "...We proceed by induction on the size of the graph. Assume the bound holds for all $k < n$. For a graph G of size n, we decompose it into connected components..."

Review 0 (Critique R_0):

- **Blocking Error:** The induction step assumes that removing a vertex preserves d-regularity, which is false. Removing vertices from a d-regular graph creates a graph with maximum degree at most d, but it is not necessarily regular.
- **Missing Definition:** The symbol $\psi(G)$ appears in Eq (3) but is not defined in the lockfile or the text.
- **Recommendation:** Replace the inductive argument with a variational characterization using the Courant-Fischer min-max theorem.

Cycle 1 (Revision P_1):

> "...**Revised Argument:** By the Courant-Fischer theorem, $\lambda_2 = \min_{x \perp \mathbf{1}, x \neq 0} \frac{x^T L x}{x^T x}$. Let x be the vector...**Definition:** Let $\psi(G)$ denote the graph conductance..."

Review 1 (R_1):

- **Status:** Accepted. No blocking errors found.

12.3 Layer 4d excerpt: Formalization Validation

Axiom audit:

```
#print axioms spectral_gap_lower_bound
- [propext, Classical.choice, Quot.sound]   ✓
```

Statement match: Human confirmed: Lean statement `theorem spectral_gap_lower_bound ...` matches Interpretation Dossier v1.
Pitfall logged: Initial attempt used `NNReal` for eigenvalues, causing type mismatch with Laplacian defined over $\mathbb{R}$. Added to pitfall registry: "Use $\mathbb{R}$ for spectral quantities unless positivity proof is needed."

12.4 Claim Ledger Excerpt

ID	**Claim Description**	**Status**	**Evidence Pointer**
C01	L is positive semi-definite	Derived	Lemma 2.1 (from $x^T Lx = \sum(x_i - x_j)^2$)
C07	$\lambda_2 \geq d - 2\sqrt{d-1}$	Cited	`Alon1986`, Theorem 1, Eq (1.2)
C12	Operator norm $\|M\| \leq 1$	Derived	Step 3.4 via Triangle Inequality
C19	Gap > 0 for $n \in [5, 50]$	Computed	`exp_v1.py` (SHA: `a7b8...`) w/ seed `42`

13 Human-in-the-Loop Checkpoints

```
CHECKPOINT 1: Problem interpretation
  [HUMAN: Confirm / Correct / Clarify]

CHECKPOINT 2: Strategy selection
  [HUMAN: Proceed / Redirect / Add constraint]

CHECKPOINT 3: Solution review
  "Proof complete. Confidence: HIGH.
   Novelty: No similar proof structure found."
  [HUMAN: Accept / Flag for expert / Reject]
```

14 Discussion

The "First Proof" benchmark provided a useful stress test. By applying a structured multi-agent workflow with predefined prompts and no injected domain-specific proof steps, we obtained draft solutions for all ten problems. The key architectural insight is that *no single model or single-pass approach suffices.* The combination of computational experimentation, meta-prompted planning, and iterative adversarial review addresses different failure modes.

Integrating lessons from AI-assisted formalization practice—particularly the Lean validation protocols documented by the Lean community [9] and the iterative prompt-engineering workflows demonstrated by Kirov [11]—strengthens the architecture in two ways. First, the *statement-proof firewall* (Safeguard S6) addresses the dominant failure mode in formal verification: proving a subtly different statement than intended. Second, the *pitfall registry* and *post-run learning extraction* create a feedback loop that improves the system across runs, transforming each formalization failure into reusable knowledge.

Looking forward, the most impactful improvements will come from better orchestration—specifically, better management of the **human** checkpoints, rigid artifact contracts, and formal verification integration. As Althöfer noted two decades ago [3], the power lies in the combination of machine generation and **human** selection.

15 References

References

[1] Abouzaid, M., et al. (2026). *First Proof.* arXiv:2602.05192v1 [cs.AI].

[2] Feng, T., Trinh, T., et al. (2026). "Semi-Autonomous Mathematics Discovery with Gemini: A Case Study on the Erdős Problems." arXiv:2601.22401v1 [cs.AI].

[3] Althöfer, I. (2004). "Improved game play by multiple computer hints." *Theoretical Computer Science*, 313, 315–324.

[4] Harmonic. "Aristotle: IMO-level Automated Theorem Proving." https://harmonic.fun

[5] Wolz, D. (2026). "Multi-Agent Strategy for Solving Research-Level Mathematics." https://althofer.de/first-proof-competition/first-proof-report.html

[6] Alon, N. (1986). "Eigenvalues and expanders." *Combinatorica*, 6(2), 83–96.

[7] Cheeger, J. (1971). "A lower bound for the smallest eigenvalue of the Laplacian." *Problems in Analysis*, Princeton University Press, 195–200.

[8] Tao, T. et al. (2026). "I think I managed to get my favorite AI tool to solve an open Erdős problem! What do I do next?" https://github.com/teorth/erdosproblems/wiki/

[9] Lean Community (2025). "Did you prove it?" https://leanprover-community.github.io/did_you_prove_it.html

[10] Lean Reference (2025). "Validating Proofs." https://lean-lang.org/doc/reference/latest/ValidatingProofs/

[11] Kirov, R. (2026). "Leaning on AI." https://rkirov.github.io/posts/lean5/

[12] Kirov, R. (2026). Claude Code prompt for Lean formalization. https://github.com/rkirov/analysis/blob/main/CLAUDE.md

[13] Galois, Inc. (2025). "Claude Can (Sometimes) Prove It." https://www.galois.com/articles/claude-can-sometimes-prove-it