Step-Level Elaboration Improves Natural Language Verification of Research-Level Mathematical Proofs

Yifeng Sun [1] **Kun Yuan** [2]

Abstract

Large Language Models (LLMs) struggle to rigorously verify complex mathematical proofs. Standard global evaluation approaches suffer from "context poisoning," in which superficially plausible statements mask subtle logical flaws, leading to hallucination or over-skepticism. To address this, we shift from global evaluation to strict step-level verification: our framework attempts detailed constructive elaboration for each deduction step while strictly constraining the sources of applied theorems. We evaluate on a curated adversarial diagnostic suite of research-level proofs drawn from the FirstProof challenge. A systematic ablation study suggests that constructive step-level elaboration and control over external mathematical sources are indispensable for localizing subtle logical errors. Beyond improving global evaluation, our approach fundamentally alters the failure taxonomy. Error analysis reveals that, rather than exhibiting severe logical hallucinations, remaining rejections are primarily instances of "pedantic hyper-rigor" stemming from unstated domain conventions, effectively exposing implicit ambiguities within the expert benchmark itself. Our findings suggest that shifting verification from direct judgment to detailed constructive proof elaboration, together with control over external mathematical sources, can substantially improve pure natural-language agents' ability to distinguish rigorous proofs from flawed ones, with the potential to strengthen agentic reasoning on frontier mathematical concepts that the base model does not already know well, and implications for future automated proof-review agents. Code and prompts are available at GitHub.

[1]Department of Mathematics, Southern University of Science and Technology [2]Center for Machine Learning Research, Peking University. Correspondence to: Kun Yuan <kunyuan@pku.edu.cn>.

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

The self-correction and reasoning capabilities of Large Language Models (LLMs) and AI agents have improved significantly with scaling and reinforcement learning (DeepSeek-AI et al., 2025). In the mathematical domain, agent frameworks have demonstrated impressive problem-solving results (Feng et al., 2026; Ju et al., 2026). However, when verifying research-level mathematical proofs, current models frequently falter. Standard global evaluation approaches often suffer from "context poisoning," where superficially plausible statements mask subtle spurious justifications, leading agents to either hallucinate validity or exhibit over-skepticism.

While formal verification systems such as Lean (de Moura & Ullrich, 2021) offer absolute guarantees, their application is bottlenecked by human-built infrastructures and a heavy reliance on manually maintained libraries (e.g., Mathlib (The mathlib Community, 2019)). This naturally constrains the ability of current automated agents to formally verify advanced mathematical concepts. Therefore, verifying proofs in natural language remains an essential frontier. Generating fine-grained elaborations of informal proofs serves as a crucial bridge: it not only aids human mathematicians in peer review but also assists agentic systems in eventually translating informal math into rigorous formal code (Wu et al., 2022; Jiang et al., 2022; Azerbayev et al., 2023; Yang et al., 2023; Zheng et al., 2021).

Crucially, this focus on AI self-correction serves a fundamentally different role than formalization itself. Formal verification systems operate on a paradigm of ultimate accountability, where the underlying infrastructure (e.g., Lean) or a human expert bears the responsibility of providing an absolute guarantee of correctness. In contrast, our objective is to equip the agent with an intrinsic capacity to distinguish sound reasoning from flawed logic during its exploratory and generative phases. After all, agents will inevitably encounter scenarios where they must rely entirely on pure natural language reasoning to conceptualize and discover solutions. In the highly unconstrained search space of mathematical discovery, enabling an agent to recognize its own missteps is critical for pruning dead ends and drastically reducing the generation of hallucinatory or structurally flawed

proofs. Ultimately, we aim to demonstrate a core principle of agentic reasoning: **by forcing a model to explicitly and exhaustively write out the underlying details of an informal proof, we inherently amplify its ability to discover subtle logical errors that its standard reasoning would otherwise gloss over**. It is important to emphasize that, to validate our approach, we merely task the agent with the process of elaboration rather than demanding a flawlessly exhaustive natural language proof as the final output. Consequently, our objective is not to reinvent de Bruijn's Mathematical Vernacular (MV). Rather, we aim to demonstrate that agents—much like human mathematicians—leverage the very act of detailed writing to ascertain which conclusions are genuinely correct and rigorously proven.

To achieve this, we aim to design an agent framework that evaluates proofs with the strictness of a human mathematician reviewing a manuscript. Such a system must prioritize **soundness** (eliminating false acceptances of flawed proofs) over mere completeness, and effectively **leverage inference-time compute** to achieve greater clarity the longer it reasons. We introduce a crucial methodological shift: **reframing verification from direct anomaly detection to constructive elaboration** via **strict step-level verification**. Rather than directly querying an LLM to spot flaws within a global context—a task highly susceptible to context poisoning and hallucination—we test whether the proof can be rigorously and explicitly expanded. For a mathematical text S treated as a string, we decompose it into an ordered sequence of **steps** $(s_1, s_2, \ldots, s_n)$, isolating **deduction steps** that contain logical assertions. By "elaborating a step," we mean expanding a deduction step s_i into a finer-grained derivation string s'_i in natural language that strictly preserves the original reasoning trajectory but explicitly unpacks the underlying logic. Consequently, anomaly detection emerges naturally from the generative process: if an agent fails to construct a valid s'_i, we suspect s_i harbors a logical gap or unsound justification. By forcing the agent to elaborate, we replace the LLM's opaque internal judgments with strict generation constraints (Lightman et al., 2023; Cobbe et al., 2021). Every elaborated step is required to explicitly ground its applied theorems and logical derivations, reducing superficial plausibility.

To organize this source-aware elaboration process and make the provenance of each deduction explicit, we use a formal descriptive scaffold. Specifically, in our implementation, we model the logical dependencies of any deduction step using a tripartite architecture: **internal context** (Γ_i, encompassing theorems, definitions, and hypotheses directly originating from the proof text S), **external knowledge** (Σ_i, comprising non-trivial statements drawn from outside literature or established domain consensus), and **background theory** ($\mathcal{T}_i$, representing trivial statements or rewrite rules necessary to syntactically derive the assertion).

Operationally, we implement this source-aware step-level verification as an autonomous, state-driven agentic workflow guided by fixed prompt files. By default, every deduction step s_i is initialized with a conservative local status. During each iteration, the agent attempts to elaborate the remaining `open` steps. The state transitions are based on whether the agent can close the step while preserving the original route and explicitly supporting any external theorem use: if an elaboration is successfully derived, the step transitions to `verified`; conversely, if the agent identifies a distinct logical error, it flags the step as `flawed`. To rigorously guard against false positives (i.e., pedantic over-rejection), any `flawed` step triggers a fixed confirmation phase: the agent attempts to confirm the error, potentially reverting the state back to `open` if the perceived flaw is resolved upon re-evaluation. This iterative process is governed by strict halting criteria: (1) **Acceptance:** the proof is deemed valid if and only if all steps reach the `verified` state; (2) **Rejection:** the process terminates immediately and rejects the proof if a `flawed` state is definitively confirmed upon reflection; and (3) **Exhaustion:** to prevent infinite loops on intractable gaps, the system globally halts and rejects the proof after five consecutive reasoning resumptions without an observed progress signal.

To empirically validate our proof of concept and demonstrate its potential reward, we construct a highly curated diagnostic suite of challenging mathematical proofs. We focus on a small-scale sample to ensure absolute clarity in our evaluation: every proof has been meticulously verified by human experts to establish an unambiguous ground truth, classifying them as either strictly valid or explicitly invalid. Specifically, our diagnostic suite is heavily anchored in the **FirstProof** challenge (Abouzaid et al., 2026; First Proof Project, 2026a;b). To establish an unambiguous ground truth, we collected proofs that have undergone rigorous expert evaluation, securing a highly curated collection of 21 research-grade proofs definitively classified as either strictly valid or explicitly invalid. These 21 samples are strategically drawn from three authoritative sources: (1) official First-Proof releases, (2) correct proofs generated by the Aletheia, and (3) generations from OpenAI's solutions. By anchoring our evaluation on this precise, expert-verified suite, we can investigate how our step-level verification effectively isolates subtle reasoning errors that standard global evaluators consistently miss.

In summary, our main contributions are threefold:

- **Methodological Paradigm Shift:** We reframe natural language proof verification from direct, black-box anomaly detection to *constructive elaboration*. Our implementation uses a $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ tripartite architecture to organize internal contexts, external knowledge, and background theories, making applied theorem sources explicit

and mitigating superficial pattern matching.

- **Autonomous Verification Workflow:** We propose a state-driven agentic framework governed by strict step-level constraints. Equipped with a fixed confirmation phase and robust halting criteria, our system prioritizes logical soundness and effectively leverages inference-time compute to prevent the "context poisoning" that plagues global evaluators.
- **Empirical Insights via Micro-benchmarking:** Our evaluation demonstrates that while even highly capable standard baselines can still be deceived by structurally rigorous pseudoproofs, our framework achieves zero false acceptances on the problem set. Furthermore, our qualitative analysis reveals a critical bottleneck in current verification tasks: the vulnerability of base models to unstated, domain-specific hidden conventions.

2. Related Work

Math Agents Agentic frameworks for mathematics have advanced rapidly and are producing more and more significant results in research-level mathematics. These agents may or may not utilize proof assistants and formal languages. Recent frontier breakthroughs have heavily favored the second paradigm: agents conduct exploration, reasoning, and proof construction entirely in pure natural language. For instance, RETHLAS (Ju et al., 2026) can operate as an informal reasoning agent that mimics human mathematical workflows—exploring literature and proposing candidate natural-language proofs for advanced open problems. Similarly, ALETHEIA (Feng et al., 2026) functions as an end-to-end natural language research agent capable of iteratively generating, verifying, and revising solutions from PhD-level exercises to autonomous research papers. Most recently, OPENAI'S MODEL (OpenAI, 2026) achieved a historic milestone by disproving Erdős's 80-year-old planar unit distance conjecture using highly complex, purely informal reasoning. However, despite their immense generative power, a critical vulnerability of these pure natural language agents is their susceptibility to hallucinations and subtle logical gaps, particularly when self-evaluating long-horizon mathematical proofs. (Liu et al., 2026b; Liu & Qin, 2026; Liu et al., 2026a) Our work directly addresses this vulnerability. We demonstrate a structured pathway for pure natural language agents to independently overcome these blind spots: by enforcing strict, step-level detailed elaboration and source-controlled theorem use, agents can systematically unpack their reasoning, rigorously detect anomalies, and eliminate hallucinations without needing to fall back on external formalization infrastructures.

Mathematical Vernacular The conceptual ambition of bridging informal human mathematical discourse and strict mechanical logic dates back to de Bruijn's Mathematical Vernacular (MV) (de Bruijn, 1994). MV was proposed as an intermediate representational language that obeys formal grammatical rules while preserving the natural "resting points" (steps) of mathematical reasoning, thus serving as a stepping stone toward fully coded formal systems (e.g., Automath). Importantly, we do not intend to replicate MV or construct a rigid natural-language proof assistant. Rather, we adopt its underlying philosophy to help autonomous agents systematically reduce logical hallucinations. In our implementation, structuring natural language proofs around the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ tuple extraction provides one concrete way to isolate these resting points and record which claims come from the proof text versus external knowledge. Crucially, in practice, we do not algorithmically verify whether the agent flawlessly executes this extraction; the mere attempt to engage in this highly structured elaboration is sufficient to expose hidden logical flaws and elevate the agent's evaluative accuracy.

3. Methodology

For a mathematical text S treated as a string, it can naturally be decomposed into an ordered sequence of steps $(s_1, s_2, \ldots, s_n)$. Unlike fully formalized proofs in systems like Lean or Rocq (de Moura & Ullrich, 2021; The mathlib Community, 2019), natural language mathematical proofs inherently omit trivial algebraic manipulations and elementary logical deductions to maintain readability. Because of these inherent omissions, evaluating a proof as a monolithic global text often obscures subtle flaws and logical gaps.

To address this, we reframe verification from direct anomaly detection to **constructive elaboration**. By "elaborating a step", we mean that for a given deduction step s_i with an assertion (derived conclusion statement) a_i, we provide an expanded derivation s'_i that concludes with the identical assertion as s_i. Crucially, s'_i must strictly preserve the original reasoning trajectory of s_i, but unpack the logic at a finer granularity. If we fail to come up with a valid expanded derivation, we suspect s_i to contain a logical gap or flaw.

3.1. The Tripartite Explanation Framework

To systematically investigate whether an elaboration s'_i successfully bridges a gap without demanding explicit syntactic completeness (which misaligns with human communication), we introduce the formal concept of an explanation.

It is crucial to clarify that this theoretical framework is not artificially engineered solely for AI agents; rather, it stems directly from a refinement of the routine cognitive actions human mathematicians employ when explaining and validating a proof. By adopting this stance, the framework is sufficiently *descriptive* to ensure universal applicability to human mathematical discourse, while simultaneously pro-

viding a rigorous, *prescriptive* structure that conveniently facilitates systematical analysis and guides agentic workflows. (Ganesalingam, 2013) In our implementation, we explicitly prompt the agent to output its elaborations according to the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ tripartite structure. However, it is worth noting that we do not claim this explicit formatting is necessarily the optimal approach, given the additional generation overhead it imposes on the agent. Alternative implementations may achieve similar rigor by simply requiring the agent to maintain a detailed, distinct record of internal contexts versus external knowledge, without strictly enforcing the tripartite syntax.

We model the logical dependencies of any deduction step (or its elaboration) using this framework. We define an **explanation** of the step s_i as a triple of sets $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$. A step s_i is considered valid if there exists an explanation such that:

$$\Gamma_i \cup \Sigma_i \vdash_{\mathcal{T}_i} a_i \tag{1}$$

where $\vdash$ denotes rigorous informal entailment. The components reflect how mathematical justifications are sourced:

- **Local Context (Γ_i):** This contains finitely many statements (theorems, definitions, and hypotheses) directly originating from S. These statements may be explicit or implicit, but they must be effective within the current scope (e.g., local assumptions within a case analysis block expire once the block concludes).
- **Global Knowledge (Σ_i):** This comprises non-trivial statements drawn from external literature or established domain consensus (i.e., outside of S). To prevent the hallucination of non-existent theorems, any statement invoked from Σ_i must be critically examined.
- **Background Theory ($\mathcal{T}_i$):** This comprises trivial statements, fundamental logical axioms (e.g., Modus Ponens), and rewrite rules. By isolating $\mathcal{T}_i$, we bypass the need for rigid algebraic hierarchies (like those in Mathlib), allowing the direct manipulation of sound statements over concrete objects while safely decoupling structural type constraints from the core deductive logic.

Crucially, an explanation $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ should not be viewed as the complete set of all contextually valid truths, but exclusively as the *minimal required set* of statements necessary to validate the natural language proof step rigorously. A comprehensive discussion of this framework and what constitutes a valid explanation is provided in Appendix B.

3.2. State-Driven Verification Agent

While the tripartite framework describes a natural decomposition of human mathematical validation, it also serves as a useful analytical scaffolding for AI systems. To explore the feasibility of highly efficient natural language proof verification, we implement a lightweight, state-driven agent. The agent utilizes this framework because it conveniently enforces traceable reasoning and provides a systematic taxonomy for isolating failure modes.

The agent operates without relying on a formal proof assistant or domain-specific theorem whitelists. Instead, it maintains a discrete cognitive state $c_i \in \{\texttt{OPEN}, \texttt{VERIFIED}, \texttt{FLAWED}\}$ for each step s_i.

Initialization and Iteration Upon initialization, the agent assigns an `OPEN` status to all deduction steps. During each iteration, the agent selects the most critical `OPEN` step and attempts to write a thick elaboration s_i', explicitly extracting the corresponding $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$.

State Transition Rules The agent's transitions are governed by strict auditing mechanisms to prevent both pedantic false positives and dangerous false negatives:

1. `OPEN` $\rightarrow$ `VERIFIED`: A step transitions to `VERIFIED` if and only if its written elaboration is logically closed *and* every external item used in Σ_i is explicitly supported (either by a direct sub-proof generated within the step file or by a precise literature citation).
2. `OPEN` $\rightarrow$ `FLAWED`: A step is marked `FLAWED` only when the agent can exhibit an **explicit flaw witness**. This includes an exact false statement, an invalid inference from written premises, an explicit contradiction with the already verified local context Γ_i, or a demonstrably misapplied external theorem in Σ_i.
3. **Flaw Confirmation** (`FLAWED` $\rightarrow$ `OPEN`): To rigorously guard against pedantic false positives, any preliminary transition to `FLAWED` triggers an adversarial investigation. The agent evaluates whether the proposed flaw witness represents a genuine structural mathematical error or merely a superficial textual nitpick (e.g., relying on an unstated standard domain convention or a terminology-dependent edge case). If the perceived flaw is determined to be an epistemic ambiguity rather than a logical breakdown, the step is conservatively reverted to `OPEN`.
4. **Dynamic Decomposition** (`OPEN` $\rightarrow \{\texttt{OPEN}_a, \texttt{OPEN}_b\}$): If a deduction step s_i is deemed too coarse or encompasses multiple non-trivial logical leaps, the agent is permitted to dynamically decompose it into finer-grained sub-steps. The original s_i is replaced in the global sequence by these new sub-steps, which are individually initialized as `OPEN` and evaluated independently.

We emphasize a critical design choice: *failure to close a difficult theorem does not by itself count as a flaw.* If an external burden remains unsupported, the step conservatively remains `OPEN`.

Global Halting Criteria The autonomous verification loop progressively updates the step states and their dependencies. The process iterates until one of the following strict halting criteria is triggered, determining the final validity of the global proof S:

- **Acceptance (Soundness):** The global proof is deemed valid and accepted if and only if the entire sequence successfully reaches the terminal verified state: $\forall i, c_i =$ `VERIFIED`. If the text contains several proofs, it is considered valid if entire sequence of one of the proofs is valid.
- **Rejection via Flaw Confirmation:** If any step enters the `FLAWED` state, it must undergo the aforementioned Flaw Confirmation investigation. If a flaw is explicitly confirmed (i.e., it is verified as a genuine structural error and not reverted to `OPEN`), the system halts and the global proof is immediately rejected.
- **Rejection via Exhaustion:** The agent may fail to give a disproof of a flawed step. Thus to prevent looping on intractable gaps or missing foundational knowledge, the runner monitors observed progress signals. If five consecutive resume iterations produce neither an `OPEN`→`VERIFIED` transition nor an `OPEN`-step split, the system terminates the loop and conservatively rejects the proof.

External Knowledge Retrieval In our pipeline, the external knowledge set Σ_i is primarily sourced through active internet retrieval. To ensure the reliability of this external knowledge, our prompt files explicitly direct the agent to prioritize established mathematical textbooks and directly relevant research literature. While these instructions guide the agent's search behavior, we do not computationally enforce strict constraints on the exact provenance or syntactic precision of the cited theorems. However, general open-ended retrieval introduces a broader theoretical challenge: unconstrained agents often overlook the strict technical constraints or local assumptions required by retrieved theorems. This risks creating a "contextual misapplication," where a globally valid theorem is incorrectly applied to a local context (Γ_i) because its underlying prerequisites are not fully met. Specifically, without source-aware elaboration and strict faithfulness to the original text, an agent might invoke unmentioned external knowledge to artificially justify flawed steps during extended multi-turn executions. Successfully navigating this latent risk underscores the critical necessity of controlling external knowledge integration while preserving the author's original route. Consequently, a crucial direction for future mathematical agents is the development of rigorous controls over external knowledge integration. Future systems must move beyond basic search by explicitly verifying the authoritative provenance of the literature and ensuring the complete, precise extraction of all prerequisite conditions before any external theorem is applied.

Structured Auditing via the Theorem Ledger To rigorously ground external mathematical dependencies (Σ_i), our framework asks the agent to maintain a *Theorem Ledger*—a centralized registry tracking external claims invoked by the agent. During the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ tuple extraction (full schema in Appendix), the ledger records the `Status` of cited theorems $\Sigma_j \in \Sigma_i$. A deduction is marked `VERIFIED` only if the agent provides explicit support (a proof sketch or precise citation) and confirms its valid applicability within the local context Γ_i. If any dependency remains `OPEN`, downstream reasoning is flagged as *conditionally closed*, preventing premature validation based on unvetted assumptions. This tracking mechanism transforms verification from a black-box judgment into a transparent, auditable process.

4. Experiments

We design our empirical evaluation to answer two core research questions: (1) How does our structured, agentic approach compare against standard global evaluators and naive granular prompting across diverse mathematical fallacies? (2) How does the agent utilize its state-driven logic to navigate and resolve highly adversarial, research-level proofs during extended inference?

To address these, we first present a comprehensive evaluation across different verification pipelines on a curated 21-proof suite (Section 4.1 and 4.2). Subsequently, we provide a deep qualitative analysis of the agent's cognitive trajectory on specific intractable cases (Section 4.3).

4.1. Experimental Setup

Diagnostic Suite We evaluate our framework on a highly curated diagnostic suite of 21 research-grade mathematical proofs derived from the FirstProof challenge (Abouzaid et al., 2026; First Proof Project, 2026b). This suite comprises both fundamentally valid proofs (which may contain minor, fixable omissions) and invalid proofs containing fatal mathematical flaws or gaps. Specifically, we structure the diagnostic suite into two distinct subsets based on their ground-truth validity:

Valid Proofs (11 cases) This subset consists of 9 proofs generated by the Aletheia framework (Feng et al., 2026) and 2 proofs from OpenAI. The Aletheia subset (comprising variants 2A, 2B, 5A, 7B, 8B, 9A, 9B, 10A, and 10B, where the suffixes denote outputs from two distinct agent architectures) was rigorously vetted and certified as mathematically sound by human domain experts in the original Aletheia study. The remaining two valid cases are OpenAI's proofs for problems 4 and 6, which have reached a defini-

tive consensus of correctness within the formal mathematics community (e.g., Zulip expert discussions).

Invalid Proofs (10 cases) To systematically source authentic, AI-generated flawed proofs, we analyzed the human expert critiques provided in the original FirstProof paper (Abouzaid et al., 2026). We specifically targeted problems where the baseline LLMs were explicitly evaluated as committing unambiguous logical breakdowns, retrieving the corresponding raw proof texts directly from the official FirstProof repository. There are 8 proofs collected in this way. To ensure the inclusion of deeply adversarial examples, we augmented this set with Aletheia's proof 7A and OpenAI's proof 2, both of which are recognized flawed. In particular, some of the key mathematical source material for our analysis of the FirstProof Question 7 and Question 8 cases is documented in the official solutions/comments document and in the associated mathematical background references (First Proof Project, 2026b; Weinberger, 2022; Fowler, 2012; Alfieri & Novak, 2026).

Baselines and Pipeline Configurations To isolate the source of our framework's performance, we compare our agent against three distinct baselines, testing the limits of direct prompting and textual granularity:

1. **Global GPT-5.4-xhigh:** A standard zero-shot LLM-as-a-judge prompt, representing the state-of-the-art capability of a generic frontier model directly evaluating the monolithic text.
2. **Global Codex 5.4-xhigh:** The identical base model used by our agent, prompted to globally verify the proof by reading sentence by sentence. Also it is told to be strict. This isolates the model capacity of retrieving flaws and gaps.
3. **Codex 5.4-xhigh Sentence-by-Sentence (Ablation):** To determine whether performance gains merely stem from granular reading, we implement a baseline where the proof is mechanically split into single sentences. The model evaluates each sentence sequentially but lacks explicit source/dependency tracking and state-driven memory. The prompts are designed to be almost the same, except it does not require constructive elaboration with exact external theorem support.
4. **Constructive Verification Agent (Ours):** The fully equipped state-machine agent implementing source-aware constructive elaboration through the tripartite explanation framework.

To ensure a fair comparison, the zero-shot baseline prompt was empirically optimized through preliminary experiments to faithfully elicit the model's peak reasoning capability. Quantitative results for suboptimal prompt variants are omitted for brevity.

Evaluation Metrics A verification attempt is considered successful if and only if it satisfies a strict dual-criterion: it must accurately assign the correct global verdict (`VERIFIED` or `FLAWED`), *and* for flawed proofs, it must correctly isolate an offending step without hallucinating false negatives elsewhere in the proof.

Data Contamination Risk and Audit Protocol Since our diagnostic suite is derived from the public FirstProof challenge, data contamination remains a serious concern when evaluating frontier models. For completed runs, we audited the retrieval and reasoning logs; if a log mentioned FirstProof-related terms or otherwise indicated direct exposure to FirstProof source material or discussions, we discarded that run and repeated the experiment. Apart from such explicit exposure, the risk is substantially mitigated by the structure of the protocol: a verdict is not accepted merely because the model recognizes a global answer pattern, but must be supported by local step elaborations, explicit dependency tracking, and auditable uses of external knowledge. As an additional sanity check, we found that prompting Gemini 3.1 Pro to "find any flaws or nontrivial gaps" rather than asking whether a proof is correct can solve all 21 cases in our suite. We do not interpret this as evidence that a one-shot prompt is superior to our method. A more plausible concern is that Gemini may have been tuned, directly or indirectly, on internet-visible FirstProof materials, creating a risk of benchmark leakage. In particular, when Gemini judges that a correct solution route is valid, it may be recognizing an internet-exposed answer pattern without fully attending to the subtle technical assumptions that make the argument sound. This observation reinforces the need to treat public benchmark results for natural-language proof verification with caution, and motivates our emphasis on auditable step-level reasoning rather than opaque global verdicts.

4.2. General Verification Performance

We evaluated the four pipelines on the 21-proof diagnostic suite over 3 independent trials. While the standard direct prompt baseline (operating at a sampling temperature of 1.0) exhibited slight outcome variance, our framework demonstrated macro-level consistency, yielding identical final verdicts across all runs. The aggregated results are presented in Table 1.

In our evaluation on the curated 21-proof suite, we observe that current large language models demonstrate strong capabilities in standard contexts: the standard global baseline successfully identified the logical errors in 8 out of the 10 explicitly invalid proofs. However, a detailed analysis of the remaining 2 cases where the global pipeline failed reveals a specific vulnerability. In these specific cases, the flawed proofs package incorrect mathematical knowledge—which the LLM is unfamiliar with—into structurally rigorous and

Table 1. Verification performance on the 21-proof diagnostic suite (avg@3 over three independent trials). We report the number of correctly classified proofs and the specific error types (False Positives/Negatives). The agent is soundness-biased: it eliminates hallucinated validations on invalid proofs in this suite, while conservatively rejecting more valid proofs than the sentence-by-sentence ablation.

Pipeline	**Valid Proofs (11 total)**		**Invalid Proofs (10 total)**		**Overall Acc.**
	Correct (✓)	False Positives	Correct (✓)	False Negatives	
Global GPT-5.4	8.33	2.67	7.67	2.33	16/21
Global Codex-5.4	7.33	3.67	8	2	15.33/21
Codex 5.4 Sentence-by-Sentence	9	2	9	1	18/21
Agent (Ours)	**8**	**3**	**10**	**0**	**18/21**

plausible formats. This empirical result highlights that proof verification remains a highly challenging task, particularly for novel or frontier mathematics where LLMs lack prior familiarity with the specific domain knowledge and can be easily deceived by formal-sounding pseudoproofs.

To address this limitation, we evaluated our step-level verification agent under a strict avg@3 setting, supplemented by multi-pass experiments on these specific adversarial examples. By requiring constructive elaboration and explicit tracing of external knowledge sources, our framework successfully isolated these camouflaged errors, successfully rejecting all invalid proofs. Furthermore, out of the valid proofs, our agent failed to verify only 3 cases. Paradoxically, these failures highlight the meticulousness of our framework. The agent did not fail due to logical reasoning deficits, but because the underlying base model lacked familiarity with highly specific domain conventions, where certain mathematical objects inherently possess implicit properties by default. Consequently, it failed to establish the necessary prerequisite statements for the local context.

For example, in Question 7 of the FirstProof dataset, a human expert naturally assumes that the subgroup Γ is explicitly linear (since it is a subgroup of a real semi-simple Lie group, and by convention within this specific research domain, such groups are treated as linear) (First Proof Project, 2026b; Weinberger, 2022). While the standard baseline simply ignored this subtle leap and glossed over the text, our agent rigorously noted that under the general definition, a subgroup of a real semi-simple Lie group is not necessarily linear by default (Morris, 2015). Because the base model was unfamiliar with this established domain convention, it conservatively refused to arbitrarily add the linearity assumption into the local context, ultimately losing a fundamental condition required to verify the proof. This contrast perfectly demonstrates that our framework enforces genuine deductive verification over superficial pattern matching, even if it halts when confronted with domain-specific conventions it has not yet mastered.

This consistency reveals that the failures of standard LLMs in mathematical verification are not artifacts of sampling noise, but stem from systemic deductive deficiencies.

Baseline Performance and Failure Modes The baseline models—Global GPT-5.4, Global Codex 5.4, and the Sentence-by-Sentence ablation—demonstrated severe systemic vulnerabilities, consistently failing on both valid and invalid proofs across all trials.

- **False Negatives (Accepting Invalid Proofs):** The baselines entirely failed to detect embedded flawed statements. For instance, all three baseline pipelines confidently and repeatedly verified invalid proofs such as `[Aletheia 7A]`. Without a rigorous mechanism to audit external theorems, these models were systematically deceived by formally rigorous but mathematically vacuous text.
- **False Positives (Rejecting Valid Proofs):** Conversely, on the 11 valid proofs, the baselines generated pedantic false positives. They systematically flagged valid proofs such as `[Aletheia 7B, 8B, 9A]` as `FLAWED`, hallucinating non-existent logical gaps or consistently complaining about standard, unstated notational conventions regardless of the sampling randomness.

Agent Performance In contrast to the global baselines, our Constructive Verification Agent achieved robust structural isolation and exhibited a soundness-biased error profile, eliminating false acceptances on invalid proofs while producing consistent final verdicts under the high-temperature setting.

- **Reducing False Negatives:** On the 10 invalid proofs, the agent successfully identified and isolated the fatal deductive flaw in 100% of the cases. By requiring each step to be constructively elaborated and every external theorem source to be explicitly supported, the agent blocked the mathematically invalid steps that bypassed the baseline evaluators.
- **Robustness against False Positives:** On the 11 valid proofs, the agent correctly verified `8` proofs. For the remaining `3` valid proofs (`Aletheia 7B, 10A, 10B`), the agent halted in the `FLAWED` or `OPEN` state due

to genuinely ambiguous, unstated domain conventions. The agent flagged the proof of problem 10 as flawed because it strictly adhered to the mathematical definition of the stated positive semi-definite condition—which permits zero diagonal entries and invalidates direct element-wise inversion—without adopting the human expert's unstated practical convention that typical RKHS kernels inherently possess strictly positive diagonals. Crucially, owing to its explicit Flaw Confirmation mechanism, the agent produced **zero** arbitrary `FLAWED` judgments on fundamentally sound proofs.

When provided with the 7B model's reference solution and the initial problem conditions, our verifier successfully validated the proof (see Appendix C).

5. Limitations and Future Works

While our framework demonstrates the immense potential of LLM-based agents in verifying natural language mathematical proofs, certain limitations provide clear avenues for future research.

Implicit Conventions and Human-in-the-loop First, base LLMs inherently struggle with unstated mathematical folklore, frequently mischaracterizing benign domain-specific notations as explicit flaws. To address this, a highly promising direction is a human-in-the-loop (HITL) collaborative paradigm, where the agent autonomously prunes logical dependencies but dynamically queries human experts when confronting profound epistemic ambiguities.

Residual Hallucinations vs. Formal Guarantees Second, unlike formal theorem provers that offer absolute mechanical guarantees, our purely text-based agent remains susceptible to subtle hallucinations. These can happen when the agent overly simplifies the application of complex, web-retrieved external theorems. While our current system mitigates this by requiring source-aware constructive elaborations and explicit external-theorem support, it does not currently employ a secondary adversarial pass to actively hunt for flaws within its own accepted deductions. Future work will introduce an iterative, adversarial re-verification mechanism to critically audit steps that have already transitioned to the `VERIFIED` state, further minimizing false negatives.

Large-scale Evaluation Finally, while our curated 21-proof diagnostic suite enables deep, qualitative dissection of the agent's cognitive trajectory and failure modes, scaling this evaluation to a massive, multi-domain benchmark of formal and informal mathematics remains an important next step to fully chart the generalization bounds of our architecture.

In conclusion, pure natural language mathematical agents are poised to drive continued breakthroughs in the near future. Our work demonstrates a fundamental cognitive parallel: much like human mathematicians, autonomous agents can elevate their ability to discern valid mathematical reasoning from flawed logic through the rigorous act of detailed elaboration and careful control of external mathematical sources. Ultimately, our contribution extends beyond a standalone verifier implementation. **More crucially, we present a generalizable verification paradigm that can be seamlessly integrated into the internal reasoning processes of any future natural language mathematical agent.**

References

Abouzaid, M., Blumberg, A. J., Hairer, M., Kileel, J., Kolda, T. G., Nelson, P. D., Spielman, D., Srivastava, N., Ward, R., Weinberger, S., and Williams, L. First proof, 2026. URL https://arxiv.org/abs/2602.05192.

Alfieri, A. and Novak, C. Research report: some constructions in polyhedral symplectic topology suggested by ai, 2026. URL https://arxiv.org/abs/2602.13519.

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.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

de Bruijn, N. The mathematical vernacular, a language for mathematics with typed sets. In Nederpelt, R., Geuvers, J., and de Vrijer, R. (eds.), *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pp. 865–935. Elsevier, 1994. doi: https://doi.org/10.1016/S0049-237X(08)70231-3. URL https://www.sciencedirect.com/science/article/pii/S0049237X08702313.

de Moura, L. and Ullrich, S. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G.,

et al. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Feng, T., Jung, J., Kim, S.-h., Pagano, C., Gukov, S., Tsai, C.-C., Woodruff, D., Javanmard, A., Mokhtari, A., Hwang, D., et al. Aletheia tackles firstproof autonomously. *arXiv preprint arXiv:2602.21201*, 2026.

First Proof Project. First batch | first proof project, 2026a. URL https://1stproof.org/first-batch.html. Project page for the first batch of First Proof problems.

First Proof Project. First proof solutions and comments, 2026b. URL https://1stproof.org/documents/FirstProofSolutionsComments.pdf. Official solutions and commentary PDF for the First Proof project.

Fowler, J. Finiteness properties for some rational Poincaré duality groups. *Illinois Journal of Mathematics*, 56(2):281–299, 2012. doi: 10.1215/ijm/1385129948. URL https://doi.org/10.1215/ijm/1385129948.

Ganesalingam, M. The language of mathematics. In *The Language of Mathematics: A Linguistic and Philosophical Investigation*, pp. 17–38. Springer, 2013.

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, 2022. URL https://arxiv.org/abs/2210.12283.

Ju, H., Gao, G., Jiang, J., Wu, B., Sun, Z., Chen, L., Wang, Y., Wang, Y., Wang, Z., He, W., et al. Automated conjecture resolution with formal verification. *arXiv preprint arXiv:2604.03789*, 2026.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, J. and Qin, S. Shokurov's global index conjecture for threefold foliations. *arXiv preprint arXiv:2605.22735*, 2026. URL https://arxiv.org/abs/2605.22735.

Liu, J., Hu, R., and Qin, S. Boundedness of total cartier indices for rational singularities in families. *arXiv preprint arXiv:2605.22782*, 2026a. URL https://arxiv.org/abs/2605.22782.

Liu, J., Sun, Z., and Jiang, J. Optimal bounds in bend-and-break for foliations. *arXiv preprint arXiv:2605.20754*, 2026b. URL https://arxiv.org/abs/2605.20754.

Morris, D. W. *Introduction to arithmetic groups*, volume 2. Deductive Press Lieu de publication inconnu, 2015.

OpenAI. Planar point sets with many unit distances, 2026. https://cdn.openai.com/pdf/74c24085-19b0-4534-9c90-465b8e29ad73/unit-distance-proof.pdf.

The mathlib Community. The Lean mathematical library, 2019. URL https://arxiv.org/abs/1910.09336.

Weinberger, S. *Variations on a Theme of Borel: An Essay on the Role of the Fundamental Group in Rigidity*. Cambridge University Press, 2022. doi: 10.1017/9781316529645. URL https://doi.org/10.1017/9781316529645.

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.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models, 2023. URL https://arxiv.org/abs/2306.15626.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2021. URL https://arxiv.org/abs/2109.00110.

A. Prompts for the Verification Agent

In this appendix, we present the prompts used to drive our state-based verification agent. The prompts are administered to the Codex agent as fixed instruction prompts.

A.1. Instruction Prompts

Prompt 1: Initialization

```
Run a fully automatic verifier in
this folder.  Read only README.md,
STEP_TEMPLATE.txt, OUTPUT_FORMATS.txt,
and source/input.txt.
Split the proof into minimal logical
deduction units.  Every original
source sentence with mathematical
content must appear verbatim in
the Original step field of at least
one step file.  Initialize steps/.
Repeatedly work on the most critical
untouched step, otherwise the most
```

critical open step.
Each step file must be a thick elaboration organized by $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$, with the full original step, the exact target assertion, a detailed deduction, and a diagnosis. Treat every step as requiring its own local closure analysis: never mark a step VERIFIED without individually checking its own written Γ_i, Σ_i, $\mathcal{T}_i$, Deduction, and every used item of Σ_i.
For every used item of Σ_i, write the exact as-used statement and include either a proof or a precise source, plus a brief applicability note. Be conservative: if a used item of Σ_i is not closed by direct local reasoning, or by a proof or precise source that you can state concretely in the step file, keep it OPEN rather than VERIFIED. Do not treat a statement as 'confirmed' merely because it looks standard or plausible.
A step is VERIFIED only if every used item of Σ_i is 'confirmed'; otherwise it is OPEN, unless there is an explicit flaw witness, in which case it is FLAWED. Initialize the verifier now: split the proof into detailed minimal proof-obligation steps, splitting explicit intermediate assertions before downstream consequences.
Maintain only steps/, dependency.txt, stepsprocess.txt, and answer.txt.
Stop only when every step has been judged as VERIFIED, OPEN, or FLAWED.

Prompt 2: Resume

Resume the verifier in this folder.
Read README.md, STEP_TEMPLATE.txt, OUTPUT_FORMATS.txt, dependency.txt, stepsprocess.txt, answer.txt, and all unresolved files in steps/.
If there is a step that is too coarse, resplit the unresolved file that is still too coarse to be one minimal logical deduction unit.
Every original source sentence with mathematical content must appear verbatim in the Original step field of at least one step file. Then continue the main loop.
If a step has remained OPEN since the first pass and still resists closure, consider whether it has an explicit flaw witness and should be FLAWED. For any current OPEN step hard to resolve, consult the local reference/* materials or search online for relevant detailed information, exact statements, or precise sources.
If search online, you should only look for standard text book or references that DIRECTLY CONNECT to the specific problem. DO NOT accept randomly retrieved papers merely to force a proof to work.
Before changing any unresolved step to VERIFIED, recheck that every used item of Σ_i has an exact as-used statement together with either a proof or a precise source. Do not mark a step VERIFIED by replacing the original inference with a different proof or alternative argument. Default negative verdict is OPEN. Use FLAWED only with an explicit flaw witness.

Prompt 3: Flaw Confirm

Investigate whether the FLAWED step in this folder is actually FLAWED.
Read README.md, STEP_TEMPLATE.txt, OUTPUT_FORMATS.txt, dependency.txt, stepsprocess.txt, answer.txt, the unresolved files in steps/, and reference/* if present.
Choose exactly one currently FLAWED step, preferably the most critical one.
Keep the Original step verbatim and keep the same final assertion. First investigate the step: expand the relevant concepts, exact statements, hypotheses, scope, and nearby lemmas in the local reference/* materials and online source, and use this to judge whether the recorded flaw witness shows that the original sentence or exact as-used inference is false or invalid as written.
Do not treat a mere wording, naming, or terminology imprecision as a flaw witness unless it changes the mathematical claim, breaks the local inference, or materially threatens the validity of the proof.
If yes, keep FLAWED. If not, change it to OPEN and explain exactly why the witness fails.

A.2. File Formats and Step Templates

To enforce strict state tracking and prevent the LLM from generating unstructured text, the agent is required to output its intermediate reasoning and global status into highly constrained text files. The formats for these ledgers and templates are defined below.

Prompt 4: Global Ledgers (OUTPUT_FORMATS.txt)

```
[dependency.txt]
List only external statements that are
still OPEN or already FALSE.
Format:
D1
- step(s):
- exact as-used statement:
- status:  OPEN / FALSE
- why:
- offending point if false:

[stepsprocess.txt]
For each step:
Step:
- status:  UNTOUCHED / OPEN / VERIFIED
/ FLAWED
- minimal unit:  yes / no
- criticality:
- main blocker:
- all used sigma closed:  yes / no
- flaw witness:
- next action:

[answer.txt]
Format:
Overall verdict:  RUNNING / VERIFIED /
FLAWED
Current key step:
If FLAWED:
- offending step:
- exact false statement or invalid
inference:
- flaw witness:
If still running:
- remaining open steps:
- active dependencies:
Per-step summary:
- step:
- status:
- one-line reason:
Rule:
Do not use FLAWED unless an explicit
flaw witness has been written.
Do not use VERIFIED unless all
used external statements are marked
'confirmed'.
```

Prompt 5: Logical Unit Template (STEP_TEMPLATE.txt)

```
Step ID:
Why this is one minimal logical
deduction unit:
Original step:
[paste the full original step string
verbatim]
Target assertion a_i:
Original route:
Explanation:
[write a thick natural-language
elaboration that preserves the same
assertion and route]
Statement Extraction (Γ_i, Σ_i, T_i):
Assertion:
Local Context Γ_i:
- γ_1:
- γ_2:
Global Knowledge Σ_i:
- σ_1:
   exact as-used statement:
   proof or precise source:
   why it applies here:
   status:  confirmed / OPEN / false
- σ_2:
Sigma Support:
[for each used σ_j, include either a
proof or a precise source]
Background Theory T_i:
- t_1:
- t_2:
Deduction:
[derive the assertion from Γ_i,
confirmed Σ_i, and T_i; if some used σ_j
is still OPEN, state that the route is
only conditionally closed]
Diagnosis:
- conclusion preserved:
- route preserved:
- exact offending dependence, if any:
- flaw witness, if any:
Step Result:
- status:  UNTOUCHED / OPEN / VERIFIED
/ FLAWED
- main blocker:
- flaw witness:
- dependencies:
- last update:
```

B. Extended Discussion on the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ Framework and Explanation Validity

Consider a natural language proof text S and its ordered sequence of steps (substrings) $(s_1, s_2, \ldots, s_n)$. Without loss of generality (by carefully choosing the substrings for steps and using conjunction) , we restrict that a step contains at most one new mathematical statement. Throughout the paper, we consider assumptions and declarations both as mathematical statements.

Let s_i be a deduction step of S. Let the statement a_i be the assertion of s_i. Unlike fully formalized proofs in systems like Lean or Rocq, natural language mathematical proofs inherently omit trivial algebraic manipulations and elementary logical deductions to maintain readability. In this paper, we do not require the LLM to elaborate proofs down to atomic axiomatic steps, as this would lead to an unnatural combinatorial explosion of reasoning length and degrade model performance. When defining the assertion a_i extracted from a natural language proof,

we adopt a stance of foundational agnosticism. Modern informal mathematics—ranging from elementary calculus to advanced subjects like Topos theory or Wiles' proof of Fermat's Last Theorem—is rarely written with a strict, singular axiomatic foundation (e.g., pure ZFC) in mind. Therefore, we do not restrict a_i to a specific foundational axiomatic system. Rather, we assume the existence of a sufficient, implicit foundational theory that is universally accepted for the specific domain of the proof.

We must also address the validity of natural language statements that appear underspecified in isolation, such as the simple assertion $a = b$. In strict formal systems, an equality requires explicit type alignment (e.g., verifying that both sides belong to the same algebraic structure). In well-formed (which we may assume) natural mathematical discourse, however, such type constraints are rarely restated at every step; instead, they are implicitly inherited from the ambient context (captured by Γ_i in our framework). As long as a statement is semantically well-posed within this context, it operates as a valid deductive step. Demanding explicit syntactic completeness for every variable introduces an unnatural overhead that misaligns with human communication. Ultimately, if we accept a natural language proof as a valid logical artifact, we must recognize that its constituent statements—though syntactically incomplete in isolation—are rigorously well-defined when situated in the proof's narrative flow. Crucially, this approach does not imply an omission of type verification. Rather, by isolating type-related statements, we can safely and cleanly decouple the structural type constraints from the core deductive logic, making the natural language step significantly easier to evaluate.

For a practical way to formalize the elaboration process, we define an **explanation** of the step s_i as a triple of sets $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$, which encompasses all the necessary statements required to logically derive the assertion a_i. Thus for any elaboration s'_i (in natural language), there exists an explanation $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ that **explains** s'_i. In this context, we say the triple **explain**s the elaboration if every critical mathematical assertion articulated within the natural language elaboration and the step is subsumed either directly by the sets of the triple or by the intermediate statements deduced during the derivation process. Moreover, the derivation logically entailed by this triple must strictly adhere to the specific computational or deductive methodology dictated by the natural language text. Γ_i contains finitely many statements (theorems, lemmas, definitions, and hypothesis) from S. The statements within Γ_i may either be explicitly stated in S or left implicit. Σ_i comprises a finite set of statements drawn from external literature or established domain consensus, that is, outside of S. Here statements from Σ_i are considered non-trivial. Within an explanation of s'_i, there may exist statements from Σ_i that are implicitly assumed rather than explicitly articulated in s'_i. $\mathcal{T}_i$ comprises a finite set of what are often considered trivial statements (or rewrite rules) representing the background theory. It assists Γ_i and Σ_i in syntactically deriving the assertion a_i. Fundamentally, our framework aims to model any detailed elaboration of a proof step. Consequently, any mathematical statement invoked in such an expanded derivation must necessarily originate from this triple.

When we say a step s_i is valid, we usually mean there exists an explanation $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ of s_i such that:

$$\Gamma_i, \Sigma_i, \mathcal{T}_i \vdash a_i$$

Here $\vdash$ denotes a rigorous informal entailment. Also, all $\Gamma_i, \Sigma_i, \mathcal{T}_i$ are valid. For Γ_i to be valid, all the statements it contains should first be effective in the step s_i. Statements in S has their scope of effectiveness. A context statement (an assumption or a declaration) has its scope: assumptions and variables are no longer valid from a certain point onwards. Consequently, s_i may also fall outside the scope of other preceding mathematical statements, including lemmas and theorems. For example, if the proof contains a case analysis, then for every case the proof establishes a block with the corresponding assumption. The assumption's scope terminates once the discussion for the case is over. The theorems and lemmas proved in the case may also be invalid for the next case, if local assumptions and declarations are used. Second, any implicit statements within Γ_i must be validly inferred from the context. The admissibility of such implicit elements is highly sensitive to both the active discourse and the established consensus of the mathematical community. Consequently, LLMs should be able to correctly identify valid implicit statements based on the context. Finally, since Γ_i contains deductions and theorems from the context, the statements of Γ_i should all be valid. For Σ_i to be valid, all statements within it must be drawn from universally accepted mathematical facts or sound external literature.

We now explain the validity of $\mathcal{T}_i$. Since $\mathcal{T}_i$ captures the routine background steps skipped by human writers, it provides a practical interface between informal mathematical text and formal axioms. There is an immediate problem of what is considered as a nontrivial statement and has to be written explicitly. Since the explanation here actually corresponds to expanded derivations of s_i, the threshold for non-triviality can be carefully calibrated. Also, for $\mathcal{T}_i$ to be valid, all the statements it contains should be logically sound. Throughout this framework, we operate under the standard assumption of soundness for the underlying mathe-

matical foundation. Let us now consider a simple example of a proof and two possible explanation of different steps. Note that by admitting the validity of the local context in eliminating ambiguity, we can omit type declarations for the assertions and the other statements.

Case Study: Deconstructing the Proof of $\sqrt{2} \notin \mathbb{Q}$

Background Proof Context:
Theorem: $\sqrt{2}$ is an irrational number.
Step 1 (s_1): *Assume for the sake of contradiction that $\sqrt{2}$ is a rational number.*
Step 2 (s_2): *By the definition of rational numbers, there exist integers a and b (where $b \neq 0$) such that $\sqrt{2} = \frac{a}{b}$.*
Step 3 (s_3): *We can further assume that a and b are coprime (i.e., the fraction $\frac{a}{b}$ is in its simplest form, sharing no common factors other than 1).*
Step 4 (s_4): *Squaring both sides of the equation gives $2 = \frac{a^2}{b^2}$.*
Step 5 (s_5): *Multiplying both sides by b^2 yields $2b^2 = a^2$.*
Step 6 (s_6): *Since b is an integer, b^2 is an integer, making $2b^2$ an even number. Therefore, a^2 must be an even number.*
Step 7 (s_7): *If the square of an integer is even, then the integer itself must be even. Thus, a is an even integer.*
Step 8 (s_8): *Since a is even, there exists an integer k such that $a = 2k$.*
Step 9 (s_9): *Substituting $a = 2k$ into the equation from Step 5 ($2b^2 = a^2$) gives $2b^2 = (2k)^2 = 4k^2$.*
Step 10 (s_{10}): *Dividing both sides by 2 yields $b^2 = 2k^2$.*
Step 11 (s_{11}): *Following the same logic as before, this implies that b^2 is an even number.*
Step 12 (s_{12}): *Consequently, b must also be an even number.*
Step 13 (s_{13}): *Since both a and b are even numbers, they share a common factor of at least 2.*
Step 14 (s_{14}): *This directly contradicts our assumption in Step 3 that a and b are coprime.*
Step 15 (s_{15}): *Therefore, our initial assumption in Step 1 must be false, concluding that $\sqrt{2}$ is irrational.*

Step 7 (s_7):

- **Assertion (a_7):** a is an even integer.
- **Local Context (Γ_7):** $\{a^2 \text{ is even}\} \cup \{a \in \mathbb{Z}\}$ *(Note: $a \in \mathbb{Z}$ is an implicit statement)*
- **Global Knowledge (Σ_7):** $\{\forall n \in \mathbb{Z}, n^2 \text{ is even} \implies n \text{ is even}\}$
- **Background Theory ($\mathcal{T}_7$):** $\emptyset$ *(Direct application of Modus Ponens)*

Step 9 (s_9):

- **Assertion (a_9):** $2b^2 = (2k)^2 = 4k^2$.
- **Local Context (Γ_9):** $\{a = 2k, \quad 2b^2 = a^2\}$
- **Global Knowledge (Σ_9):** $\emptyset$ *(No external lemmas invoked)*
- **Background Theory ($\mathcal{T}_9$):** Equivalence properties of equality (e.g., symmetry, transitivity, and substitution/congruence) combined with fundamental properties of integer arithmetic (e.g., commutativity, associativity, and evaluation of ground constants such as $2 \times 2 = 4$).

We now analyze the benefits of not restricting $\mathcal{T}_i$ to rigid, hardcoded rules, primarily regarding its interface with foundational axioms. Although we do not specify a definitive axiomatic system, our framework inherently assumes universally permitted operations like Modus Ponens. This universal compatibility affords us considerable flexibility in formally specifying $\mathcal{T}_i$. Furthermore, the flexibility in specifying $\mathcal{T}_i$ drastically reduces the deductive overhead within $\vdash$ and minimizes the total number of explicit statements required. For example, in formal libraries like Lean's `Mathlib`, performing basic integer arithmetic typically necessitates instantiating deep algebraic hierarchies, such as commutative rings. In contrast, allowing all kinds of sound statements bypasses this formal complexity, enabling the system to directly reason with sound statements over $\mathbb{Z}$. Furthermore, since the operations within $\mathcal{T}_i$ are inherently routine, we can leverage this flexibility to formulate statements directly about the concrete objects, bypassing the need for explicit variable substitution. This also mirrors human conventions in mathematical discourse: when executing elementary transformations, we naturally manipulate the expressions directly. Consequently, the elements within $\mathcal{T}_9$ are concrete statements like $(2k)^2 = 2k \cdot 2k$, rather than abstract, uninstantiated schemas such as $x^2 = x \cdot x$.

Note that the explicit type declaration $a^2 \in \mathbb{Z}$ is omitted from Γ_7 for the same reason we bypass redundant variable bindings. By the ambient context, we establish $a \in \mathbb{Z}$. And since $\{\forall n \in \mathbb{Z}, n^2 \text{ is even} \implies n \text{ is even}\}$ is an unambiguous well-established statement under the current context, we don't need to write $a^2 \in \mathbb{Z}$ here. Consequently, an explanation $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ should not be viewed as the complete set of all contextually valid truths, but exclusively as the minimal required set of statements necessary to validate the natural language proof step rigorously.

In the example above, we directly structured an elabora-

tion using the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ format. In what follows, we demonstrate that given an elaboration s'_i of a proof step, we can extract its constituent statements and systematically categorize them into this triple framework. Then, find an explanation that explains s'_i, though the explanation may not be valid. Consider the following example from Firstproof generated by Aletheia.

Case Study: Extraction Of Statements

Problem and Proof Sketch

Problem: Suppose that Γ is a uniform lattice in a real semi-simple group, and that Γ contains some 2-torsion. Is it possible for Γ to be the fundamental group of a compact manifold without boundary whose universal cover is acyclic over the rational numbers $\mathbb{Q}$?

Proof: The generated proof constructs a logical contradiction centered around the compactly supported Lefschetz number, $L_c(\gamma, X)$. The model computes this topological invariant through two divergent pathways. Algebraically, it correctly applies Poincaré duality alongside the $\mathbb{Q}$-acyclicity of X to deduce that $H_c^*(X; \mathbb{Q})$ is concentrated exclusively in the top degree, cleanly evaluating to $L_c(\gamma, X) = \pm 1 \neq 0$. Topologically, the model evaluates the 2-sheeted regular covering $X \to Y = X/\langle\gamma\rangle$ using the transfer map, arriving at the valid relation $\chi_c(Y) = \frac{1}{2}(\chi_c(X) + L_c(\gamma, X))$. However, to mathematically force a contradiction, the model introduces a critical hallucination: it erroneously asserts that the compactly supported Euler characteristic is strictly multiplicative (i.e., $\chi_c(X) = 2\chi_c(Y)$) for infinite cell complexes. By substituting this flawed generalization into the transfer equation, the generated proof strictly forces $L_c(\gamma, X) = 0$, resulting in the false algebraic contradiction $0 = \pm 1$.

Step s_i (Flawed)

"However, for any finite k-sheeted unbranched covering of locally compact ENRs with finite Betti numbers, the compactly supported Euler characteristic is strictly multiplicative. Thus, $\chi_c(X) = 2\chi_c(Y)$."

Explanation generated by LLM

The author introduces a general rule regarding covering spaces. They assert:

> *"for any finite k-sheeted unbranched covering... the compactly supported Euler characteristic is strictly multiplicative."*

Expressed as a mathematical formula, the author is asserting that for any total space E and base space B connected by a k-sheeted covering map, the relationship is:

$$\chi_c(E) = k \cdot \chi_c(B)$$

To use this general formula, the author must establish the specific values for E, B, and k based on the geometry of the spaces in the proof.

- **The Total Space (E):** The author is using X (the universal cover of the manifold M) as the total space.
- **The Base Space (B):** The author defines the base space Y as the quotient $X/\langle\gamma\rangle$.
- **The Sheet Count (k):** The author establishes that γ is a torsion element of order 2 ($\gamma^2 = \text{id}$). Because γ generates a strictly free action of $\mathbb{Z}/2\mathbb{Z}$ on X, the projection map from X down to the quotient Y is a regular covering map. Because the group acting on it has exactly 2 elements, the number of "sheets" (the size of the fiber over any point in Y) is exactly 2. Therefore, $k = 2$.

The author takes the specific parameters established in Step 2:

- $E \to X$
- $B \to Y$
- $k \to 2$

And substitutes them directly into the general formula from Step 1 ($\chi_c(E) = k \cdot \chi_c(B)$).

This direct application of the parameters to the multiplicative rule yields the final assertion:

$$\chi_c(X) = 2\chi_c(Y)$$

That is the entirety of the deduction for that specific step. The author simply proves the covering is 2-sheeted and plugs $k = 2$ into their (flawed) multiplicative formula.

Statement Extraction $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$

Assertion (a_i): $\chi_c(X) = 2\chi_c(Y)$.

Local Context (Γ_i):

γ_1 : The projection $X \to Y$ is a 2-sheeted regular covering map, generated by a strictly free group action.

γ_2 : $H_c^*(X; \mathbb{Q})$ and $H_c^*(Y; \mathbb{Q})$ are finite-dimensional.

γ_3 : X is an n-dimensional topological manifold. Y is a topological manifold.

All of them are explicit.

Global Knowledge (Σ_i):

σ_1 : *[Flawed lemma]* For any finite k-sheeted unbranched covering of locally compact ENRs with finite Betti numbers, the compactly supported Euler characteristic is strictly multiplicative.

σ_2 : Every finite-dimensional topological manifold is a locally compact ENR.

σ_3 : A topological manifold has finite compactly supported Betti numbers if its compactly supported rational cohomology is finite-dimensional.

σ_4 : If $X \to Y$ is a covering map between manifolds, then $\dim(Y) = \dim(X)$.

Background Theory ($\mathcal{T}_i$):

t_1 : The property of 'the Euler characteristic being strictly multiplicative' is defined by the following equation under algebraic rewrite rules: $\chi(\text{total space}) = k \cdot \chi(\text{base space})$, where k is the number of sheets of the covering.

t_2 : A regular covering map generated by a strictly free group action is an unbranched covering.

t_3 : A k-sheeted covering is finite.

Deduction $\vdash$

Since γ_1 establishes a covering map $X \to Y$ and γ_3 identifies both as manifolds, σ_4 confirms they share the same dimension n, which by σ_2 ensures both are locally compact ENRs. Concurrently, because γ_2 confirms their com-

pactly supported rational cohomology is finite-dimensional, σ_3 guarantees both possess finite compactly supported Betti numbers. Given that γ_1 specifies the action is strictly free, t_2 classifies it as unbranched, and since it is a 2-sheeted covering, t_3 confirms it is finite k-sheeted ($k = 2$). With all prerequisites of the flawed lemma σ_1 met (a finite k-sheeted unbranched covering of locally compact ENRs with finite Betti numbers), we apply t_1 with $k = 2$ to yield the final assertion a_i: $\chi_c(X) = 2\chi_c(Y)$.

The green highlighted part corresponds to σ_1. The yellow highlighted statement corresponds to t_1. The blue highlighted statement corresponds to γ_1. Notice a few things here. First, we ignore the statements "γ generates a strictly free action of $\mathbb{Z}/2\mathbb{Z}$ on X," and "the group acting on it has exactly 2 elements." But we still consider our $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ explains the elaboration. By definition, the explanation should contain every critical mathematical assertion articulated within the natural language text. Thus, by considering the above two statements to be not critical (which is reasonable), we can simplify our $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$. Given the inherent linguistic variations and rhetorical scaffolding present in human-authored proofs, the extraction of critical statements from natural language inherently demands a degree of flexibility. Second, we did not take arguments like E is X into account. This is also for simplification. By the same manner of the first example, we omit the detail of Modus Ponen. The generated elaboration doesn't verify all required conditions (for example, finite Betti number) for the flawed theorem, thus we can fill them up in the explanation. Though the deduction process $\vdash$ of the explanation is sound, the statements it contains are not valid, making the explanation itself not valid. For certain steps, the logical deduction is sufficiently explicit that it requires almost no hidden calculations, rendering the corresponding elaboration essentially deterministic in practice.

In summary, this work seeks to analyze the rigor of natural language mathematical proofs. We do not attempt to enforce the absolute strictness required by systems like the Mathematical Vernacular (MV) or formal proof assistants. Empirically, however, we demonstrate that subtle logical errors can be successfully localized simply by writing out the proof in granular detail. The critical factor in this approach is maintaining strict faithfulness to the author's original intent. Ultimately, our theoretical framework serves primarily as philosophical guidance for the agent's reasoning, rather than a rigid requirement to perfectly instantiate every formal explanation. As our experiments confirm, the mere act of forcing the agent to elaborate in detail intrinsically elevates its verification performance.

C. Case Study: The Boundary of Implicit Conventions (Aletheia 7A & 7B)

To gain deeper insights into the agent's cognitive trajectory, we present a detailed comparative case study on the proofs `Aletheia 7A` (invalid) and `Aletheia 7B` (valid).

The original text of `7A` actually consists of two separate proofs: the first is an invalid pseudoproof containing a highly concealed, fatal error concerning "multiplicity", while the second fundamentally relies on the first. For GPT-5.4, formulating a precise disproof for this specific error is difficult, as explicit counterexamples are scarcely available via web retrieval and fall outside the model's internal parametric knowledge. Furthermore, within the specific scope of First-Proof Problem 7, it is an accepted academic convention to assume that semisimple Lie groups are linear—an assumption that does not strictly hold in general mathematical contexts. Consequently, an unguided agent is highly susceptible to being distracted by this linearity assumption (generating a false flaw witness) or by downstream errors in the second proof.

To stress-test our agent's ability to identify the true structural error, we manually truncated `7A` to include only the first proof and explicitly appended the linearity condition. Across 10 independent runs, the agent yielded one false negative (marking the flawed step as `VALID` during initialization rather than checking it carefully) and successfully rejects the proof 9 times, marking the specific step `OPEN` or `FLAWED`. However, a closer inspection reveals a critical logical subtlety: the agent justified its `FLAWED` verdict by pointing out that the multiplicity claim explicitly contradicted the local context. While this contradiction genuinely exists, penalizing the step strictly on these grounds is mathematically unreasonable here, as the step resides within a proof by contradiction (*reductio ad absurdum*)—where deriving absurdities is the expected trajectory. This highlights the profound difficulty in distinguishing between an invalid inference rule and a valid derivation of a contradiction.

Conversely, our experiments on the fundamentally valid proof `Aletheia 7B` perfectly illuminate the agent's underlying verification power and its dynamic self-correcting capabilities. Though the agent fails to validate the proof in the 3 runs, when we evaluated `7B` by explicitly providing the linearity assumption and the reference solution sketch, the agent did not simply rubber-stamp the text. Instead, it exhibited a rigorous, non-linear cognitive trajectory.

During the verification process, the agent initially proposed two potential flaw witnesses that threatened to invalidate the proof. In one of them, the agent's web retrieval module returned an alternative formulation of a specific series expansion. Detecting a difference from the retrieved literature, the agent instinctively marked the author's step as

FLAWED. The action is then rejected by our *Flaw Confirmation* mechanism. Engaged in adversarial self-reflection, the agent re-evaluated the discrepancy and recognized that the mere existence of a different formalization online does not mathematically invalidate the author's specific local derivation. Consequently, the agent correctly dismissed its own false alarm, averting a premature false positive.

After overriding these superficial discrepancies and conducting further targeted retrieval, the agent meticulously verified the highly non-trivial claims regarding KO-theory (real topological K-theory) and successfully closed the proof.

This juxtaposition offers a more grounded observation: when explicit premises are provided, the agent is capable of processing complex mathematical concepts like K-theory. More importantly, this iterative pattern—searching for external literature, raising potential issues, and self-correcting—demonstrates a practical pathway for scaling test-time compute in mathematical verification. By allocating more inference steps to information gathering and adversarial reflection, the system can progressively resolve textual ambiguities. However, the agent's frequent stalling on implicit human conventions serves as a humbling reminder. It indicates that while deductive logic and external retrieval are necessary, they are not yet sufficient to fully bridge the gap between strict formalization and the nuanced, context-dependent nature of human mathematical communication.

D. Prompts for the Sentence-by-Sentence Ablation Baseline

In this section, we present the prompts used for the **Sentence-by-Sentence** ablation baseline discussed in Section 4.1.

Unlike our main Constructive Verification Agent, which requires source-aware constructive elaboration and maintains an explicit external theorem ledger (implemented through the $(\Gamma_i, \Sigma_i, \mathcal{T}_i)$ scaffold), this ablation baseline is designed to test the effect of mere textual granularity. Specifically, it instructs the agent to break the proof down into sentence-level steps but only requests a "plain naive analysis in prose" without rigid dependency tracking. This separates gains from mere textual granularity from those due to explicit dependency/source tracking and theorem-use control. To ensure a fair and controlled ablation, the overall execution pipeline—including the reflection and flaw confirmation prompts—remains identical to our primary version.

Ablation Prompt 1: Initialization

```
Run the verifier in this folder.  Read
only README.md and source/input.txt.
Split the proof into small steps.
Create one .txt file in steps/ for
each step.  Every original source
sentence with mathematical content
must appear verbatim in at least one
step file.  Initialize or refresh
steps/, stepsprocess.txt, and
answer.txt.
On the first pass, do not read
source/theory.txt, source/example.txt,
source/standards.txt, or reference/*.
Work step by step.  In each step
file, keep the original text and
then write a plain naive analysis
in prose of whether the step closes,
what is missing, or whether there
is an explicit flaw witness.  Be
conservative:  if the step is not
convincingly closed, keep it OPEN.
Use FLAWED only with an explicit flaw
witness.
Maintain only steps/, stepsprocess.txt,
and answer.txt.  Stop only when every
step has been marked VERIFIED, OPEN, or
FLAWED.
```

Ablation Prompt 2: Resume

```
Resume the verifier in this folder.
Read README.md, stepsprocess.txt,
answer.txt, and all unresolved
files in steps/.  If a step is
still too coarse, split it smaller.
Every original source sentence with
mathematical content must appear
verbatim in at least one step file.
Continue step by step.  If a step has
remained OPEN since the first pass,
consider whether there is an explicit
flaw witness and it should be FLAWED.
For a hard OPEN step, you may consult
reference/* or search online for
relevant details.  Before changing any
unresolved step to VERIFIED, recheck
that the step really closes.
Default negative verdict is OPEN.
Use FLAWED only with an explicit
flaw witness.  Maintain only steps/,
stepsprocess.txt, and answer.txt.
```

E. Implementation Details and Hyperparameters

To ensure full reproducibility of our empirical results, we detail the exact execution environment and API parameter configurations utilized for our framework.

Execution Environment. All experimental evaluations were conducted within the `codex` framework (version

0.124.0). We strictly utilized the framework's default configuration profile for environmental setup. Beyond this basic scaffolding, all complex agentic behaviors, formatting rules, and state-machine constraints were injected directly into the GPT-5.4-xhigh engine via natural language system prompts. This design avoids overfitting to specialized framework configurations and explicitly demonstrates the out-of-the-box generalizability of our verification approach.

LLM Generation Hyperparameters. To maintain a rigorously fair comparison, identical generation parameters were applied to the GPT-5.4-xhigh API calls across all baselines and our agentic pipeline. We deliberately operated at a standard temperature of 1.0 to simulate natural generation variance. The fact that our agent yields deterministic, monotonic convergence under these high-entropy sampling conditions further corroborates the robustness of our prompt-driven verification workflow. The complete parameter suite governing the LLM API requests is summarized in Table 2.

Table 2. Complete hyperparameter configurations applied directly to the GPT-5.4-xhigh API endpoints across all verification pipelines.

Hyperparameter	Value
Temperature	1.0
Top-p	1.0
Top-k	0
Min-p	0.0
Top-a	0.0
Frequency Penalty	0.0
Presence Penalty	0.0
Repetition Penalty	1.0

F. A Worked Trace of a Successful Verification Run for Problem 7b

This appendix records a representative successful verification trace for Problem 7b, based on the segment of the execution log that culminates in a fully verified proof state. The point of this example is not that the agent generated a correct proof in one shot. Rather, the log shows a staged verification process in which the agent repeatedly refines the proof graph, isolates unresolved dependencies, revises an incorrect intermediate diagnosis, and only then closes the remaining steps.

The successful portion of the run is the segment in which the global status eventually changes to `Overall verdict: VERIFIED`. Within that segment, the agent's behavior is best understood as a sequence of state transitions over a ledger of proof obligations, rather than as a single forward derivation.

F.1. Initial Reduction of the Open Proof State

At the beginning of the successful segment, the proof is not globally closed. The agent first strengthens the verification of the real assembly injectivity step, recorded as `step_15a_real_assembly_injectivity`. The key effect of this move is structural: it eliminates one major upstream dependency and reduces the remaining unresolved part of the proof to the signature-package portion.

The agent then observes that the existing step `step_15b1_signature_equals_assembly` is too coarse. In its own diagnosis, this step mixes two logically different bridges:

(i) the *assembly/index bridge*, namely that the analytic symmetric signature is the assembly image of a signature class; and
(ii) the *characteristic-class bridge*, namely that this signature class corresponds rationally to the Poincaré dual of the Hirzebruch L-class.

Instead of keeping this as one unresolved block, the agent explicitly resplits it into smaller units:

- `step_15b1a_signature_class_to_assembly`
- `step_15b1b_signature_class_to_lclass`
- `step_15b1c_signature_equals_assembly`

It also rewires the downstream chain so that each later step depends on the newly separated obligations rather than on one monolithic unresolved assertion:

```
step_15b2_m0_pushforward_formula →
    step_15c_lclass_equality →
        step_16_degree_pm1 →
    step_17_final_contradiction
```

This resplitting is the first decisive move in the successful run. It turns a vague unresolved block into a small dependency graph with identifiable root causes.

F.2. Compression to Two Genuine External Gaps

After the split, the proof still contains several `OPEN` steps, but the agent recognizes that these are largely downstream consequences of only two genuine external gaps, recorded in the dependency ledger as `D6b1a` and `D6b1b`. Informally:

- **`D6b1a`:** the real KO-theoretic assembly/index bridge.
- **`D6b1b`:** the real KO-homological L-class bridge.

This is an important feature of the trace. The raw number of open files is not itself the right complexity measure. Once the proof has been resplit correctly, the open graph becomes shallow: many open steps are merely inherited consequences of a very small number of unresolved inputs.

F.3. An Incorrect Flaw Diagnosis and Its Retraction

The run then exhibits a nontrivial self-correction. The agent temporarily marks `step_15b1c_signature_equals_assembly` as `FLAWED`. The reason given is a purported contradiction with a Rosenberg–Weinberger normalization theorem, which seemed to show that the exact real KO-theoretic formula written in the step could not hold with coefficient 1 and the Hirzebruch L-class exactly as written.

This is not the final diagnosis. A separate follow-up phase is launched specifically to determine whether the recorded flaw witness really implies that the original step is false as used. This matters because the verification protocol distinguishes sharply between:

- a statement that is *genuinely false*;
- a statement that may still be true, but whose current justification is *incomplete or misaligned* with the cited source.

Upon re-reading the relevant source, the agent concludes that the previous witness targeted the signature-operator transformation s_n, whereas the proof step under inspection was using a symmetric-signature route. Therefore the witness did **not** actually refute the exact as-used inference. The result is a rollback from `FLAWED` to `OPEN`.

This retraction is one of the most informative parts of the trace. It shows that the agent is not merely searching for counterexamples, but is also checking whether a counterexample hits the correct formal target. In other words, the system distinguishes a failed proof route from a failed theorem statement.

F.4. Closure of the Two Remaining Root Dependencies

After retracting the incorrect flaw diagnosis, the agent returns to the two root gaps.

1. Closing `D6b1a`. The agent verifies `step_15b1a_signature_class_to_assembly`. The successful route here is to replace an indirect complex-K-theoretic justification with a more exact real KO-theoretic source: Rosenberg and Weinberger's Lipschitz-signature theorem is used to construct the real KO-homology signature class and to identify its assembly image with the symmetric signature. Once this step is marked `VERIFIED`, the remaining active dependency set shrinks from {`D6b1a`, `D6b1b`} to just {`D6b1b`}.

2. Closing `D6b1b`. The agent closes this gap by rewriting `step_15b1b_signature_class_to_lclass`. Here the successful move is not simply to cite a missing theorem verbatim. Instead, the agent builds a local bridge:

(1) identify the real signature class in KO-homology;
(2) complexify it to the standard complex signature class;
(3) invoke the known complex K-homological L-class formula;
(4) use rational real/complex comparison to recover the required real KO-homology coordinates.

This is the step at which the open graph finally collapses. Once `step_15b1b_signature_class_to_lclass` is verified, the recombination step `step_15b1c_signature_equals_assembly` also becomes verifiable, since its two incoming components are now both closed.

F.5. Propagation to the Final Contradiction

After the two root dependencies are closed, the rest of the proof is discharged by dependency propagation rather than by new conceptual discoveries.

- `step_15b1c_...` is verified by recombining the now-verified assembly/index bridge with the L-class bridge.
- `step_15b2_...` is verified by specializing the base formula to M_0 and X_0.
- `step_15c_...` is verified by using injectivity of the real assembly map to pull back the signature equality.
- `step_16_...` is verified by reading off the top-degree consequence, namely that $D = \pm 1$.
- `step_17_...` is finally verified by combining this conclusion with the already verified evenness statement for D.

(Note: Long file names are abbreviated here for narrative flow, while remaining rigorously tracked in the system ledgers.)

At that point, the global ledger is updated to `remaining open steps: none` and `Overall verdict: VERIFIED`. The run concludes with a consistency pass over the step files, the dependency ledger, the process summary, and the final answer file to ensure that no stale `OPEN` or `FLAWED` status remains.

F.6. Interpretation

This trace is useful because it illustrates what a successful agentic proof-verification run actually looks like in practice. It is not a single-pass proof synthesis. Instead, it has four characteristic ingredients:

Phase 1: State Decomposition: A coarse unresolved claim is split into minimal logical units.

Phase 2: Dependency Localization: Many apparent open steps are reduced to a very small set of root gaps.

Phase 3: Error Correction: An intermediate `FLAWED` di-

agnosis is explicitly re-examined and retracted when the witness is found to miss the exact target statement.

Phase 4: **Closure by Propagation:** Once the root gaps are closed, the remaining steps verify in sequence through the dependency graph.

For this reason, the `7B` example should not be described as a case where the agent simply "found the proof." A more accurate description is that it executed a ledger-guided verification procedure: it refined the proof graph, isolated the true bottlenecks, corrected a mistaken local refutation, and only then propagated the repaired information to a fully verified final contradiction.