MLP-BENCH: TOWARDS END-TO-END FORMAL RESEARCH ASSISTANCE IN MACHINE LEARNING THEORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated strong mathematical reasoning and autonomous research abilities, yet whether they can assist the rigorous construction and verification of machine learning (ML) theory remains underexplored. Since ML theory rests on precise assumptions and derivations, we ground our study in Lean, which mechanically checks every derivation against its stated assumptions. Mirroring how theory is actually developed and refereed, we frame formal ML theory research as two complementary capabilities: *forward construction*, which assists theory derivation, and *backward verification*, which assists reviewing. We instantiate this framework in the **M**achine Learning **L**ean **P**roof Benchmark (**MLP-Bench**), a 360-problem benchmark spanning 16 topics. *Forward construction* is realized by two tasks, formalizing natural-language (NL) theorems into Lean statements and proving given Lean statements. *Backward verification* is realized by a third task, judging whether a proof is correct and, if not, localizing the first erroneous step with Lean-checked evidence. Evaluating 11 models, *forward construction* remains challenging. Auto-formalization reaches at most a **27.5%** bidirectional equivalence (BEq) pass rate against gold Lean statements, with omitted implicit assumptions a major failure mode, and specialized provers solve **0.0%** of theorem-proving problems. In *backward verification*, models localize errors more reliably by constructing Lean-checked refutation than by attempting to prove each step. To our knowledge, MLP-Bench is the first Lean benchmark for end-to-end LLM assistance in ML theory research.

1 INTRODUCTION

Large language model (LLM) agents have been widely used for assisting scientific research. Idea-generation agents propose hypotheses and identify promising research directions (Lu et al., 2024). These ideas can then be implemented by coding agents, which run experiments, debug them, and inspect the results (Huang et al., 2024b; Schmidgall et al., 2025). At the other end of the pipeline sit review agents, which rely on the surrounding literature to assess a paper along multiple dimensions (Liu et al., 2026; Zhu et al., 2025). However, these efforts largely center on empirical research, and whether LLMs can assist in rigorously constructing and verifying theoretical results remains unclear. The gap matters because the two kinds of claims are checked differently: an empirical claim can be re-checked by rerunning an experiment, whereas a theoretical claim is trustworthy only when every assumption holds and every logical step is sound. Although LLMs show strong mathematical reasoning, they still produce subtle logical errors and struggle to repair them without external feedback (Lewkowycz et al., 2022; Huang et al., 2024a). Assessing whether LLM agents can rigorously assist theoretical ML research end-to-end, from constructing theorems to verifying them, is therefore an important but unaddressed problem.

Lean 4 (de Moura & Ullrich, 2021) offers a rigorous testbed for this problem. Its kernel checks every assumption and step, and its Mathlib library (The mathlib Community, 2020) supplies a broad base of formalized mathematics to build on. Recent frontier models have used it to formalize a claimed resolution of the Navier–Stokes Millennium Prize Problem (OpenAI, 2026) and an existing proof of Fermat's Last Theorem (Anthropic, 2026), underscoring its growing role in assisting verifiable theoretical research. However, existing Lean benchmarks largely target competition problems, text-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Comparison with representative proof-reasoning benchmarks. ✓ marks an explicitly targeted capability; "research-level" denotes realistic research or graduate-level material.

Benchmark	Source	Auto-formalization	Theorem-proving	Proof-verification	Research-level	ML theory
ProofNet (Azerbayev et al., 2023)	Textbooks	✓	✓	×	×	×
LeanDojo (Yang et al., 2023)	Formal libraries	×	✓	×	×	×
PutnamBench (Tsoukalas et al., 2024)	Competition	×	✓	×	×	×
RLM25 (Poiroux et al., 2025)	Research projects	✓	×	×	✓	×
FormalMATH (Yu et al., 2025)	Mixed exercises	×	✓	×	×	×
FormalML (Yang et al., 2026)	Formal libraries	×	✓	×	×	✓
ProcessBench (Zheng et al., 2025)	Math solutions	×	×	✓	×	×
Hard2Verify (Pandit et al., 2026)	Competition math	×	×	✓	×	×
ArxivMathGradingBench (Barkallah et al., 2026)	Research papers	×	×	✓	✓	×
MLP-Bench (ours)	**Research papers, textbooks, graduate lectures**	✓	✓	✓	✓	✓

book exercises, or goals drawn from formal libraries (Zheng et al., 2022; Azerbayev et al., 2023; Yang et al., 2023; Tsoukalas et al., 2024). FormalML moves toward ML theory but evaluates subgoal completion inside an existing Lean skeleton (Yang et al., 2026). Table 1 shows that no prior benchmark jointly covers auto-formalization, theorem-proving, and proof-verification for research-level ML theory. This gap matters, because research-level ML theory requires models to recover assumptions left implicit in papers when formalizing claims, construct long, multi-step proofs from scratch under Lean feedback, and scrutinize the logical rigor of proof steps. We therefore ask: *are today's LLM agents ready to assist theoretical ML research end-to-end under formal verification?*

To answer this question, we introduce **MLP-Bench** (**M**achine Learning **L**ean **P**roof **Bench**mark), which evaluates LLM agents on the complementary capabilities of *forward construction* and *backward verification* for theoretical ML in Lean, spanning three tasks. The *forward* mode covers two tasks: auto-formalization and theorem-proving. In auto-formalization, a model is given an informal theorem and its surrounding context and must faithfully translate the theorem into a Lean statement; in theorem-proving, a model is given a human-verified Lean statement and required to construct a complete proof. The *backward* mode covers proof verification. Each instance presents 3–5 consecutive theorems from a paper or textbook, each with a multi-step proof, either fully correct or containing an error. A model must judge correctness and, if incorrect, localize the first erroneous step with Lean-checked evidence. Figure 1 illustrates all three tasks with concrete examples.

Evaluating models on MLP-Bench yields three key findings. First, auto-formalization remains a major bottleneck: even the strongest model, Claude-Opus-5, reaches only a **27.5%** BEq pass rate against gold Lean statements, with a major failure mode arising because ML theorems leave assumptions such as integrability or measurability implicit, and models struggle to recognize them and write them explicitly in Lean. Second, specialized Lean provers score **0.0%** on our theorem-proving task and remain markedly weaker on individual, decomposed proof steps, suggesting that pure-mathematics training data does not generalize to ML theory. Third, in *backward verification*, requiring a Lean-checked refutation to reject a step localizes errors far more reliably than rejecting on proof failure, suggesting that models are better at finding counterexamples than at proving steps.

Our contributions are summarized as follows:

- We propose a **forward–backward** framework for evaluating end-to-end LLM assistance in theoretical ML research under formal verification, and instantiate it as **MLP-Bench**, spanning auto-formalization, theorem-proving, and proof-verification.
- Our benchmark innovates by focusing on graduate- and research-level ML theory, whose theorems leave assumptions implicit and whose proofs are substantially longer and harder than those of the competition-level problems prior Lean benchmarks emphasize.
- We identify implicit-assumption recovery as a bottleneck in *forward construction* and show that complete theorem-proving benefits substantially from the agentic setting. For *backward verification*, requiring Lean-checked refutations yields more reliable error localization than flagging steps on proof failure.

More broadly, **MLP-Bench** offers a testbed for tracking whether agentic systems can move beyond empirical experimentation to construct and scrutinize theoretical claims rigorously. Its findings point toward directions for future work, from closing the assumption-recovery gap in auto-formalization to building practical verification tools on Lean-checked refutation.

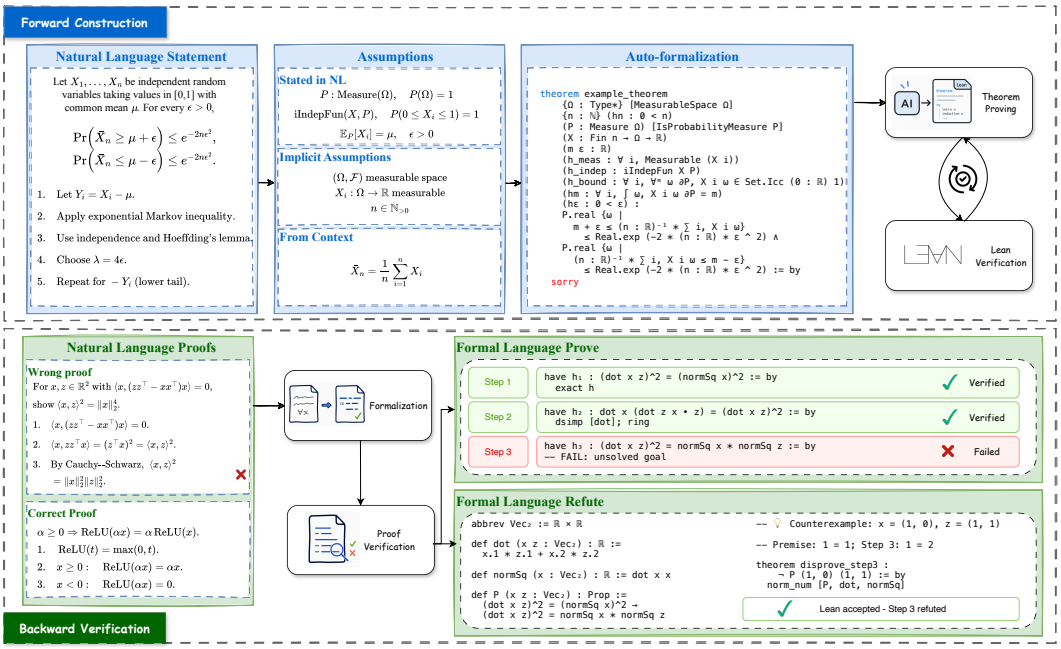


Figure 1: **MLP-Bench forward-backward framework:** *Forward construction* (top) covers auto-formalization and theorem-proving. *Backward verification* (bottom) covers proof verification, localizing the first erroneous step with Lean-checked evidence. Check Section 2 for further details.

2 MLP-Bench

To assess how well LLM agents can assist theoretical ML research, we introduce **MLP-Bench**, comprising 360 problems across 16 topics. It evaluates *forward construction* and *backward verification* in Lean through auto-formalization, theorem-proving, and proof-verification. In this section, we describe the design and construction of the three tasks in detail.

2.1 Benchmark Construction

2.1.1 Data Collection & Processing

For *forward construction*, we draw graduate-level tasks from graduate instructional materials and research-level tasks from *Journal of Machine Learning Research* (JMLR) papers published between 2002 and 2024. For proof verification, each problem consists of three to five ordered theorems with their NL proofs. Fully correct groups are taken from textbooks, graduate lecture notes, and JMLR papers, while real flawed cases are recovered from ICLR submissions and public peer reviews from 2016 through 2026 (OpenReview, 2026). In addition to these real flawed cases, we construct synthetic cases by introducing a localized error into a verified proof, such as weakening an assumption or reversing an inequality; to ensure diverse error types, LLMs generate a large pool of candidate errors under rubrics, from which the authors select representative ones. The distribution over real and synthetic flawed cases is shown in Appendix A.2. The source collections are listed in Appendix A.1. We include source materials only when their licenses permit open release of the derived benchmark data or their authors have granted permission for such release. We convert each source PDF to Markdown using MinerU (Wang et al., 2024) and then use GPT-5.5 to extract necessary context.

2.1.2 Formalization & Validation

For *forward-construction* tasks, the authors first outline each formalization, and an LLM agent fills in the Lean statement. The agent strictly follows this outline, writing out in Lean the implicit assumptions and notation expansions the authors identified. Candidates undergo manual semantic faithfulness checks and Lean compilation, with failures revised by the authors using compiler and proof-state feedback. Proofs are then constructed through a human-in-the-loop process that follows

the original NL proof, with reviewers continually checking faithfulness and correcting statement issues until complete Lean-checked proofs are obtained. All 120 Lean statements have gold proofs aligned with their original NL proofs. For theorem-proving instances, we remove the target Lean proof while retaining the NL statement as context.

For proof-verification, error locations are recorded when synthetic errors are introduced or when cases are curated from flaws identified in peer reviews. Human reviewers verify that, within each flawed group of theorem–proof pairs, the labeled step is the first *substantive error*, with all preceding steps valid. A substantive error is a false claim or an invalid proof step under the stated assumptions, even if the theorem's conclusion remains true. Minor issues commonly tolerated in ML theory (e.g., trivially valid boundary cases or loose but valid bounds) do not count; see Appendix D.1 for detailed criteria. For groups labeled as correct, reviewers check every theorem and proof to confirm that no substantive error is present. Reviewer qualifications are given in Appendix A.3.

2.1.3 Benchmark Statistics

MLP-Bench contains 120 theorem instances for *forward construction*, each yielding one auto-formalization problem and one theorem-proving problem, for a total of 240 problems. Each theorem instance includes the original NL theorem, context, and proof, along with a verified Lean statement and its gold proof. 120 *backward-verification* problems each present three to five ordered theorems with their NL proofs, with 60 problems containing a substantive error, while the remaining 60 are fully correct. Together, these form 360 problems over 240 instances (120 theorems and 120 verification groups) across 16 topics; Figure 2 shows their topic distribution.

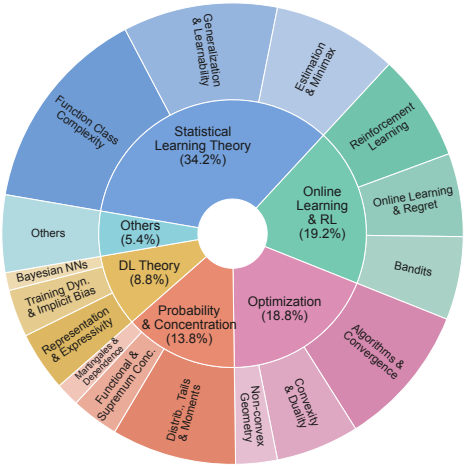


Figure 2: Topic distribution of MLP-Bench.

2.2 Task Description

In this section, we describe the *forward-construction* and *backward-verification* tasks. Case studies and prompts are provided in Appendices E and F, respectively.

2.2.1 Forward Construction

Auto-formalization. Given a Lean 4 file with one target theorem's statement blanked out, together with the NL source theorem and its context, the model must restore the text between the theorem's name and `:=` while preserving the intended mathematical meaning. The proof is fixed to `sorry`, isolating statement formalization from proving. We evaluate the generated statement against its gold Lean statement with BEq (Liu et al., 2025b), a faithfulness metric that deems two statements equivalent when each can be rewritten into a form definitionally equal to the other. For higher recall, we let two strong agents derive each statement from the other, accepting equivalence only when Lean verifies both directions and the derivations pass the gates of Appendix B.3. We also report a *validity rate*: the proportion of generated statements that compile and that neither agent refutes. BEq inherits the high-precision design of the original metric, so certified pairs can be trusted as faithful even if some true equivalences go uncertified. Validity instead gives a loose upper bound with high recall, since every faithful statement compiles and cannot be refuted.

Theorem-proving. Given a Lean 4 file with a human-verified target statement and its proof replaced by `sorry`, the model must supply a complete Lean proof without changing the statement. Unlike auto-formalization, this task provides the formal claim and evaluates only the ability to prove it. We report the proportion of problems solved, counting a proof as successful only when Lean accepts the completed proof without `sorry` or newly introduced axioms.

2.2.2 Backward Verification

Proof-verification. Each problem presents 3–5 ordered theorems with NL proofs pre-segmented into steps. The model must locate the first substantive error, if any. We compare two formal language

(FL) decision rules, both requiring the model to formalize every step in Lean itself. This mirrors real-world verification, where reviewers receive only NL proofs and no Lean formalization is available. Inspired by Safe (Liu et al., 2025a), *FL Prove* flags the first step the model fails to prove, treating proof failure as a rejection signal. *FL Refute* instead flags a step only when Lean accepts the model's refutation of the step's formalization. The prediction is the earliest flagged step, or no error if none is flagged. FL Prove thus reports no error only when every step is proved, whereas FL Refute reports no error as long as every step's formalization compiles and no refutation is provided, even if proofs are left as `sorry`. A prediction is correct only if it reports no error for a correct group or exactly matches the first annotated error in a flawed one. We report precision, recall, accuracy, and F1 with flawed theorem groups as the positive class. A mislocated error counts as both a false positive and a false negative. Non-compiling or malformed outputs are scored as incorrect.

3 Experiments

3.1 Setup

We evaluate 11 open- and closed-source models under the task definitions and metrics in Section 2, with results in Tables 2, 3, and 4. Apart from general-purpose models, we include DeepSeek-Prover-V2-7B (Ren et al., 2025), Goedel-Prover-V2-8B (Lin et al., 2026), and Kimina-Prover-Distill-8B (Wang et al., 2025; Numina & Kimi Team, 2025) as specialized Lean prover baselines. For auto-formalization, GPT-5.6-Sol and Claude-Opus-5 serve as the two evaluator agents that search for Lean equivalence proofs and refutations, each run once per output, so no model is judged solely by itself. Since Lean checks every proof, evaluator bias cannot inflate scores (Appendix B.2). BEq accepts equivalence verified by either evaluator, while validity requires compilation and no refutation from either. Three gates reject proofs that do not genuinely derive one statement from the other (Appendix B.3). For theorem-proving, we compare Chain-of-Thought (CoT) (Wei et al., 2022) with autonomous agents on 60 graduate and 60 research-level problems. For proof-verification, we compare CoT and autonomous agents under both *FL Prove* and *FL Refute* settings. For agent evaluations, we use Codex as the default harness and Claude Code for Claude models, leveraging their native integration. DeepSeek-Prover lacks iterative refinement, so its agent results are marked "–"; for Goedel-Prover and Kimina-Prover, we use their official error-fixing workflows (Appendix B).

3.2 Main Results

Tables 2 and 3 report auto-formalization and theorem-proving results for *forward construction*, while Table 4 reports proof-verification results for *backward verification*.

Table 2: **Auto-formalization** results on 120 theorems, reporting bidirectional equivalence (BEq) and validity (Valid) rates (%) under CoT and Agent settings.

Model	BEq ↑		Valid ↑	
	CoT	Agent	CoT	Agent
Closed-source Models				
GPT-5.6-Luna	13.3	17.5	41.7	55.8
GPT-5.6-Terra	13.3	18.3	40.0	55.0
GPT-5.6-Sol	<u>20.8</u>	<u>25.0</u>	49.2	<u>66.7</u>
Claude-Sonnet-5	15.0	17.5	<u>50.8</u>	64.2
Claude-Opus-5	**23.3**	**27.5**	**74.2**	**78.3**
Open-source Models				
DeepSeek-V4 Flash	10.0	15.0	38.3	45.8
Gemma-4 26B-A4B	0.0	0.0	17.5	20.0
Qwen3.5-9B	0.0	0.0	15.0	16.7

Table 3: **Theorem-proving** results on 60 graduate and 60 research-level Lean statements, reporting Lean-verified proof success rates (%) under CoT and Agent settings.

Model	Graduate		Research	
	CoT	Agent	CoT	Agent
Closed-source Models				
GPT-5.6-Luna	0.0	3.3	0.0	10.0
GPT-5.6-Terra	0.0	5.0	0.0	10.0
GPT-5.6-Sol	**1.7**	<u>48.3</u>	0.0	**41.7**
Claude-Sonnet-5	0.0	46.7	0.0	28.3
Claude-Opus-5	0.0	**68.3**	0.0	<u>40.0</u>
Open-source Models				
Kimina-Prover-Distill-8B	0.0	0.0	0.0	0.0
DeepSeek-Prover-V2-7B	0.0	-	0.0	-
Goedel-Prover-V2-8B	0.0	0.0	0.0	0.0

3.2.1 Forward Construction

Auto-formalization. Faithfully translating natural-language theorems into Lean remains challeng-

Table 4: **Proof-verification** results on 120 MLP-Bench problems, comparing FL Prove and FL Refute under CoT and Agent settings using precision, recall, accuracy, and F1 (%); parentheses report Agent F1 gains in percentage points from FL Prove to FL Refute.

Model	Method	Precision ↑		Recall ↑		Accuracy ↑		F1 ↑	
		CoT	Agent	CoT	Agent	CoT	Agent	CoT	Agent
Closed-source Models									
GPT-5.6-Luna	FL Prove	1.7	2.5	3.3	5.0	3.3	2.5	2.3	3.4
	FL Refute	9.1	15.3	11.7	21.7	13.3	19.2	10.2	17.9 (+14.5)
GPT-5.6-Terra	FL Prove	1.7	3.3	3.3	6.7	1.7	3.3	2.2	4.4
	FL Refute	21.2	27.0	30.0	33.3	23.3	32.5	24.8	29.9 (+25.5)
GPT-5.6-Sol	FL Prove	1.8	8.8	3.3	16.7	1.7	10.8	2.3	11.5
	FL Refute	19.1	30.9	30.0	50.0	20.8	39.2	23.4	38.2 (+26.7)
Claude-Sonnet-5	FL Prove	2.6	12.8	5.0	23.3	3.3	19.2	3.4	16.6
	FL Refute	26.0	46.0	21.7	38.3	34.2	55.0	23.6	41.8 (+25.2)
Claude-Opus-5	FL Prove	8.0	20.4	11.7	31.7	21.7	35.8	9.5	24.8
	FL Refute	**59.5**	**61.7**	**36.7**	**61.7**	**62.5**	**67.5**	**45.4**	**61.7** (+36.9)
Open-source Models									
DeepSeek-V4 Flash	FL Prove	1.0	1.7	1.7	3.3	0.8	1.7	1.3	2.3
	FL Refute	0.0	8.3	0.0	5.0	18.3	25.0	0.0	6.2 (+3.9)
Gemma-4 26B-A4B	FL Prove	1.0	0.0	1.7	0.0	0.8	0.0	1.3	0.0
	FL Refute	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0 (+0.0)
Qwen3.5-9B	FL Prove	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	FL Refute	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0 (+0.0)

ing. Claude-Opus-5 achieves the highest agentic BEq score, but only **27.5%** of its statements are verified as equivalent to the gold. GPT-5.6-Sol, another top-tier model, reaches 66.7% validity, so **33.3%** of its statements fail to compile or are refuted. Thus, leading models struggle to produce faithful and valid statements. Our agentic BEq also certifies far more true equivalences than the symbolic BEq^+ metric (Appendix B.2). Section 4.1.1 traces most failures to omitted assumptions.

Theorem-proving. Successful proof construction largely depends on agentic iteration and repair. CoT achieves at most **1.7%** on graduate problems and **0.0%** on research problems, whereas agents reach 68.3% and 41.7%, respectively. Yet all three open-source specialized provers score **0.0%** on both splits. Appendix C.1.4 shows that this remains the same even when sampling is scaled up to pass@32. This shows how poorly current specialized provers transfer to ML theory. Section 4.1.2 examines specialized provers' limitations and the challenges of constructing long ML proofs.

3.2.2 Backward Verification

Proof-verification. FL Refute consistently outperforms FL Prove across all five closed-source models. In the Agent setting, switching from FL Prove to FL Refute raises GPT-5.6-Luna's F1 from 3.4% to 17.9% and Claude-Opus-5's from 24.8% to 61.7%. The advantage therefore holds across models with different sizes. This suggests that a Lean-checked refutation is a more reliable error signal than a failed proof attempt. Section 4.2.1 examines this advantage through error-localization behavior and models' relative strengths in refuting versus proving claims. Appendix D.2 analyzes formalization faithfulness, and Appendix D.3 compares FL Refute with an agentic NL baseline, where Lean refutation improves over NL for Claude models but not GPT models.

4 Experiment Analysis

4.1 Forward Construction

4.1.1 Auto-formalization

We identify the omission of implicit assumptions as a central failure mode of current models in auto-formalization. Recovering basic prerequisites such as measurability and integrability is foun-

dational, because correctly translating the rest of a theorem cannot compensate for missing hypotheses needed for its validity. We systematically examine how often models omit assumptions, how retaining assumptions relates to semantic equivalence, and which types they omit.

Assumption omission is a major problem. Table 5 shows that, even among Lean statements that compile, **65.0–83.8%** are flagged as omitting at least one assumption in the gold statement.

To identify omissions, we ask GPT-6-Astra and Claude-Opus-5 to independently list the gold assumptions missing from a compiled prediction, treating assumptions restated in different notation as present. An assumption counts as omitted when both judges flag it and as present when neither does, and the authors adjudicate the cases where the two judges disagree.

Table 5: Assumption omission rates and BEq with no flagged omissions (%). Parentheses show gains over overall Agent BEq in percentage points.

Model	Omissions ≥ 1 (%)	BEq (%) No omissions
GPT-5.6-Luna	79.1	47.8 (+30.3)
GPT-5.6-Terra	79.5	43.5 (+25.2)
GPT-5.6-Sol	73.4	48.3 (+23.3)
Claude-Sonnet-5	73.3	41.9 (+24.4)
Claude-Opus-5	65.0	48.8 (+21.3)
DeepSeek-V4 Flash	83.8	50.0 (+35.0)
Gemma-4 26B-A4B	75.7	0.0 (+0.0)
Qwen3.5-9B	81.2	0.0 (+0.0)

Complete assumptions are associated with higher BEq. Table 5 reports BEq on predictions with no flagged omissions. For the five closed-source models, BEq on this subset ranges from **41.9%** to **48.8%**, exceeding their overall Agent BEq scores by **21.3** to **30.3** percentage points, which suggests that complete assumptions are associated with more faithful formalization. However, BEq stays no more than 50% even in this subset, and both Gemma and Qwen remain at zero. Recovering the necessary assumptions is thus necessary but not sufficient, as models must also preserve the remaining mathematical content of the statement.

Omissions span foundational conditions in ML theory. Figure 3 breaks down omitted assumptions by family for Claude-Opus-5 and GPT-5.6-Sol. Measurability and integrability account for substantial shares in both models, alongside conditions on positivity, range, non-emptiness, probability, and regularity. The large "Other" category indicates that omissions also extend beyond these named families. The problem is therefore not confined to one type of technical condition. Rather, models broadly struggle to make explicit the prerequisites that support statements in ML theory.

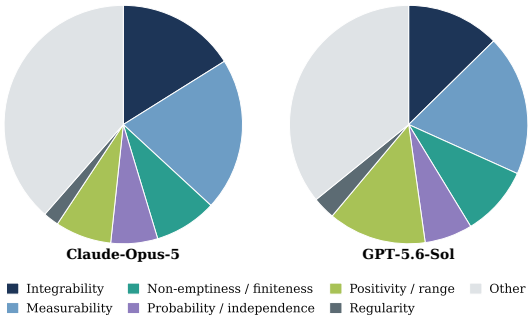


Figure 3: Distribution of omitted assumptions by family for two leading models.

Together, these analyses show that (1) models frequently omit basic assumptions across many families of conditions, and (2) complete assumptions are associated with higher BEq but do not guarantee faithful formalization.

4.1.2 THEOREM-PROVING

In this subsection, we investigate why specialized theorem provers struggle with ML theory problems. We first examine their ability to prove single steps, and then how proof length compounds the difficulty of constructing graduate- and research-level ML proofs.

Single-step proving. We extract 150 non-trivial leaf subgoals from gold proofs, excluding those closed by a single tactic, each with its local context. Models attempt each subgoal once, without tools, compiler feedback, or retries. Success requires compilation without `sorryAx`.

Table 6 shows that specialized provers solve only **12.7–18.7%** of subgoals, compared with 52.0–84.7% for general-purpose models. Despite their strong performance on competition-level mathematics, which we confirm on miniF2F (Appendix C.1.5), specialized provers fail on most individual ML theory subgoals even when given the local context. This suggests limited generalization to ML theory, where probability, measure theory, and integration play a central role. Specialized provers also perform poorly at ML proof subgoal decomposition (Appendix C.1.3).

Long proofs underscore the need for iterative refinement. Claude-Opus-5 performs best on single-step proving, achieving a success rate of **84.7%**. However, its CoT success rate on the whole-theorem proving task is **0.0%**. This contrast reflects the challenge of constructing complete proofs for our graduate- and research-level problems, whose gold proofs average **107** subgoals each. Solving these problems requires correctly solving every subgoal, rather than isolated ones. As errors compound over longer proofs, whole-theorem success rates may decline **exponentially** with proof length. Finding a correct decomposition adds further difficulty, helping explain Claude-Opus-5's **0.0%** whole-theorem CoT success despite strong single-step performance.

Table 6: Solve rates on 150 subgoals.

Model	Solve rate (%)
GPT-5.6-Luna	52.0
GPT-5.6-Terra	60.7
GPT-5.6-Sol	70.0
Claude-Sonnet-5	64.0
Claude-Opus-5	**84.7**
Kimina-Prover-Distill-8B	12.7
DeepSeek-Prover-V2-7B	18.7
Goedel-Prover-V2-8B	15.3

Together with the gains from agentic proving, this gap underscores the practical need for iterative refinement on graduate- and research-level problems. Appendix C.1.1 examines how models reduce Lean compilation errors over successive revisions.

Together, these analyses highlight two findings: (1) specialized provers fail to generalize effectively to graduate- and research-level ML theory, and (2) iterative refinement is essential for tackling graduate- and research-level problems in our benchmark.

4.2 Backward Verification

4.2.1 Proof-verification

We analyze proof-verification results, focusing on error-localization behavior and on how models' ability to prove steps compares with their ability to refute them.

FL Prove introduces an early-localization bias. Figure 4 compares FL Prove and FL Refute in the Agent setting, averaged across five models. Localization offsets measure the difference between the reported and real error steps, with negative values indicating earlier reports. For better visualization, bar heights use a square-root scale. FL Prove predominantly reports steps before the first actual error. This reflects a limitation of using proof failure as a rejection signal, as a valid step may remain unproved because of limited proving capability. Since the agent reports the earliest flagged step, such failures can preempt the actual error and bias localization toward earlier steps. FL Refute instead requires a Lean-checked refutation, shifting reports toward the gold step and bringing the mean offset closer to zero. Thus, requiring evidence of falsehood reduces the premature rejection induced by flagging unproved steps, although both rules still make localization mistakes.

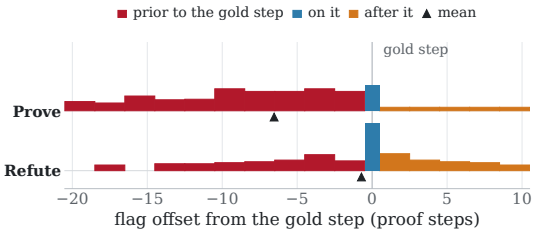


Figure 4: Error-localization offsets on flagged flawed proofs in the Agent setting, averaged across five models.

Refuting is more tractable than proving. To examine counterexample construction separately from error localization, we perturb each of the 120 benchmark Lean statements by modifying hypotheses, conclusions, constants, quantifiers, or inequalities. Models must prove the negated target without being told what changed. Table 7 shows that refutation success exceeds agent proving success in Table 3 by **45.8** to **67.5** percentage points. Proving requires a coherent sequence of valid deductions covering all cases allowed by the hypotheses, whereas refuting a universal claim requires only one valid counterexample. Given a false formal statement, refutation is thus the more tractable task for current models, which helps explain why FL Refute outperforms FL Prove.

Table 7: Proving vs. refutation success (%); parentheses show gains in percentage points.

Model	Prove	Refute
Claude-Opus-5	54.2	**100.0** (+45.8)
GPT-5.6-Sol	45.0	96.7 (+51.7)
Claude-Sonnet-5	37.5	90.8 (+53.3)
GPT-5.6-Terra	7.5	75.0 (+67.5)
GPT-5.6-Luna	6.7	64.2 (+57.5)

LLMs have limited understanding of ML theory. Our analysis of counterexample construction reveals a recurring cycle: models *localize a fault*, *try special values*, and iteratively refine their counterexamples using *Lean feedback*. Models often rely on repeated Lean-guided revisions rather than immediately identifying decisive counterexamples; Appendix C.2.1 provides the detailed analysis and accuracy curves by compilation round.

These results reveal two findings. First, FL Refute outperforms FL Prove in proof verification, largely benefiting from models' stronger performance on refutation than on proving. Second, despite strong refutation performance, models still exhibit limited ML-theoretic understanding, often finding counterexamples through trial and error rather than directly identifying the mathematical reason a claim fails.

5 Related Work

5.1 Auto-Formalization

Auto-formalization translates an informal mathematical statement and its context into a semantically faithful formal statement. Early work translated competition problems into formal specifications, followed by ProofNet's undergraduate benchmark (Wu et al., 2022; Azerbayev et al., 2023). Lean Workbook and Herald build large-scale informal–formal parallel data (Ying et al., 2024; Gao et al., 2025). More recently, BEq and BEq+ evaluate semantic equivalence, with the latter also introducing ProofNetVerif, ProofNet#, and the research-level RLM25 benchmark (Liu et al., 2025b; Poiroux et al., 2025). To our knowledge, no existing benchmark targets ML theory, where faithful formalization requires recovering assumptions left implicit in the source context.

5.2 Theorem Proving

Theorem proving takes a formal statement as input and must construct a complete proof accepted by the Lean kernel. Beginning with miniF2F, benchmarks expanded through FIMO, ProofNet, and LeanDojo (Zheng et al., 2022; Liu et al., 2023; Azerbayev et al., 2023; Yang et al., 2023). PutnamBench raised difficulty and FormalMATH broadened coverage (Tsoukalas et al., 2024; Yu et al., 2025). FormalML enters ML theory but evaluates short subgoals from existing formal libraries (Yang et al., 2026). FATE pushes toward research-level abstract algebra, where specialized provers likewise collapse, but stays within pure mathematics (Jiang et al., 2026). To our knowledge, no existing benchmark requires complete, multi-step proofs for research-level ML theorems.

5.3 Proof Verification

Proof verification determines whether a candidate argument is correct and, when it is not, identifies the earliest invalid step. Outcome verifiers (Cobbe et al., 2021) were followed by step-level process supervision (Lightman et al., 2024), then by ProcessBench and Hard2Verify for first-error localization (Zheng et al., 2025; Pandit et al., 2026). Recent work evaluates complete proofs, fine-grained grading, and pseudo-formal verification (Dekoninck et al., 2026; Mahdavi et al., 2026; Ma et al., 2026; Barkallah et al., 2026). These methods focus on general mathematics and rely mainly on learned or pseudo-formal judgments. To our knowledge, no existing work grounds the verification of research-level ML proofs in Lean-checked formalizations.

6 Conclusion

We introduced **MLP-Bench**, a benchmark for evaluating LLM agents on *forward construction* and *backward verification* in theoretical ML through auto-formalization, theorem-proving, and proof-verification in Lean. Our results reveal an asymmetry in which current agents struggle to recover implicit assumptions and construct complete proofs, yet readily refute false Lean statements, which makes refutation a more reliable basis for localizing errors. This suggests that their current promise lies more in assisting critical scrutiny than in producing rigorous theoretical results. Closing this gap requires advances in both faithful formalization and research-level proving. MLP-Bench provides a testbed for tracking these advances toward agents that can reliably assist both the construction and verification of ML theory.

AI USE STATEMENT

Data and benchmark construction. For the *forward construction* portion of our benchmark, we used both Codex with GPT-5.5 and Claude Code with Claude-Opus-4.6 to draft the initial Lean code in a human-in-the-loop process: the authors provided the outline of each formalization, and the tools filled in the Lean statements and performed a preliminary check for omissions. The authors then manually checked the drafts for faithfulness to the source material, revised them where necessary, and proved all statements in Lean, again in a human-in-the-loop process with the same tools. For *backward verification*, to ensure diverse error types, we used GPT-5.5 and Claude-Opus-4.6 to generate a large pool of candidate erroneous proofs, with each modification confined to error rubrics specified by the authors, from which the authors manually selected representative errors and checked each selected problem. For all problems, Codex with GPT-5.5 extracted relevant context from the source material. We deliberately extracted additional context to reduce omissions, then manually removed irrelevant information. The mathematical content of every problem, from the formalization outlines to the error specifications, therefore originates from the authors, and humans make every final inclusion decision. The LLMs served only to implement this content, which limits any advantage to evaluated models from the same families.

Experiments and implementation. All experimental design and analysis were performed by the authors. The authors provided the overall code structure, and Claude Code with Opus-series models assisted with implementation. All code was manually reviewed and tested. Claude Code with Opus-series models also assisted in generating experimental analysis figures, which the authors checked. In the assumption-omission analysis (Section 4.1.1), GPT-6-Astra and Claude-Opus-5 serve as LLM judges that flag assumptions of the gold statement missing from a predicted statement, and the authors adjudicate their disagreements. These judgments are used only for this diagnostic analysis, not for any headline metric, which is decided by the Lean kernel. For agentic BEq, GPT-5.6-Sol and Claude-Opus-5 also serve as evaluators, but each output is checked by at least one external evaluator, and since Lean decides every proof, scores are not subject to model bias (Appendix B.2).

Writing and presentation. We used GPT-6-Astra and Claude-Fable-5.1 for language polishing, condensing overly long passages, and assistance with typesetting.

ETHICS STATEMENT

We use source materials only under licenses that permit academic use or with permission from the copyright holders, with details of the source collections provided in Appendix A.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our work. Section 2 describes the construction and task definitions of MLP-Bench, while Section 3 details the experimental setup and results; additional analyses and case studies are provided in Appendices C and E. We will open-source the code and publicly release the MLP-Bench benchmark after the double-blind review period ends.

REFERENCES

Anthropic. Formalizing Fermat's Last Theorem, 2026. URL https://www.anthropic.com/news/formalizing-fermats-last-theorem. Accessed: 2026-09-26.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023. URL https://arxiv.org/abs/2302.12433.

Slim Barkallah, Luke Bailey, Kaiyue Wen, Mohammed Abouzaid, and Tengyu Ma. Pseudo-formalization for automatic proof verification. *arXiv preprint arXiv:2605.20531*, 2026. URL https://arxiv.org/abs/2605.20531.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28*, 2021. URL https://link.springer.com/chapter/10.1007/978-3-030-79876-5_37.

Jasper Dekoninck, Ivo Petrov, Kristian Minchev, Mislav Balunovic, Martin Vechev, Miroslav Marinov, Maria Drencheva, Lyuba Konova, Milen Shumanov, Kaloyan Tsvetkov, Nikolay Drenchev, Lazar Todorov, Kalina Nikolova, Nikolay Georgiev, Vanesa Kalinkova, and Margulan Ismoldayev. The Open Proof Corpus: A large-scale study of LLM-generated mathematical proofs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.21621.

John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10(99):2899–2934, 2009. URL https://www.jmlr.org/papers/v10/duchi09a.html.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated Lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.10878.

Moritz Hardt and Max Simchowitz. Course notes for EE227C (Spring 2018): Convex optimization and approximation. University of California, Berkeley course notes, 2020. URL https://ee227c.github.io/notes/ee227c-notes.pdf.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2310.01798.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b. URL https://arxiv.org/abs/2310.03302.

Jiedong Jiang, Wanyi He, Yuefeng Wang, Guoxiong Gao, Yongle Hu, Jingting Wang, Nailin Guan, Peihao Wu, Chunbo Dai, Liang Xiao, and Bin Dong. FATE: A formal benchmark series for frontier algebra of multiple difficulty levels. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2511.02872.

Journal of Machine Learning Research. JMLR Papers. Journal archive, 2026. URL https://jmlr.org/papers/. Accessed: 2026-09-26.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2206.14858.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.03613.

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, Ming Zhang, and Qun Liu. FIMO: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*, 2023. URL https://arxiv.org/abs/2309.04295.

Chengwu Liu, Ye Yuan, Yichun Yin, Yan Xu, Xin Xu, Zaoyu Chen, Yasheng Wang, Lifeng Shang, Qun Liu, and Ming Zhang. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025a. URL https://arxiv.org/abs/2506.04592.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d630537fc4402cfa3ebbc7450a0cac91-Abstract-Conference.html.

Yaowenqi Liu, Bingxu Meng, Rui Pan, Yuxing Liu, Jerry Huang, Jiaxuan You, and Tong Zhang. GUIDE: Towards scalable advising for research ideas. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2507.08870.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. URL https://arxiv.org/abs/2408.06292.

Tengyu Ma. STATS214 / CS229M: Machine learning theory. Stanford University course page, 2021. URL https://web.stanford.edu/class/stats214/.

Wenjie Ma, Andrei Cojocaru, Neel Kolhe, Haihan Zhang, Vincent Zhuang, Matei Zaharia, and Sewon Min. Reliable fine-grained evaluation of natural language math proofs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/764d4fc73ab69289b1e10f15089274f3-Abstract-Conference.html.

Sadegh Mahdavi, Branislav Kisacanin, Shubham Toshniwal, Wei Du, Ivan Moshkov, George Armstrong, Renjie Liao, Christos Thrampoulidis, and Igor Gitman. Scaling generative verifiers for natural language mathematical proof verification and selection. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2511.13027.

Numina & Kimi Team. Kimina-Prover: Applying test-time RL search on large formal reasoning models. Hugging Face blog, 2025. URL https://huggingface.co/blog/AI-MO/kimina-prover.

OpenAI. On the Navier–Stokes Millennium Prize Problem, 2026. URL https://openai.com/index/navier-stokes-solution/. Accessed: 2026-09-26.

OpenReview. ICLR. OpenReview venue page, 2026. URL https://openreview.net/group?id=ICLR.cc. Accessed: 2026-09-26.

Dmitry Panchenko. MIT 18.465: Topics in Statistics: Statistical Learning Theory. MIT OpenCourseWare, 2007. URL https://ocw.mit.edu/courses/18-465-topics-in-statistics-statistical-learning-theory-spring-2007/. Spring 2007 course page.

Shrey Pandit, Austin Xu, Xuan-Phi Nguyen, Yifei Ming, Caiming Xiong, and Shafiq Joty. Hard2Verify: A step-level verification benchmark for open-ended frontier math. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2510.13744.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://arxiv.org/abs/2406.07222.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025. URL https://arxiv.org/abs/2504.21801.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://arxiv.org/abs/2501.04227.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2020. URL https://arxiv.org/abs/1910.09336.

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. PutnamBench: Evaluating neural theorem-provers on the Putnam mathematical competition. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. URL https://arxiv.org/abs/2407.11214.

Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, Bo Zhang, Liqun Wei, Zhihao Sui, Wei Li, Botian Shi, Yu Qiao, Dahua Lin, and Conghui He. MinerU: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*, 2024. URL https://arxiv.org/abs/2409.18839.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-Prover Preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025. URL https://arxiv.org/abs/2504.11354.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.12615.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2023. URL https://arxiv.org/abs/2306.15626.

Long Yang, Gang Zheng, Haotian Zhang, Yu Zhang, Qian Zheng, Jun Wen, and Gang Pan. Policy optimization with stochastic mirror descent. *arXiv preprint arXiv:1906.10462v2*, 2019. URL https://arxiv.org/abs/1906.10462v2. Version 2, 27 September 2019.

Xiao-Wen Yang, Zihao Zhang, Jianuo Cao, Zhi Zhou, Zenan Li, Lan-Zhe Guo, Yuan Yao, Taolue Chen, Yu-Feng Li, and Xiaoxing Ma. FormalML: A benchmark for evaluating formal subgoal completion in machine learning theory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.02335.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean Workbook: A large-scale Lean problem set formalized from natural language math problems. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/bf236666a2cc5f3ae05d2e08485efc4c-Abstract-Datasets_and_Benchmarks_Track.html.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. FormalMATH: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025. URL https://arxiv.org/abs/2505.02735.

Tong Zhang. *Mathematical Analysis of Machine Learning Algorithms*. Cambridge University Press, 2023. doi: 10.1017/9781009093057. URL https://www.tongzhang-ml.org/lt-book/lt-book.pdf.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2412.06559.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *The Tenth International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2109.00110.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2023. URL https://arxiv.org/abs/2306.05685.

Minjun Zhu, Yixuan Weng, Linyi Yang, and Yue Zhang. DeepReview: Improving LLM-based paper review with human-like deep thinking process. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2503.08569.

APPENDIX

A Benchmark Details **16**
A.1 Source Collections and Data Contamination 16
A.2 Proof-Verification Distribution 16
A.3 Annotator and Reviewer Qualifications 16

B Experiment Details **17**
B.1 Models, Harnesses, and Budgets 17
B.2 Auto-formalization Evaluator Design 17
B.3 Gates against Spurious Equivalence 18

C Additional Forward-Construction Analysis **18**
C.1 Theorem-Proving 18
C.2 Proof-Verification 20

D Additional Backward-Verification Analysis **21**
D.1 Error Criteria 21
D.2 Formalization Analysis 21
D.3 Agentic NL versus FL Refute 22

E Case Studies **23**
E.1 Graduate-Level Forward Construction 23
E.2 Research-Level Forward Construction 24
E.3 Agentic Proof Verification 26

F Prompts **27**
F.1 Auto-Formalization 27
F.2 Theorem-Proving 28
F.3 Proof-Verification 28

A Benchmark Details

A.1 Source Collections and Data Contamination

All source materials are used either under licenses that permit academic use or with permission from the copyright holders. To the best of our knowledge, no theorem in our source collections has a publicly available Lean formalization or machine-checked proof.

For Proof-verification, prior exposure to NL content on OpenReview cannot be ruled out, but three properties limit its effect. The sources contain no Lean code, so nothing the FL rules require can be recalled. Public reviews point to a flaw by page, equation, or line number rather than restating the proof, and our problems carry none of these references, so a remembered comment does not map to a step index. Finally, the model never renders the verdict itself, and under FL Refute a flag still requires a counterexample that Lean accepts.

Table 8: Source collections used to construct MLP-Bench.

Level	Source	Material
Graduate	MIT 18.465, *Statistical Learning Theory* (Panchenko, 2007)	Course notes
Graduate	UC Berkeley EE227C, *Convex Optimization and Approximation* (Hardt & Simchowitz, 2020)	Course notes
Graduate	Stanford CS229M/Stats 214, *Machine Learning Theory* (Ma, 2021)	Course notes
Graduate	*Mathematical Analysis of Machine Learning Algorithms* (Zhang, 2023)	Textbook
Research	*Journal of Machine Learning Research* archive (Journal of Machine Learning Research, 2026)	Papers, 2002–2024
Research	ICLR OpenReview (OpenReview, 2026)	Submissions and public reviews, 2016–2026

A.2 Proof-Verification Distribution

The 120 proof-verification problems are balanced between incorrect and correct proofs. Incorrect proofs comprise 21 synthetically generated cases and 39 cases drawn from ICLR submissions and reviews, while correct proofs comprise 34 cases from textbooks and graduate lecture notes and 26 cases from JMLR papers (Figure 5).

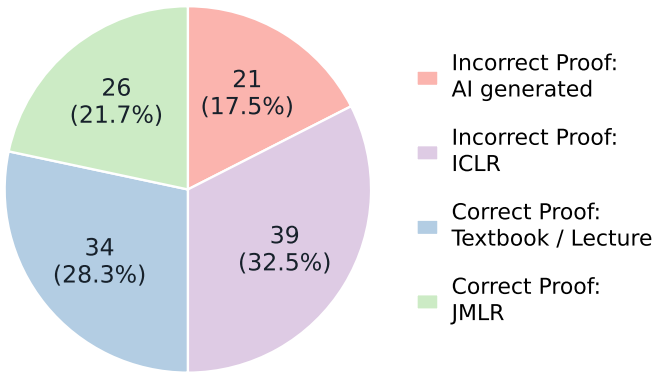


Figure 5: **Distribution of proof-verification problems.** Counts and percentages are shown by proof correctness and source category.

A.3 Annotator and Reviewer Qualifications

The benchmark was constructed and reviewed by two PhD students and one master's student in computer science. All have published at top-tier ML conferences, and all are authors of this paper.

B Experiment Details

B.1 Models, Harnesses, and Budgets

Evaluation snapshot. Our main experiment was conducted during August 13–19, 2026. We report this evaluation window because model updates may affect performance over time; results obtained at other dates may therefore differ. We run each configuration once, since agents in our predominantly agentic setting receive continual feedback from the Lean kernel, which keeps outcomes stable across runs.

Model access and hardware. We accessed `deepseek-v4-flash` through the OpenRouter API and ran experiments with the other open-source models on two NVIDIA A6000 GPUs. For OpenAI models, we used Codex CLI (version `0.145.0`) with a Max subscription; for Claude models, we used Claude Code (version `2.1.241`) with a Max subscription. All closed-source models were run at medium reasoning effort in both CoT and agentic modes.

Lean environment. All experiments used Lean `4.27.0-rc1` and Mathlib `v4.27.0-rc1`.

Agentic time budgets and termination. In agentic mode, we set per-instance time limits of 1,200 seconds for auto-formalization, 2,400 seconds for theorem proving, and 3,600 seconds for proof verification. No closed-source model run reached its time limit; all terminated autonomously. Some open-source model runs reached the time limit after repeatedly looping without substantive progress. We interpret this behavior as a limitation of the models' ability to make progress rather than evidence of an insufficient time budget. Based on these observed trajectories, we do not expect terminating such stalled runs to materially affect the reported results.

CoT output budgets and scoring. In CoT mode, we impose neither a wall-clock timeout nor an additional output-token cap. Each single-turn response is instead bounded by the model's native maximum output allowance, which cannot be exceeded and includes thinking tokens. For GPT-5.6-series models and Claude-Opus-5, this maximum is 128,000 tokens. Responses that exhaust the model's native limit without producing a valid final output are treated as invalid and scored as incorrect.

Agent affordances. All agent evaluations run inside an offline sandbox containing only the task's Lean project. The agent may read and write files within it, search the local Mathlib source, and invoke `lake` to compile the project and read the resulting Lean messages, which is its only channel for feedback on correctness. All other affordances are disabled, including web search, package installation, and access to files outside the project directory. Each problem runs in a fresh sandbox.

B.2 Auto-formalization Evaluator Design

Two of the evaluated models, GPT-5.6-Sol and Claude-Opus-5, also act as evaluators. This differs fundamentally from LLM-as-a-judge setups (Zheng et al., 2023), since the evaluator renders no verdict and its sole function is to search for Lean proofs, $S \vdash S^\star$ and $S^\star \vdash S$ for BEq and a refutation for validity, all of which are accepted or rejected by the Lean kernel. Equivalence and refutation are therefore established by the kernel rather than by the agent's opinion, and no disposition toward a particular statement can make Lean accept a proof that does not typecheck. The only latitude an evaluator has is how hard it searches, which mainly affects recall, since the gates of Appendix B.3 can only reject. We use two evaluators so that every model, including the two evaluators themselves, which check each other, is always assessed by at least one external evaluator rather than being judged solely by itself. BEq accepts equivalence verified by either evaluator and validity requires no refutation from either, so in both cases an evaluator contributes only Lean-certified proofs and never withholds credit from another model, while the union of the two yields tighter estimates of the true correctness rate.

Recall of the agentic protocol. To quantify the recall gained by agentic proof search, we rescore every auto-formalization output with BEq$^+$ (Poiroux et al., 2025), a deterministic symbolic metric that attempts each direction of the equivalence with a fixed tactic script and no language model,

namely `exact?` restricted to the other statement, then `apply` or `convert` followed by automation such as `tauto` and `simp_all_arith!` (Table 9). BEq$^+$ certifies at most 5.8% of any model's statements, whereas the gated agentic protocol certifies 27.5% for Claude-Opus-5. It also fails to separate models, since all ten closed-source cells lie between 3.3% and 5.8%, whereas agentic BEq spans 13.3% to 27.5%. It reflects the transformations that research-level statements require. Unfolding a definition, reordering binders, or rewriting a bound into an equivalent form is rarely closed by a fixed tactic script, so symbolic search leaves many equivalences uncertified. To confirm that agentic certifications are genuine, the authors manually inspected 20 randomly sampled pairs certified by agentic BEq, comprising 40 directional proofs, and found every proof to be a genuine derivation of one statement from the other.

Table 9: Auto-formalization equivalence rates (%) on 120 theorems under the symbolic BEq$^+$ metric (Poiroux et al., 2025) and under our gated agentic BEq protocol.

Model	**BEq$^+$**		**Agentic BEq (ours)**	
	CoT	**Agent**	**CoT**	**Agent**
GPT-5.6-Luna	4.2	5.0	13.3	17.5
GPT-5.6-Terra	5.0	5.0	13.3	18.3
GPT-5.6-Sol	5.8	4.2	20.8	25.0
Claude-Sonnet-5	3.3	5.0	15.0	17.5
Claude-Opus-5	5.0	5.0	23.3	27.5
DeepSeek-V4 Flash	1.7	3.3	10.0	15.0
Gemma-4 26B-A4B	0.0	0.0	0.0	0.0
Qwen3.5-9B	0.0	0.0	0.0	0.0

B.3 Gates against Spurious Equivalence

Because every gold statement is a true theorem, a Lean proof of $S \to S^\star$ can in principle ignore S and prove $S^\star$ from scratch, and when S is itself true the reverse direction can ignore $S^\star$ in the same way. Mutual derivability would then certify only that both statements are provable, not that they mean the same thing. The original BEq avoids this by restricting the evaluator to a whitelist of tactics and by requiring the transformed goal to be definitionally equal to the other statement (Liu et al., 2025b). We adopt the *Normal* tactic whitelist of Liu et al. (2025b), namely `apply`, `cases'`, `constructor`, `exact`, `exact?`, `ext`, `have`, `intro`, `intros`, `rw`, and `use`. Because our evaluators are agents, we additionally pass every accepted direction through three gates that target the three ways a proof can succeed without a genuine derivation. The *strip gate* removes the hypothesis encoding the source statement from the theorem signature and recompiles the unchanged proof; a proof that still compiles never used the source and is rejected. The *no-source gate* removes the source statement's declaration from the file and reruns the evaluator's proof search on the target alone under the same budget; if the target can be proved without the source, the direction carries no evidence of equivalence and is rejected. The *bridge gate* targets proofs that use the source statement but supply a substantive auxiliary lemma absent from it, which would let a weaker statement pass. The evaluator prompt forbids introducing such lemmas, and an LLM screen inspects every accepted proof for added mathematics; flagged directions are rejected. A pair counts as BEq-equivalent only if both directions survive all three gates, and all BEq figures in the paper are computed after gating.

C Additional Forward-Construction Analysis

C.1 Theorem-Proving

Unless stated otherwise, theorem-proving trajectory analyses use medium effort and a time limit of 2,400 seconds. They cover 60 problems at the graduate level and 20 papers at the research level with 60 checked targets. Observable actions are tool calls or text output events. Output tokens include thinking.

C.1.1 Answer-File Error Trajectories

Across graduate and research problems, the mean number of Lean errors falls below one at the sixth compilation for Opus, compared with the seventeenth for Sol (Figure 6). Opus is therefore substantially more efficient than Sol at reducing errors in terms of compilation rounds. Even the strongest model still relies on repeated Lean feedback to refine its proof attempts. A compilation without errors does not guarantee a complete proof, since `sorry` produces only a warning.

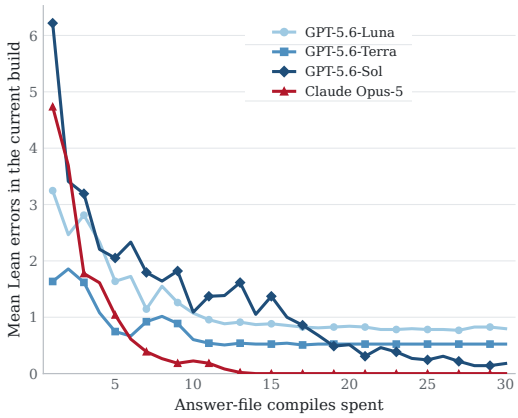


Figure 6: **Answer-file error trajectories.** Mean Lean errors by compilation count, carrying final counts forward for completed runs. Both graduate and research levels are pooled; runs require at least three answer-file compilations, excluding scratch probes.

C.1.2 Budget Curves and Per-Problem Spend

Opus gains coverage faster as the action budget increases, but Sol uses fewer tokens per solved theorem. On graduate problems, Sol uses about 36k tokens per solved theorem, compared with 59k for Opus (Figure 7(c)). Opus uses fewer but longer actions, so its advantage in action efficiency does not translate into lower token cost per solved theorem.

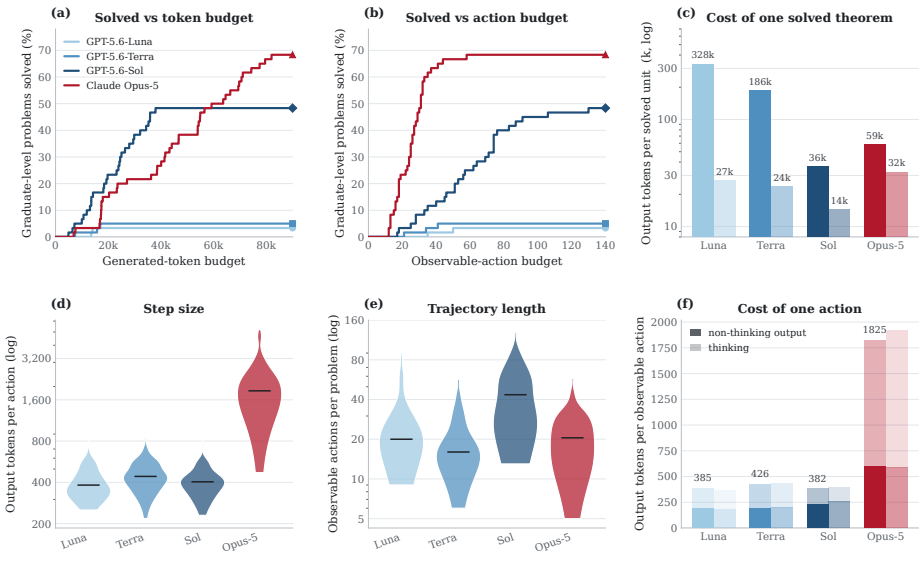


Figure 7: **Budget curves and per-problem spend.** (a)–(b) Solve rate at the graduate level by token and action budget. (c) Tokens per solved unit. (d) Tokens per action. (e) Actions per problem. (f) Non-thinking and thinking tokens per action. Panels (c) and (f) distinguish the graduate level (saturated) from the research level (lighter); (d)–(e) pool both. GPT thinking counts use `reasoning_output_tokens`; Opus uses harness-reported estimates as a lower bound.

C.1.3 Decomposition Quality

We test three specialized provers on 60 graduate theorems each, filtering for non-empty output, non-trivial decomposition, well-typed subgoals, and a recognizable strategy judged by GPT-5.6-Sol using predefined rubrics. Only 18.9% of attempts have well-typed subgoals; none are judged useful, and Sol completes none of their proof skeletons (Figure 8). This result shows specialized Lean provers do not have the ability to decompose ML theorems into useful subgoals.

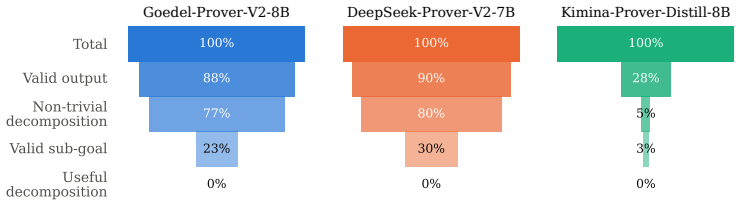


Figure 8: Decomposition funnel on 60 theorems per model; percentages use all 60 attempts.

C.1.4 Repeated Sampling Does Not Rescue Specialized Provers

The 0.0% rates reported in Table 3 are obtained under a single attempt per problem, which leaves open whether repeated sampling would recover some successes. We therefore re-evaluate the three specialized provers with pass@32, counting a problem as solved if any of 32 independent samples yields a Lean-accepted proof without `sorry` or newly introduced axioms. Table 10 shows that all three remain at 0.0% on both splits.

Table 10: Specialized provers under pass@32 (%).

Model	**Graduate**	**Research**
Kimina-Prover-Distill-8B	0.0	0.0
DeepSeek-Prover-V2-7B	0.0	0.0
Goedel-Prover-V2-8B	0.0	0.0

C.1.5 Evaluation on miniF2F

On miniF2F, the three specialized provers all solve at least 55.0% of the problems, on par with GPT-5.6-Sol and above Terra and Luna. The ranking reverses on the subgoals in Table 6, where no prover exceeds 18.7% while even the weakest GPT-5.6 variant reaches 52.0%. This suggests that the provers' ability does not generalize from competition-style mathematics to machine learning theory. The results of miniF2F are in Table 11.

Table 11: Pass@1 on a 20-problem miniF2F subset (%).

Model	**Pass@1**
GPT-5.6-Luna	30.0
GPT-5.6-Terra	50.0
GPT-5.6-Sol	60.0
Kimina-Prover-Distill-8B	60.0
DeepSeek-Prover-V2-7B	55.0
Goedel-Prover-V2-8B	65.0

C.2 Proof-Verification

C.2.1 Counterexample Construction Behavior

Even Opus needs about five attempts to reach 50% accuracy (Figure 9). This suggests that models rely mainly on repeated machine verification to find counterexamples through trial and error. Their

mathematical intuition alone remains insufficient to reliably identify why a claim fails and construct a counterexample directly.

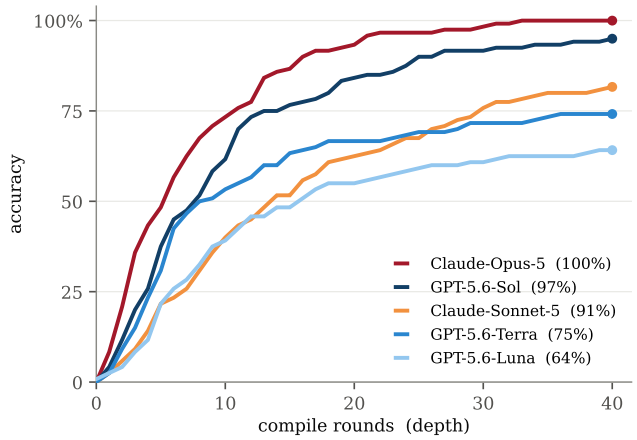


Figure 9: Counterexample accuracy by Lean compilation round. The horizontal axis is truncated at 40 rounds; the values in the legend are the final success rates after all rounds (cf. Table 7).

D Additional Backward-Verification Analysis

D.1 Error Criteria

Errors are assessed at the earliest invalid step. A correct final conclusion or a later correction does not excuse an earlier error. ML papers and even textbooks routinely tolerate loose bounds, compressed notation, and omitted routine conditions. To adapt our criteria to how ML theory is actually written, Table 12 lists the error types subject to the exemptions in Table 13.

Table 12: Error types and criteria for proof verification.

Type	**Criterion**
False statement or computation	An equation, inequality, constant, index, or calculation is invalid as written.
Invalid inference	A necessary case is omitted or a quantifier shift is invalid.
Unjustified application	A required assumption is unavailable or a cited result does not apply, subject to the exemptions below.

Table 13: Exemptions from error annotation in proof verification.

Case	**Criterion**
Loose bound	The bound remains valid.
Notation, or source compression	The intended mathematics is unambiguous or valid under a standard convention; brevity alone is not an error.
Omitted boundary case	The case is vacuous, trivially true, or excluded by the hypotheses.
Omitted regularity details	Routine conventions.
Implicit justification	"Standard" or "clearly" alone is not an error; flag only when the intended result can be identified and shown not to apply.
Unrepeated assumption	The assumption appears elsewhere in the supplied context.

D.2 Formalization Analysis

Our benchmark evaluates models in a realistic theory-review setting: they receive natural-language papers, not gold formalizations, and use Lean as a tool to locate errors. Formalization is therefore

part of the model's reasoning process rather than a trusted input. We measure success by exact agreement with the annotated first-error location, or by correctly reporting no error for a fully correct group. A correct prediction thus counts as task success even if an intermediate formalization is imperfect; this measures error-localization utility. Proof verification can be viewed as a composition of the two forward-construction tasks. The model first formalizes each NL step into a Lean statement and then proves or refutes that statement. Auto-formalization and theorem-proving (Tables 2 and 3) evaluate these two abilities in isolation, so proof verification measures how well models combine them to localize errors.

Since auto-formalization reaches at most 27.5% BEq (Table 2), step formalizations are unlikely to be fully faithful either. We therefore separately analyze formalization faithfulness. Lean can certify proofs of altered versions of erroneous steps, and models can produce opposite verdicts for the same source step. Table 14 reports these two diagnostics, Flawed Step Proved and Peer Conflict, in agent mode; they reveal specific failure modes rather than measure overall formalization accuracy.

Let c index the twenty configurations of model, verification rule, and mode, and let s denote a source step. Write $P_{c,s}$ for its generated Lean proposition and $V_{c,s} \in \{\mathsf{proved}, \mathsf{refuted}, \mathsf{none}\}$ for a proof, a refutation, or no verdict.

Flawed Step Proved. Lean accepts a proof of the generated proposition at the annotated erroneous step. When the source inference is false, this indicates that the model has changed its meaning, for example by adding an assumption that removes the error. Let E_s indicate an annotated erroneous step and $B_{c,s}$ indicate that the proposition and proof pass the substantive filter. This filter excludes conclusions that merely repeat hypotheses, combine hypotheses, assert `True`, or relate an expression to itself, as well as proofs using only trivial tactics or direct reuse of hypotheses.

$$\mathrm{FSP}_{c,s} = \mathbf{1}\{E_s = 1,\ V_{c,s} = \mathsf{proved},\ B_{c,s} = 1\}\,. \tag{1}$$

Peer Conflict. The same model proves and refutes its respective formalizations of the same source step across its two agent configurations. Let $n_{\mathrm{p}}(s)$ and $n_{\mathrm{r}}(s)$ count these two groups. Their smaller count measures the extent of disagreement. Opposite verdicts imply that the formalizations cannot all preserve the same source meaning, but do not establish which translation is wrong. This diagnostic does not depend on the benchmark label, and the minority verdict is not necessarily the incorrect one.

$$\mathrm{PCC}_s = \min(n_{\mathrm{p}}(s), n_{\mathrm{r}}(s))\,. \tag{2}$$

Table 14: Formalization diagnostic rates in agent mode.

Model	**Flawed Step Proved** ($n = 60$)	**Peer Conflict** ($n = 120$)
Claude-Opus-5	26.7%	13.3%
GPT-5.6-Sol	13.3%	29.2%

Missing verdicts can hide both patterns, while translations that change the meaning without changing the truth value can escape these checks. Low rates therefore do not by themselves establish faithful formalization.

D.3 Agentic NL versus FL Refute

To measure what the Lean pipeline adds beyond NL, we run an agentic NL baseline on the same 120 problems. We compare this with FL Refute in the agentic setting through the same harnesses as Table 4. Under NL the agent receives no Lean project. It works on the theorem group in natural language and must either declare the group correct or name the flawed theorem, the step, and the error class. Requiring the class guards against guessing, since naming the right step without the right reason earns no credit. To avoid over penalizing, each flawed group is annotated with two admissible error classes, and matching either one counts. An NL prediction on a flawed group is correct only when theorem, step, and class all match, and a flag on the wrong step or with the wrong class counts as both a false positive and a false negative, as in Section 2. Table 15 reports the four metrics.

Table 15: NL versus FL Refute on the 120 proof-verification problems, both in the agentic setting (%). NL accuracy counts every correct group the model passes, whereas FL Refute passes a correct group only when every step compiles. ΔF1 is FL Refute minus NL in percentage points.

Model	Method	Precision	Recall	Accuracy	F1	ΔF1
Claude-Opus-5	NL	59.2	48.3	69.2	53.2	+8.5
	FL Refute	61.7	61.7	67.5	61.7	
Claude-Sonnet-5	NL	43.9	30.0	60.8	35.6	+6.2
	FL Refute	46.0	38.3	55.0	41.8	
GPT-5.6-Sol	NL	42.5	56.7	55.0	48.6	−10.4
	FL Refute	30.9	50.0	39.2	38.2	
GPT-5.6-Terra	NL	41.3	51.7	55.0	45.9	−16.0
	FL Refute	27.0	33.3	32.5	29.9	
GPT-5.6-Luna	NL	33.7	48.3	43.3	39.7	−21.8
	FL Refute	15.3	21.7	19.2	17.9	

Claude verifies conservatively, GPT verifies too strictly. Under NL the two families fail in opposite ways. Both Claude agents have precision well above recall, 59.2 against 48.3 for Opus-5. They rarely flag a correct group, and the flags they raise are usually right, but they let many real errors through. All three GPT agents have recall above precision. They catch most real errors but object to correct proofs almost as often. One family under-reports and the other over-reports, and this difference decides what Lean does for each of them.

Lean helps Claude by making the machine expose the error. The gain for Claude is recall. Under NL, Opus-5 recovers only 48.3 of the real errors, because a conservative verifier does not commit to an error it merely suspects. Under FL Refute it does not have to. The agent formalizes the suspicious step, the kernel confirms the counterexample, and the machine supplies the confidence the agent lacked. Recall rises to 61.7 for Opus-5 and from 30.0 to 38.3 for Sonnet-5, while precision stays where it was, so every additional flag is a real error that NL had let through.

Lean hurts GPT by certifying its over-reporting. For GPT both precision and recall fall, and one habit explains both. Over-reporting gets correct groups flagged and gets flawed groups flagged too early, before the real error, so the earliest-flag rule reports a sound step instead. The agents are already over-strict, and unfaithful formalization aggravates it, since Lean certifies a refutation of the agent's own formalization of a sound step as readily as one of a genuine error.

E Case Studies

These cases illustrate theorem proving, proof verification, and the verification failure modes analyzed in Appendix D. For the theorem-proving cases, the displayed Lean programs are shortened to the theorem interface and the proof steps most relevant to the case; the benchmark files contain the complete compilable files.

E.1 Graduate-Level Forward Construction

The graduate-level item `ch13:proposition:13_2` (Hardt & Simchowitz, 2020, Section 13, Proposition 13.2) characterizes the normal cone of an intersection with an affine set under a relative-interior constraint qualification.

Natural-language problem

Let $\Omega \subseteq \mathbb{R}^d$ be closed and convex, and let $\mathcal{A} = \{x : Ax = b\}$. Suppose that $\mathrm{ri}(\Omega) \cap \mathcal{A} \neq \varnothing$. Then, for any $x \in \Omega \cap \mathcal{A}$,

$$N_{\Omega \cap \mathcal{A}}(x) = N_\Omega(x) + \{A^\top \lambda : \lambda \in \mathbb{R}^n\}.$$

Lean statement (proof omitted)

```
def normalCone {E : Type*} [NormedAddCommGroup E] [
InnerProductSpace Real E]
    (S : Set E) (x : E) : Set E :=
  {u | forall z, z ∈ S -> inner Real (z - x) u <= 0}

theorem ch13_proposition_13_2
    {d n : Nat}
    (Omega : Set (EuclideanSpace Real (Fin d)))
    (hOmegaClosed : IsClosed Omega) (hOmegaConvex : Convex Real
Omega)
    (A : EuclideanSpace Real (Fin d) →ℓ[Real] EuclideanSpace Real (
Fin n))
    (b : EuclideanSpace Real (Fin n))
    (affine : Set (EuclideanSpace Real (Fin d)))
    (hAffine : affine = {x | A x = b})
    (hCQ : (intrinsicInterior Real Omega ∩ affine).Nonempty)
    (x : EuclideanSpace Real (Fin d)) (hx : x ∈ Omega ∩ affine) :
    normalCone (Omega ∩ affine) x
      = normalCone Omega x
        + Set.range (fun mu : EuclideanSpace Real (Fin n) =>
            (LinearMap.adjoint A) mu) := by
  sorry
```

Gold proof. The following Lean skeleton is schematic: ellipses denote omitted context, statements, and proof steps, not a standalone compilable proof.

Gold proof (Lean skeleton)

```
have hxOmega := hx.1
have hxAffine := hx.2
obtain ⟨x0, hx0ri, hx0Affine⟩ := hCQ
...
have hdiffH : x0 - x ∈ H := by ...
have hdiffL : x0 - x ∈ L := by ...
have hdiffS : x0 - x ∈ S := by ...
have hgen (z : EuclideanSpace Real (Fin d))
    (hz : z ∈ Omega) : ... := by ...
have hstep : forall l : L, exists c : Real, 0 < c /\ ... := by
  ...
apply Set.Subset.antisymm
· ...
· ...
```

E.2 Research-Level Forward Construction

The research-level item `v10-duchi09a::theorem6` bounds the regret of forward–backward splitting (FoBoS) (Duchi & Singer, 2009).

Natural-language problem

Assume $\|w_t - w^*\| \le D$ for all iterations and that the subgradient sets ∂f_t and ∂r are bounded in norm by G. For any $c > 0$, FoBoS with $\eta_t = c/\sqrt{t}$ satisfies

$$R_{f+r}(T) \le 2GD + \left(\frac{D^2}{2c} + 7G^2c\right)\sqrt{T}.$$

Lean statement (proof omitted)

```
def IsFoBoSOnlineRun (f : Nat -> E -> Real) (r : E -> Real)
    (w gf gr : Nat -> E) (eta : Nat -> Real) : Prop :=
  (forall t, HasSubgradientAt (f t) (gf t) (w t)) /\
  (forall t, HasSubgradientAt r (gr t) (w t)) /\
  (forall t, w (t + 1) = w t - eta t • gf t - eta t • gr (t + 1))

def Regret (f : Nat -> E -> Real) (r : E -> Real)
    (w : Nat -> E) (wstar : E) (T : Nat) : Real :=
  (Finset.Icc 1 T).sum
    (fun t => f t (w t) + r (w t) - (f t wstar + r wstar))

theorem theorem6
    (f : Nat -> E -> Real) (r : E -> Real)
    (w gf gr : Nat -> E) (eta : Nat -> Real) (D G c : Real)
    (wstar : E)
    (hconvf : forall t, ConvexOn Real Set.univ (f t))
    (hconvr : ConvexOn Real Set.univ r)
    (hfnn : forall t x, 0 <= f t x) (hrnn : forall x, 0 <= r x)
    (hrun : IsFoBoSOnlineRun f r w gf gr eta)
    (hD : forall t, 1 <= t -> norm (w t - wstar) <= D) (hDnn : 0 <=
 D)
    (hGf : forall t x g, HasSubgradientAt (f t) g x -> norm g <= G)
    (hGr : forall x g, HasSubgradientAt r g x -> norm g <= G)
    (hGnn : 0 <= G) (hc : 0 < c)
    (hEta : forall t, 1 <= t -> eta t = c / Real.sqrt t)
    (T : Nat) (hT : 0 < T) :
    Regret f r w wstar T
      <= 2 * G * D + (D ^ 2 / (2 * c) + 7 * G ^ 2 * c) * Real.sqrt
T := by
  sorry
```

Gold proof. The displayed statement omits the surrounding namespace and ambient-space context, and its placeholder only suppresses the proof for presentation. As above, ellipses in the Lean skeleton denote omitted material.

Gold proof (Lean skeleton)

```
obtain ⟨hsf, hsr, hupd⟩ := hrun
set A := fun t => norm (w t - wstar) ^ 2
set B := fun t : Nat => Real.sqrt t / (2 * c)
have hstep : forall t, t ∈ Finset.Icc 1 T -> ... := by
  ...
have hreg : Regret f r w wstar T <= ... := by
  ...
have hsum2 :
    (Finset.Icc 1 T).sum
      (fun t => (c / Real.sqrt t / 2) * G ^ 2)
      <= c * G ^ 2 * Real.sqrt T := by
  ...
have hAD : forall t, 1 <= t -> A t <= D ^ 2 := by
  ...
...
```

```
have habel := abel_bound A B (D ^ 2) hAD hB0 hBm T hT
have hlast : r (w 1) - r (w (T + 1)) <= 2 * G * D := by
  ...
linarith
```

E.3 Agentic Proof Verification

For this case, we choose an instance with a wrong proof step (Yang et al., 2019). The flawed step is Eq. (40) of arXiv version 2 of that paper; equation numbering differs across its versions. Claude-Opus-5 correctly localizes the error to T2 step 3 under agentic FL Refute.

Source theorem and erroneous step

The mirror policy optimization theorem uses the averaged gradient estimator

$$\widehat{g}_k = \frac{1}{k}\sum_{i=1}^{k} g_i, \qquad g_i = \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau_i)\Big|_{\theta=\theta_i}.$$

Under bounded score and Hessian of $\log \pi_\theta$, it claims the following bound on the expected squared Bregman gradient at $\widetilde{\theta}_N$:

$$\frac{J(\theta^*) - J(\theta_1) + (\sigma^2/\zeta)\sum_{k=1}^{N} \alpha_k/k}{\sum_{k=1}^{N}(\zeta\alpha_k - L\alpha_k^2)}.$$

Step 3 of the proof, Eq. (40), asserts

$$\mathbb{E}[-g_k] = \mathbb{E}[-\widehat{g}_k] = -\nabla J(\theta_k).$$

The error. Even when each trajectory estimator is unbiased at its own iterate, linearity of expectation gives

$$\mathbb{E}[\widehat{g}_k] = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[\nabla J(\theta_i)],$$

not, in general, $\mathbb{E}[\nabla J(\theta_k)]$. The running average is therefore not generally unbiased at the current iterate.

Lean-checked refutation. The agent formalizes T2 step 3 as a claim that a sequence's running mean always equals its final term. Taking $\mathrm{gradJ}(i) = i$ and $k = 2$ gives $1.5 \neq 2$:

Refutation of the formalized step

```
theorem refute_running_mean :
    Not (forall (gradJ : Nat -> Real) (k : Nat), 1 <= k ->
      (1 / (k : Real)) * (Finset.Icc 1 k).sum gradJ = gradJ k) :=
by
  intro h
  have := h (fun i => (i : Real)) 2 (by norm_num)
  norm_num [show Finset.Icc 1 2 = ({1, 2} : Finset Nat) from rfl]
at this
```

Lean certifies this refutation of the formalized claim; its relevance to the source step depends on faithful translation.

FL Prove versus FL Refute. Both agentic modes return the gold location, T2 step 3, but use different evidence. FL Prove reports the first elaborated step it cannot prove; FL Refute requires a Lean-checked refutation.

Table 16: Claude-Opus-5 on item `SkxpDT4YvS` (arXiv:1906.10462v2) in agent mode.

	FL Prove	**FL Refute**
Rejection evidence	Failure to prove	Lean-checked refutation
Reported location	T2 step 3 (correct)	T2 step 3 (correct)
Steps proved	11	10
Other step outcomes	2 failures of proof	3 refutations
Affected steps	T2 step 3; T3 step 2	T2 steps 3–4; T3 step 2
Refutations accepted	0 (not requested)	3 of 3 offered

F PROMPTS

F.1 AUTO-FORMALIZATION

System prompt

```
You are formalizing mathematics in Lean 4 (Mathlib).

`{answer}` in your working directory is a Lean file whose target theorem has had
its statement replaced by `<<HOLE 1>>`. Your job is to write that statement.

WORKFLOW (do these in order, do not stop early):
 1. Read `{answer}`.
 2. EDIT `{answer}`, replacing `<<HOLE 1>>` with the statement.
 3. Compile:  cd {project} && lake env lean --json $OLDPWD/{answer}
 4. If it reports an error, edit and compile again. Repeat until the only
    message is the expected `declaration uses 'sorry'` warning.
 5. Only then reply DONE, and paste the FINAL compiler output verbatim in your reply, which is the
      exact text `lake env lean` printed on your last run.

Step 3 is not optional and not something to skip when the statement "looks right". Writing a Lean
      statement that elaborates is most of this task; an uncompiled answer is usually wrong in a way
      you cannot see by reading it.

Your answer is read from the FILE, not from your message. A run that never edits `{answer}` scores
      zero, however good the reasoning in your reply.

* You MAY and SHOULD look things up in Mathlib. Its source is readable at `{mathlib}` -- use `rg`
      there when you are unsure of a name, a notation or a binder form (for example whether a sum
      over a finset is written `in` or the correct alternative), and `#check` / `#print` in a scratch
      file for a signature. Getting the spelling from Mathlib is not cheating; guessing it is how
      runs fail.
* Do NOT search for this theorem, its statement, or the benchmark it comes from. They are not on
      this filesystem and turns spent looking are wasted. The mathematics you are formalizing is in
      the natural language below and nowhere else.
* Keep the theorem's name, keep its proof as exactly `sorry` -- you are writing a STATEMENT, not a
      proof -- and change nothing else in the file. The definitions above the theorem are fixed and
      the statement must be phrased in terms of them.
* Say exactly what the natural language says: no more general, no more specific.
  Compiling is necessary, not sufficient.
```

User prompt

```
TARGET THEOREM: `{target_name}`

THE STATEMENT TO WRITE (currently a placeholder):
```lean
{masked_statement}
```

NATURAL LANGUAGE:
{nl}

Edit `answer.lean`, compile it from `{project}`, and iterate until it is clean.
```

F.2 THEOREM-PROVING

System prompt

```
You are an expert Lean 4 / Mathlib prover working AUTONOMOUSLY.

You are given a Lean 4 source file ON DISK whose target theorems (and any helper lemmas) end in
      'sorry'. Your job: edit that file IN PLACE until it COMPILES CLEANLY with every 'sorry'
      replaced by a real proof.

Work autonomously and persist in solving the target theorems:
- Edit the file directly (Edit/Write/apply_patch).
- Compile-check it yourself as often as you like with:
    cd {lean_project} && lake env lean {workfile}
  It imports Mathlib; a clean compile prints no errors. A remaining 'sorry' shows as 'declaration
      uses 'sorry''.
- Search Mathlib for useful definitions and lemmas. If you cannot quickly find a suitable lemma,
      PROVE THAT STEP YOURSELF (an inline 'have', or your own private helper lemma derived from
      Mathlib). A missing library lemma is NEVER a reason to leave a 'sorry'.

Hard rules:
- Fill EVERY 'sorry'/'admit'; introduce NO 'axiom'.
- Do NOT change any theorem/lemma STATEMENT or any 'def' body; only replace proof text after ':='.
      You MAY add your own private helper lemmas. (Modifying a fixed statement will be rejected by
      the grader.)
- Do NOT edit anything under the Mathlib library; edit only your task file.
- The GRADER re-compiles your file independently; your own compile runs don't count, so make sure
      the file on disk is actually correct.

Persistence & scoring (READ THIS CAREFULLY):
- Persist in solving the target theorems. Do NOT stop or give up early because the remaining targets
      look hard; keep working, searching Mathlib, and trying different approaches. The harness takes
      WHATEVER is on disk as your answer, so partial progress is never wasted: always leave the file
      in the best state you can reach.
- Each target theorem is scored INDEPENDENTLY (per-target '#print axioms'): a target counts as
      solved iff it elaborates AND nothing in its proof term uses 'sorry'. A 'sorry' left in an
      UNRELATED target or helper does NOT hurt a target you finished.
- So PRIORITISE FINISHING WHOLE TARGETS, one at a time. Pick the most tractable target (each is
      flagged '-- TARGET «...»' right above its declaration), prove it end-to-end together with every
      helper it depends on (no 'sorry' anywhere in its dependency chain), and only then move to the
      next. Fully proving 1 of 3 targets scores 1/3; spreading effort so all three still end in
      'sorry' scores 0, so do not do that.
- KEEP THE FILE ELABORATING. Leave targets you have not finished as a literal 'sorry' (it still
      elaborates) rather than half-written tactic blocks that throw hard errors, since a hard error
      elsewhere can stop a FINISHED target from elaborating and lose its point.
```

User prompt

```
[Your task file: EDIT THIS FILE IN PLACE]
{workfile}

[Compile it with]
cd {lean_project} && lake env lean {workfile}

[Target theorems to prove: each is SCORED INDEPENDENTLY, so partial credit applies. They are also
      flagged with '-- TARGET «...»' right above their declaration in the file. Do NOT change their
      statements.]
{targets}


Keep working persistently; do not stop or give up early because the remaining targets look hard.
      Work autonomously: edit, compile, fix, repeat. Finish targets ONE AT A TIME (fully prove one,
      with all the helpers it depends on, before starting the next) so partial work earns partial
      credit. Keep the file elaborating and compiling with your best progress; the final file on disk
      is your answer.
```

F.3 PROOF-VERIFICATION

The system prompt is shared by FL Refute and FL Prove and contains no refutation-specific content. The user prompt below is for FL Refute; FL Prove uses the same text with the `REFUTATION` block, its contract, and the recap removed.

Shared system prompt

```
# What counts as the error
Judge the proof at the level of its STEPS. Report a step as flawed if it is invalid **as written**:
      a false (in)equality, a wrong constant or index, an inference that does not follow, a named
      result applied where its hypotheses fail, or an illegal quantifier move. That the theorem's
      stated conclusion happens to survive the slip is **not a defence**: a proof that reaches a true
      conclusion through a false line is still a flawed proof, and a later step silently using the
      correct value does not repair the wrong one. Trace any defect to its earliest origin step.
{exemptions}
# Error-type checklist
Judge for yourself whether each theorem's proof is correct; do NOT assume any particular one is
      flawed, and do NOT assume any error exists at all. Test every step of every theorem against
      EACH failure mode below. Work forward without skipping: process theorems in their listed order
      and steps in increasing local step-number order. Complete the ENTIRE bundle, even after
      verifying a fault. The error selected for reporting is the EARLIEST qualifying step across the
      whole bundle in that order, not one error per theorem; the evaluator selects it from your
      artefacts after checking them. For the structural types (assumption / quantifier), the error
      lives at the step that first ASSUMES or asserts the bad thing, NOT the later step where it
      visibly breaks; always trace any defect back to its origin.
{error_type_checklist}
# The bundle
Several theorem statements and proofs from the SAME paper are shown below as [T1], [T2], ... . They
      MAY contain a genuine mathematical error -- or they may ALL be correct. You are not told which.
      Step numbers are local to each theorem's own proof, never a global count across the bundle.
{upstream_formalization_rules}
# Method: step-level formalisation with Lean verification
You verify proofs by translating steps into Lean 4 and letting LEAN decide. Work however you like
      inside this session: draft a statement, check it, fix it, try a proof, check again.
WHAT YOU RETURN IS ARTEFACTS, NOT A VERDICT. Every statement and proof you hand back is re-checked
      against a fresh Lean environment before anything is scored, and your account of what the
      checker said is not read. So there is nothing to be gained by reporting a proof you did not get
      past the checker, and nothing lost by returning a statement you could not prove.
# What to return
One block per step, in order: theorems in the order they are listed, steps by their number. Do not
      skip a step because it looks obviously fine -- that judgement is exactly what this task takes
      away from you.
Use the checker available in this session to validate your artefacts. Do not stop after finding an
      error, even one accepted by the checker. Work to the end of every theorem in the bundle and
      return one block per step. After re-checking, the evaluator selects the EARLIEST qualifying
      error across the entire bundle, ordered first by the theorems' listed order and then by local
      step number.
    [[[ theorem=<local_id> step=<n> ]]]
    STATEMENT
    ```lean4
    theorem test : <the whole proposition> := by sorry
    ```
    PROOF
    ```lean4
    theorem test : <the same proposition> := by
      <tactics that close it>
    ```
{no_triage_rationale}
A step whose content is a definition or a naming -- "let m = D(F, eps, d)", "write phi(x) for
      (x-a)_+" -- still has a formal core: the defining equation. Write it.
{definition_step_example}
A step has nothing to formalise only if it states no equation, inequality, bound, or assertion about
      anything this proof has introduced -- a pointer outside this proof, or an announcement of what
      comes next. Difficulty is not a reason. Emit it as
    [[[theorem=<local_id> step=<n>]]]
    NO_CLAIM
and expect to need that rarely; more than one or two in a proof is triage. Every NO_CLAIM is counted
      and reported.
STATEMENT -- required, and its SHAPE is required
Formalize the step faithfully, so that the proposition carries the full mathematical content of the
      step as written, not a weakened, trivial, or tautological version of it.
Write the step as ONE Lean 4 theorem, proof left as `sorry`, in exactly this form:
    theorem test : <proposition> := by sorry
If the proof fixes an object, bind it once and constrain it -- do not universally quantify it and
      then require agreement.
Everything the step quantifies over goes INSIDE the proposition, as leading binders:
{binder_example}
Do NOT put binders to the left of the colon (`theorem test (B : Real) ... : ...`). The shape matters
      because the harness lifts the proposition out of it verbatim when it needs to negate it.
Do not close any goal with `plausible`, `native_decide`, `admit` or anything else that does not
      build a real proof term: every proof and refutation is re-checked with `#print axioms`, and
      anything resting on `sorryAx` or a non-standard axiom is discarded. (`by plausible` on a false
      statement is silently accepted by a naive check -- it is not accepted here.)
Nothing between blocks -- no prose, no summary, no verdict. Do NOT say which step is wrong.
```

User prompt (FL Refute)

```
## PROOF -- optional
The same theorem with a real proof. Omit this section if you could not prove it: an unproved
      statement is a legitimate outcome and a fabricated proof is not. Never write `sorry` here.
## REFUTATION -- optional, and this is how a step gets reported as WRONG
Failing to prove a statement is not disproof. It may be false, or it may be true and merely out of
      reach, or the paper may have granted something the formalisation could not carry. A step is
      reported as an error ONLY when Lean accepts a proof that its statement is FALSE.
You do not write that goal. The harness takes the proposition from your own STATEMENT, verbatim, and
      assembles
    theorem refutation : Not ( <your proposition> ) := by
      <your REFUTATION tactics>
so the thing being refuted is exactly the thing you formalised, and there is no way to refute an
      easier proposition instead. Your tactics start on the goal `Not (<proposition>)`.
Add the refutation as a third block of the step, after PROOF:
    REFUTATION
    ```lean4
    <tactics starting on `Not (<proposition>)`, usually beginning with `intro h` and then deriving
      `False`; include `intro h` yourself if used>
    ```
For a `forall`-shaped proposition the refutation is a counterexample: introduce the hypothesis,
      instantiate it at concrete values, and derive a contradiction, for example:
{counterexample_example}

You have a checker in the working directory:
{checker_usage}
{bundle}
```