FormalMATH: Benchmarking Formal Mathematical Reasoning of Large Language Models

Anonymous ACL submission

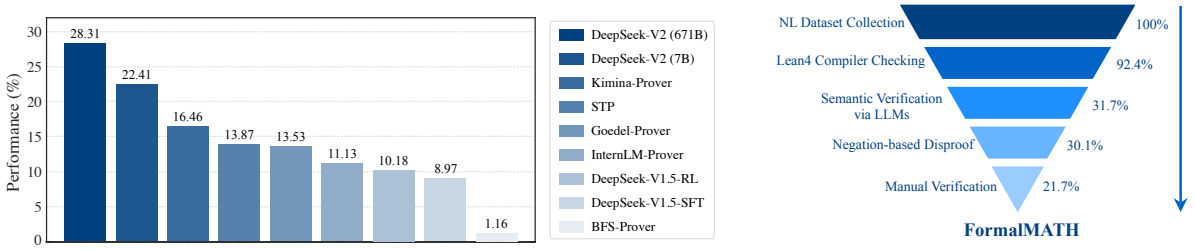
Abstract

We introduce FormalMATH, a large-scale Lean4 benchmark for formal mathematical reasoning, comprising 5,560 semantically validated Lean4 statements spanning competition-level and undergraduate mathematics. To improve reliability, FormalMATH is constructed through a human-in-the-loop pipeline that combines type checking, LLM-based hybrid symbolic alignment for specification-level semantic verification, negation-based disproving, and expert review. We conduct a systematic evaluation of frontier theorem provers under test-time scaling on FormalMATH. Results show that current provers remain far from solving the benchmark: the strongest model, DeepSeek-Prover-V2 (671B), achieves only 28.31% Pass@32. We further observe substantial domain imbalance across provers, with stronger performance in applied mathematics and algebra but weaker results in discrete mathematics. We identify common failure modes in current provers, including incomplete proofs, weak handling of inequalities, misuse of automation tactics, and redundant hypothesis introduction. Extreme test-time scaling (*e.g.*, Pass@3200) yields only modest gains on a subset of FormalMATH, indicating limitations in current training paradigms. We also observe a counterintuitive negative effect when provers are given human-written natural-language solution references, suggesting that such references can misguide formal proof generation. Overall, FormalMATH provides a challenging and reliable testbed for measuring progress in formal mathematical reasoning.

1 Introduction

Informal mathematical reasoning benchmarks are becoming increasingly saturated, limiting their ability to evaluate frontier models in mathematical reasoning (Hendrycks et al., 2021; Cobbe et al., 2021; Luong et al., 2025; Dekoninck et al., 2026). Formal mathematical reasoning is a specialized form of math reasoning grounded in formal languages (Leino, 2010; Barras et al., 1997; Moura & Ullrich, 2021), which provides a rigorous axiomatic framework for automated proof verification. This setting is inherently challenging for humans (Brasca et al., 2025; Gonthier et al., 2008; Avigad et al., 2007). Recent advances in language modeling (Brown et al., 2020; Shao et al., 2024; Bai et al., 2023) have made them increasingly promising for formal reasoning tasks.

Although several formal mathematics benchmarks (Zheng et al., 2021; Azerbayev et al., 2023; Tsoukalas et al., 2024; Ren et al., 2025; Liu et al., 2023) have been proposed and widely adopted for evaluating LLMs on autoformalization and theorem proving (Wu et al., 2022; Liu et al., 2025b; Lu et al., 2024b; Yang et al., 2023; Polu & Sutskever, 2020; Xin et al., 2024; Polu et al., 2023; Ren et al., 2025; Chervonyi et al., 2025; Wang et al., 2025; Hubert et al., 2025), they are increasingly insufficient for differentiating frontier systems and diagnosing the remaining bottlenecks in formal mathematical reasoning. First, existing benchmarks are often limited in both scope and scale, which makes them vulnerable to benchmark-specific optimization and weakens the reliability of evaluation. MiniF2F (Zheng et al., 2021), for instance, contains only 244 test problems and is largely restricted to high-school-level algebra and number theory, with recent theorem provers achieving success rates exceeding 88.9%. ProofNet (Azerbayev et al., 2023) includes only 186 problems and focuses primarily on undergraduate analysis and algebra. Even for benchmarks on which current models already obtain high scores, most validation procedures check only surface-level similarity between informal mathematical statements and their formal counterparts, rather than specification-level alignment (Zheng et al., 2021; Liu et al., 2023; Gao et al., 2025; Peng et al., 2025; Ospanov et al., 2025). Consequently, benchmark instances may still contain substantial



(a) Performance of provers on FormalMATH

(b) Data preservation

Figure 1: (a) **Performance of theorem provers on FormalMATH.** The best-performing model, DeepSeek-Prover-V2(671B), achieves only 28.31% Pass@32 on FormalMATH. (b) The percentage of data that is preserved after each filtering stage in our formalization pipeline.

UG Domain	Int	Pre	Alg	Diff	MC	SS	Other
Percentage (%)	30.8	18.9	15.9	11.0	9.6	8.9	4.9

High School Domain	Alg	NT	AM	DM	Geo	Pre	Cal	Other
Percentage (%)	50.2	22.3	9.0	8.6	5.8	1.9	1.9	0.3

Table 1: **Domain distribution in FormalMATH.** FormalMATH is a comprehensive benchmark spanning a broad range of domains. Abbreviations: Int=Integral, Pre=Precalculus, Alg=Algebra, Diff=Differentiation, MC=Multivariable Calculus, SS=Sequences and Series, NT=Number Theory, AM=Applied Mathematics, DM=Discrete Mathematics, Geo=Geometry, Cal=Calculus.

formalization faults, including misalignments between informal statements and their formal specifications (Ammanamanchi & Bhat). Such errors can make evaluation results unreliable, since a prover may solve a formal statement that does not faithfully capture the original mathematical problem.

To address these limitations, we introduce *FormalMATH*, a large-scale Lean4 benchmark containing 5560 formal statements. To enhance the reliability, we construct FormalMATH through a human-in-the-loop validation pipeline that combines Lean4 type checking, LLM-assisted specification-level semantic verification, negation-based disproof, and expert review. This pipeline is designed to go beyond surface-level correspondence between informal and formal statements, ensuring that each retained Lean4 theorem preserves the variables, constraints, assumptions, and target claims of the original mathematical problem. FormalMATH spans a broad range of mathematical domains, covering both high-school competition problems and undergraduate-level mathematics.

Our evaluation on FormalMATH yields several key findings. First, current provers remain far from solving the benchmark: the strongest model, DeepSeek-Prover-V2 (671B), reaches only 28.31% Pass@32. This sharp drop compared with prior benchmarks suggests that previous high scores may partly be artifacts of limited benchmark scale, narrow domain coverage, and saturation, rather than evidence of general formal reasoning capability. Second, extreme test-time scaling brings only modest improvements, indicating that formal proof generation is not simply a matter of sampling more candidate proofs. The bottleneck instead lies in the depth and structure of reasoning, including lemma use, problematic hypothesis introduction, and long-horizon tactic construction. Third, adding human-written natural-language solution references can hurt proof generation. Such references provide high-level mathematical intuition but often fail to specify the explicit formal tactics required by Lean, exposing the limitations of current natural-language-augmented training paradigms for formal reasoning. Our contributions include:

- **A Large and Comprehensive Lean4 Benchmark.** We present FormalMATH, a challenging benchmark comprising 5560 diverse, semantically validated mathematical statements.
- **Reliable Pipeline for Semantic Alignment Validation.** We develop a reliability-oriented construction pipeline that combines compiler checking, specification-level semantic verification, negation-based disproof, and expert review.
- **Comprehensive Empirical Evaluation on Formal Reasoning.** We provide a systematic evaluation of frontier Lean4 theorem provers, showing that current systems remain far from solving FormalMATH, with the strongest prover achieving only 28.31% Pass@32.
- **Intriguing Findings in Formal Reasoning.** We reveal several nontrivial phenomena in current formal reasoning systems: extreme test-time scaling brings only modest gains, LLM-based Lean provers exhibit failure modes such as incomplete proofs, automation misuse, weak inequality handling, and redundant hypotheses, and human-written natural-language references can even degrade proof generation.

2 FormalMATH: A Large Formal Mathematical Reasoning Benchmark

2.1 Overall Dataset Statistics

FormalMATH is a large Lean4 benchmark comprising 5560 semantically validated Lean statements, constructed through a human-in-the-loop pipeline that combines type checking, LLM-based semantic verification, and expert review. It spans a broad range of domains and difficulty levels, from high-school competition problems in algebra, number theory, discrete mathematics, and geometry to undergraduate-level topics such as calculus, linear and abstract algebra, and sequences and series (see Table 1). The original problems are collected from diverse informal sources (Table 6), leading to diversity in both topical coverage and formalization difficulty. Best-performing frontier model DeepSeek-Prover-V2 (671B) achieves 28.31% Pass@32 on FormalMATH. See data samples and reliability audit in AppendixA.1 and B.1.

2.2 Data Format and Decontamination

FormalMATH provides a Lean4-compatible JSON format dataset (Table 7). Unlike existing benchmarks (Tsoukalas et al., 2024; Zheng et al., 2021; Azerbayev et al., 2023), we provide natural-language solutions for each problem, to support analysis beyond reference-free theorem proving (*e.g.*, experiments in Section 4.4). FormalMATH also supports other tasks, including Autoformalization, proof informalization, etc. To reduce benchmark-level overlap, we compare FormalMATH against public Lean datasets released before its construction. After normalizing Lean statements by removing non-semantic formatting, theorem names, imports, and comments, we remove exact duplicates and potential near-duplicates based on statement-level similarity. For model developers, we recommend explicitly controlling for possible data leakage in training when testing models on FormalMATH.

3 Formalization and Semantic Correctness Checking

3.1 Formalization Pipeline

Given an informal statement, we sample multiple Lean candidates with autoformalizer and remove those failed type-checking. We use HSA-Verifier to check specification-level semantic alignment and retain only faithful candidates. To further reduce false positives, we apply negation-based disproving to eliminate candidates contradicted by formal proof search. The remaining high-confidence formalizations are finally sent to expert review. Figure 1 reports the preservation at each stage, and Figure 6 presents the overall pipeline. A final reliability audit shows strong annotator agreement, achieving a Gwet's AC1 of 0.9795, with all disagreements adjudicated by a third expert (See Appendix Section B.1).

3.2 HSA-Verifier: A Structured Verifier for Semantic Validation

General Framework of HSA-Verifier. HSA decomposes both the informal statement x and the Lean theorem statement y into five channels: variables and domains $\mathcal{V}$, atomic constraints $\mathcal{A}$, quantified constraints $\mathcal{Q}$, structural assumptions $\mathcal{S}$, and target statement $\mathcal{T}$. It then checks whether they align on each channel:

$$\mathcal{C}(s) = \big(\mathcal{V}, \mathcal{A}, \mathcal{Q}, \mathcal{S}, \mathcal{T}\big), \quad s \in \{x, y\}. \tag{1}$$

We apply symbolic normalization and expansion (*e.g.*, interval $\Rightarrow$ bound implications, positivity-derived constraints, and target canonicalization) to reduce superficial mismatches in surface form. We then align $\mathcal{C}(x)$ and $\mathcal{C}(y)$ channel-wise and obtain structured alignment evidence (Figure 2(a)):

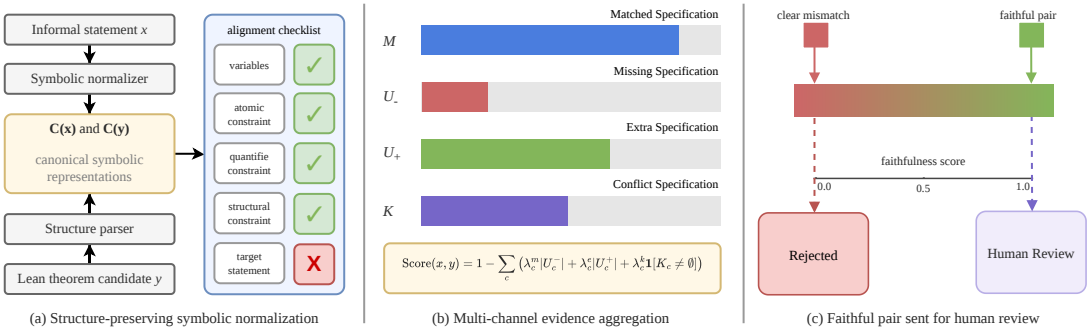
$$\mathcal{E}(x, y) = \{M_c, U_c^-, U_c^+, K_c\}_{c \in \{\mathcal{V}, \mathcal{A}, \mathcal{Q}, \mathcal{S}, \mathcal{T}\}} \tag{2}$$

where M_c denotes matched evidence, U_c^- missing evidence, U_c^+ extra evidence, and K_c conflicting evidence in channel c. For interpretability, we define an alignment score (Figure 2(b)):

$$\begin{aligned}\mathrm{Score}(x, y) = 1 - \sum_{c \in \mathcal{C}} \Big(&\lambda_c^m |U_c^-| + \lambda_c^e |U_c^+| \\ &+ \lambda_c^k \mathbf{1}\{K_c \neq \emptyset\}\Big).\end{aligned} \tag{3}$$

The score is used only as an interpretable diagnostic rather than a standalone decision rule. In our pipeline, only full-alignment pairs, *i.e.*, those with no missing, extra, or conflicting evidence, are sent to human review (Figure 2(c)).

LLM-Assisted Constraint Extraction and Verification. Purely symbolic checking is brittle because many formalization errors arise from semantic underspecification rather than syntactic mismatch. Informal statements may omit domain conditions, leave quantifier scope implicit, or rely on convention-dependent readings that are difficult to capture with fixed symbolic rules alone. HSA therefore uses an LLM to extract structured constraints from both sides and to adjudicate residual uncertainty, while requiring that all decisions be



(a) Structure-preserving symbolic normalization (b) Multi-channel evidence aggregation (c) Faithful pair sent for human review

Figure 2: **Overview of the HSA-Verifier**. HSA-Verifier provides finer-grained signals for detecting specification-level alignment than previous methods.

Method	ACC	TPR	FPR	TNR	FNR
Baselines					
Llama-3.3-70B-Instruct	0.682	0.952	0.588	0.412	0.048
Claude3.5-Sonnet	0.742	0.972	0.488	0.512	0.028
Qwen3-235B-A22B	0.848	0.904	0.208	0.792	0.096
Doubao-1.5-pro-32k	0.784	0.952	0.384	0.616	0.048
DeepSeek-R1	0.840	0.908	0.228	0.772	0.092
GPT-5-mini	0.814	0.736	0.108	0.892	0.264
DeepSeek-V3	0.798	0.712	0.116	0.884	0.288
Ours					
HSA-Verifier(GPT-5-mini)	**0.852**	0.788	0.084	0.916	0.212
HSA-Verifier(DeepSeek-V3)	**0.848**	0.824	0.128	0.872	0.176
HSA-Verifier(Claude3.5-Sonnet)	**0.830**	0.876	0.216	0.784	0.124

Table 2: **Semantic Validation Test on CriticLean-Bench**. HSA-Verifier consistently improves LLMs' ability to judge whether informal mathematical statements and Lean theorem statements are semantically aligned.

grounded in explicit symbolic alignment evidence. Thus, HSA preserves the interpretability of symbolic matching while remaining robust to linguistic variation and extraction noise (see Section C.4).

3.3 Evaluation and Analysis.

We compare HSA-Verifier with CriticLean-style LLM verifiers on CriticLeanBench (Peng et al., 2025) (prompts in Appendix C.4; Avg@4 in Table 2, we treat faithful formalizations as the positive class). We use temperature 0.6 and generation length 8,192 tokens. With GPT-5-mini, HSA-Verifier reaches 0.852 ACC, improving over its baseline 0.814 by +0.038. HSA-Verifier reaches 0.830 with Claude3.5-Sonnet, improving its baseline (0.742) by +0.088.

HSA verifies specification-level equivalence, not text-level similarity. Semantic correctness requires preserving the admissible solution set, i.e., $\mathcal{M}(x) = \mathcal{M}(y)$. HSA explicitly tests this condition via structured constraint alignment. In Example F.1, omitting $\sin x \neq 0$ for $\cot x$ changes the admissible domain, CriticLean accepts the statement based on surface-level similarity, whereas HSA correctly rejects it. In Example F.2, HSA separates target semantics from premise constraints and rejects a statement that appears formula-consistent but encodes a different optimization target. In Example F.3, HSA enforces type-consistent alignment in the final inequality and detects a non-trivial Nat/Real domain mismatch that CriticLean misses. These cases show that HSA reduces false positives by validating objective- and type-level faithfulness beyond surface expression matching.

Informal ambiguity remains challenging. Residual errors often arise not from failures of semantic alignment itself, but from under-specified or ambiguous informal statements. Such ambiguity can appear either in expression semantics or in proof intent. For example, $\sqrt[3]{\cdot}$ may denote the real cube-root function rather than exponentiation by $1/3$ (Example F.4), while phrases such as "using the Lebesgue Dominated Convergence Theorem" may be interpreted either as part of the formal specification or merely as a suggested proof strategy (Example F.5). These cases illustrate an inherent limitation of semantic validation under ambiguous natural-language inputs: when the informal statement does not uniquely determine the intended formal meaning, no verifier can resolve the ambiguity without additional assumptions or human clarification.

3.4 Disproving a Statement by Proving Its Negation.

We further filter out incorrect formalization candidates using off-the-shelf LLM-based provers (*e.g.*, DeepSeek-Prover-V1.5). For each formalized statement $\mathbf{T}_n^{(k)}$, we construct its logical negation $\neg\mathbf{T}_n^{(k)}$ via rule-based transformations, and then run proof search on the negated statement with Pass@32. If $\neg\mathbf{T}_n^{(k)}$ is successfully proved in the Lean system, the original statement is refuted under its stated assumptions and is discarded. Section B shows an example of disproving a number-theoretic con-

jecture. In total, this step filters out 1.6% of the collected candidates. See example in Appendix Section B.

4 Evaluate Provers on FormalMATH

4.1 General Settings

Provers, Sampling, and Metrics. We focus on the following proof-generation approaches:

- **Best-First Tree-Search (BFS) Methods.** Each node in the search tree is an intermediate proof state. A heuristic scoring function assigns a priority to each node. We evaluate: BFS-Prover (Xin et al., 2025), DeepSeek-Prover-V1.5-RL (Xin et al., 2024), and InternLM-V2.5-Prover (Wu et al., 2024) under BFS.
- **Single-Pass Generation Methods.** The models generate a complete proof in one pass. We consider the following models: STP (Dong & Ma, 2025), DeepSeek-Prover-V1.5s (Xin et al., 2024), DeepSeek-Prover-V2 (Xin et al., 2024), EvolProver (Tian et al., 2025), Goedel-Prover (Lin et al., 2025), and Kimina-Prover-7B (Wang et al., 2025).

Following convention in formal mathematical reasoning, we report Pass@K (Chen et al., 2021) alongside accuracy. For BFS, the total search budget is $N \times S \times T$, where each of the N attempts performs T expansion steps and considers S tactics at each step. For SPG, K is the total number of complete proof trajectories sampled. For a fair comparison, we align the number of top-level search attempts and set $K = N$.

Prompts. We adopt the same prompt template as Xin et al. (2024).

LLM Hyperparameters and Environment. We use a sampling temperature of 0.7 and top_p of 0.95. Both the Lean toolchain and Mathlib are based on version v4.15.0.

4.2 Evaluating Formal Theorem Provers on FormalMATH

Existing Provers Are Far from Solving FormalMATH. Pass@32 results are shown in Figure 1. Within the DeepSeek-V1.5, RL reaches 10.18%, compared with 8.97% for SFT. When DeepSeek-V1.5-RL is with BFS, performance drops to 1.16%, highlighting the difficulty of scaling BFS with heterogeneous training methods. STP and Goedel-Prover achieve 13.87% and 13.53%, respectively. Similar to general language modeling (Kaplan et al., 2020), results suggest that scaling model and data size is more effective than post-training refinements alone: DeepSeek-Prover-V2 (671B) reaches 28.31%, substantially improving over V1.5-based systems while still leaving the majority of FormalMATH unsolved.

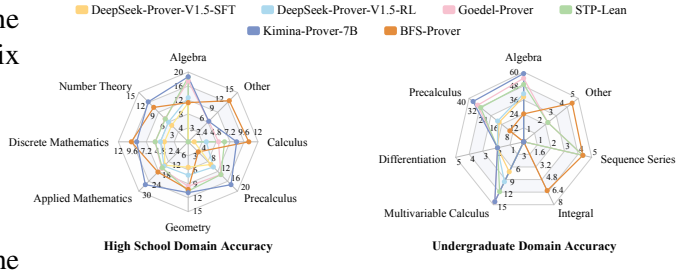


Figure 3: **Breakdown of accuracy across mathematical domains within FormalMATH.**

Model	Redun. Hyp.	Incomp. Proof	Ineq. Handle	Auto-Tac. Misuse
DeepSeek-SFT	18.0%	77.0%	8.0%	62.0%
DeepSeek-RL	34.0%	62.0%	13.0%	65.0%
Goedel	27.0%	86.0%	20.0%	78.0%
STP	24.0%	44.0%	1.0%	74.0%
Kimina	36.0%	93.0%	20.0%	43.0%

Table 3: **Error patterns in provers.**

Unbalanced Performance Across Domains. Figure 3 reveals domain bias in existing theorem provers. Goedel-Prover achieves stronger performance in algebra-related domains (*e.g.*, 17.47% in high school algebra and 50% in undergraduate algebra) but performs poorly in calculus (5.21%) and discrete math (0%). One possible explanation is a mismatch between prover training distributions and FormalMATH domains. For example, Goedel-Prover's training data (See Figure 7a) is concentrated in applied mathematics and algebra, while several FormalMATH domains require different reasoning patterns. Domain-specific performance differences may also arise because different areas of math require different reasoning patterns (*e.g.*, symbolic manipulation versus analysis-heavy argumentation) and differ in the extent to which they are supported by reusable lemmas and automation-friendly infrastructure in Lean (See Section 4.3).

4.3 Common Error Patterns in FormalMATH Proof Attempts.

We manually annotate 100 failed proof attempts sampled from each prover and identify common failure patterns on FormalMATH (see Table 3). These categories are non-exclusive, so the percentages do not sum to 100%. Representative cases are shown in Table 12.

Improper Use of Automation Tactics. Provers generate proofs that rely heavily on automation tactics (*e.g.*, `aesop` (Limperg & From, 2023), `simp`, and `linarith`), to streamline the low-level, step-

by-step reasoning required by tactic-based proofs. For example, `aesop` performs a best-first proof search over a database of tagged lemmas and applies rewriting, splitting, and instance search to discharge goals. But these tactics depend on fixed heuristics and pre-tagged lemmas that may not match the structure of every proof: when over-invoked or misconfigured, they can dramatically expand the search space, lead to nontermination or timeouts, or even transform goals into irrelevant or unsolvable forms. In particular, automated tactics often struggle to supply the explicit constructions or witnesses required by truly constructive proofs (Mandelkern, 1985), which may discharge the main proposition without building the underlying data, resulting in incomplete or invalid reasoning. Proof `omni_theorem_4000` (Table 12) fails to construct a witness a within the correct domain that satisfies both (1) $a \leq 1 \vee a > 0$ and (2) $f(x) = \begin{cases} 0, & \text{if } x \neq -a^2 \\ a, & \text{if } x = -a^2 \end{cases}$. Instead of performing case-by-case analysis, the proof, however, introduces the incorrect witness $a = 0$, and relies on `simp` to close off the remaining goals that are not designed to solve, without specifically analyzing the core function $(x+y^2)\cdot f(y\cdot f(x)) = x\cdot y\cdot f(y^2+f(x))$.

Inabilities to Handle Complex Inequalities. Provers over-rely on `linarith` and `nlinarith` when handling goals that mix linear constraints with non-linear inequalities. This strategy becomes brittle when the proof requires reasoning over rational expressions, cyclic structure, and intermediate inequality bounds rather than a direct contradiction search. For the failed proof `algebra_528739` (Table 12), a direct `nlinarith` attempt is ineffective. To eliminate the denominators, the prover would first need to introduce the common denominator $D = (a^3+b^3+abc)(b^3+c^3+abc)(c^3+a^3+abc)$.. Expanding D produces a degree-9 polynomial in three variables with roughly 55 possible monomials $\binom{9+3-1}{3-1} = \binom{11}{2} = 55$, which already indicates substantial algebraic blow-up. Even after normalization, the proof still does not reduce to a form that `nlinarith` can solve directly: one must first identify suitable term-wise bounds, apply monotonicity of reciprocals under positivity, and only then combine the resulting cyclic inequalities. This example reflects not merely a failure of algebraic normalization, but a broader inability to construct the intermediate deductive steps required in non-trivial inequality proofs.

Redundant Hypothesis Introduction. Provers introduce structurally redundant hypotheses. While these do not inherently cause logical errors, they obscure the proof's underlying logic and reduce readability. For example, in the `aime_all_2005_II_1` proof (Table 12), the unnecessary use of `revert` followed by `reintro` exemplifies this issue. These tactics are designed to generalize variables or hypotheses, a technique critical for inductive proofs or hypothesis strengthening. However, in this case: (1) no inductive reasoning requires generalization, (2) the variables `n`, `hn`, and `h` already exist in the context and can be directly used. Thus, the tactic `revert` is redundant and can be removed to simplify the proof.

Incomplete Proof. Provers generate unfinished proof that leaves critical subgoals unresolved or rely on placeholder tactics without justifying intermediate steps. In `DEMIMathAnalysis_50` (Table 12), which aims to show $\lim_{n\to\infty} \sqrt{n} \cdot \int_{-\infty}^{\infty} \frac{1}{(1+x^2)^n} dx = \sqrt{\pi}$, prover terminates prematurely after a few tactic calls that: (1) fail to justify interchanging the limit and integral and (2) fail to establish bounds on the integrand's tail decay. The flawed proof begins with an unnecessary rewrite of `sqrt` and misapplies monotonicity lemmas like `integral_mono_on` without verifying domination or integrability conditions required for the Dominated Convergence Theorem. Worse, tactics such as `tendsto_atTop_of_eventually_const` and `filter_upwards` trivialize tail behavior instead of rigorously addressing convergence. We hypothesize this error stems from short-sighted heuristic selection during language modeling of theorem provers: prioritizing tactics that maximize immediate log-probability or heuristic scores (*e.g.*, `gcongr`, `norm_num`, `simp`) over those advancing global proof progress. Such choices syntactically reshape goals while burying core challenges under shallow subgoals.

4.4 Evaluating Test-time Scaling of Formal Theorem Provers.

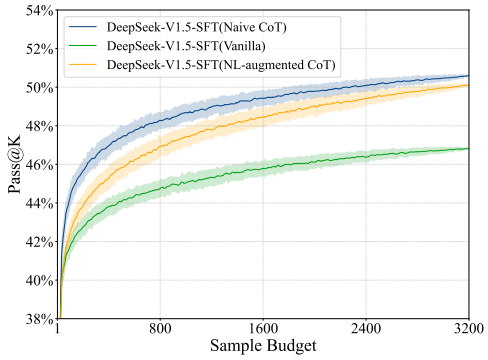
Exploring Extremely Large Sampling Budgets. Test-time scaling is a zeroth-order optimization procedure, using large sampling budgets to increase the probability of finding high-reward proofs. To support rapid, stable, and diagnostically meaningful iteration in formal reasoning research, we introduce an easier and smaller subset for FormalMATH (see Appendix H). We further scale the sampling

Method	1	4	8	16	32
Best-First Tree Search Methods					
BFS(InternLM-V2.5)	7.87	15.79	20.02	22.74	25.65
BFS(DS-Prover-V1.5-RL)	6.81	24.70	28.47	30.65	32.71
BFS(BFS-Prover)	27.10	34.04	37.56	41.75	45.88
	32	**512**	**1024**	**2048**	**3200**
Single-Pass Generation Methods					
Kimina-Prover-7B	48.94	–	–	–	–
STP	48.59	51.45	52.03	52.60	53.17
DS-Prover-V1.5-SFT	40.40	44.17	45.08	46.12	46.82
DS-Prover-V1.5-RL	47.98	49.27	49.68	50.08	50.35
Goedel-Prover	46.70	48.68	49.04	49.20	49.41
Evol-Prover	54.01	55.62	56.13	56.86	57.41
DS-Prover-V2(7B)	51.76	54.11	54.11	54.82	55.06
DS-Prover-V2(671B)	56.00	60.00	61.18	61.88	61.88
Ensemble					
Ensemble of SPG Methods (4 × 3200)	54.11	–	–	–	–

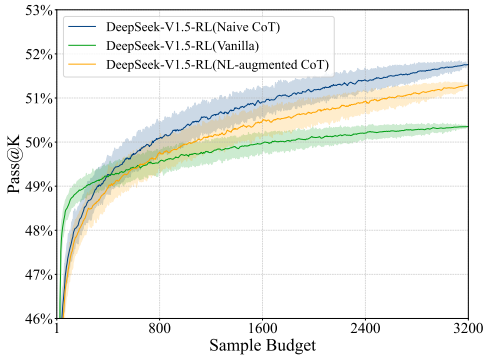
Table 4: **Test-time Scaling Performance of LLMs**. Provers exhibit marginal improvement at Pass@3200. For BFS methods, the actual search budget should be multiplied by 32×100, corresponding to the number of expansions and tactics considered at each step.

budget from 32 to 3,200 samples to study theorem-proving under extreme test-time scaling.

Subtle Performance Enhancement via Test-time Scaling. Table 4 reveals limited returns when applying test-time scaling to formal theorem proving on FormalMATH subset. For instance, STP achieves only a 4.58% absolute improvement (from 48.59% at Pass@32 to 53.17% at Pass@3200) despite a 100× sampling budget increase. While BFS-Prover demonstrates better scaling dynamics, attaining an 18.78% gain (27.10% via Pass@$1\times32\times100$ to 45.88% via Pass@$32\times32\times100$), it still underperforms SPG methods even under a 32× budget expansion. Ensembling SPG methods (*i.e.*, via composing STP, Goedel-Prover, DeepSeek-V1.5-SFT, and DeepSeek-V1.5-RL) yields only marginal gains, from 53.17% by STP alone to 54.11%, a mere 0.84% improvement. This observation stands in sharp contrast to the near-linear scaling gains reported for informal reasoning (Muennighoff et al., 2025). Possible explanation is that informal reasoning tolerates a substantial degree of imprecision during sampling: correct conclusions can sometimes emerge despite imperfect reasoning chains. Formal theorem proving, by contrast, offers no such tolerance. A single incorrect tactic, malformed term, or type error invalidates the entire proof trajectory, making naive increases in sampling budget far less effective. Although verifier-guided proof search is in principle better suited to mitigating this brittleness than SPG-style methods, current implementations remain computationally expensive and do not yet scale efficiently.



(a) DeepSeek-V1.5-SFT



(b) DeepSeek-V1.5-RL

Figure 4: **Pass@1-3200 accuracy for DeepSeek-V1.5 provers.**

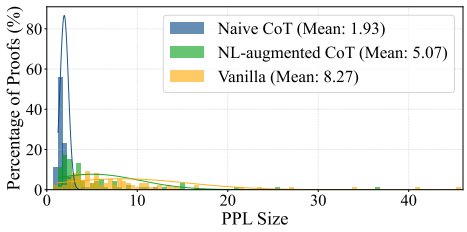


Figure 5: **Perplexity of DeepSeek-V1.5-SFT.** Human-written natural-language proofs have a negative effect on perplexity.

Negative Effect of Human Reference in Formal Reasoning. We classify prompting in DeepSeek-V1.5 into three categories (details in Appendix C): (1) Vanilla: no CoT is required; (2) Naive-CoT: CoT is required, but no informal human NL solution is provided; and (3) NL-augmented CoT: CoT is required, and an informal human NL solution is included in the prompt. We implement these CoT prompts on top of DeepSeek-V1.5s under test-time scaling. We find Naive-CoT improves over vanilla, while NL-augmented CoT does not consistently improve over Naive-CoT and reduces Pass@3200 (see Figure 4). Under $K = 3200$, DeepSeek-V1.5-SFT achieves 50.6% with Naive-CoT, 49.2% with NL-augmented CoT, and 47.0% with vanilla decoding; DeepSeek-V1.5-RL achieves 51.7%, 51.2%, and 49.8%, respectively. Figure 5 shows that NL-

Benchmark	# Problems	Difficulty
MiniF2F	244	Olympiad
ProofNet	186	Undergraduate (UG)
FIMO	149	Olympiad
PutnamBench	522	Olympiad
ProverBench	325	Olympiad
FormalMATH	**5,560**	**Olympiad & UG**

Table 5: **Comparison of Formal Benchmarks.** FormalMATH is larger and more challenging than previous benchmarks.

augmented CoT can increase model perplexity relative to naive CoT on Lean4 problems (used here as a proxy for uncertainty), which is consistent with the observation that high-level informal guidance may not map cleanly onto tactic-level proof obligations.

Case Analysis. Appendix G illustrates why NL-augmented CoT can hurt Lean4 proof generation. Although the informal solution and the Lean script aim at the same mathematical contradiction, the NL outline remains too high-level: it works `modulo 7` and states the residue facts $x^3 \bmod 7 \in \{0, 1, 6\}$ and $y^4 \bmod 7 \in \{0, 1, 2\}$, but does not translate them into the explicit case hypotheses required by Lean tactics. Consequently, when the proof invokes `omega`, the context lacks the concrete congruence constraints needed to close the branches. By contrast, the successful naive CoT proof constructs a tactic-level argument from scratch. It chooses modulus 13, explicitly splits all residue cases, introduces hypotheses such as `have h : x^3 % 13 = 5 := by ...`, and immediately discharges each branch with `omega`. This suggests that natural-language references may provide correct high-level ideas while still failing to supply the low-level proof obligations required by Lean4, thereby misguiding formal proof generation.

5 Related Works and Concluding Remarks

Autoformalization and Benchmarking in Formal Math Reasoning. Autoformalization (Szegedy, 2020; Polu & Sutskever, 2020; Wu et al., 2022; Liu et al., 2025b; Jana et al., 2025; Li et al., 2024) is challenging in ensuring semantic correctness (Li et al., 2024; Lu et al., 2024a; Gao et al., 2025) for Lean 4. (Tsoukalas et al., 2024; Azerbayev et al., 2023; Zheng et al., 2021) were handcrafted over many months by teams of PhD students. (Ying et al., 2024; Liu et al., 2023; Ren et al., 2025; Liu et al., 2025a) use LLMs (OpenAI, 2023; Jaech et al., 2024; Shao et al., 2024) for autoformalization, with selective manual validation to verify correctness. Years of efforts spent in correcting Minif2f (Xin et al., 2024; Wang et al., 2025; Ren et al., 2025; Ospanov et al., 2025; Achim et al., 2025) and Putnambench (Tsoukalas et al., 2024) showcase difficulties in guaranteeing semantic correctness for current human-AI collaboration pipelines and characteristics of evolution (Kiela et al., 2021; Hardt, 2025). (Ospanov et al., 2025; Ammanamanchi & Bhat) highlight misformalizations in formal benchmarks, including incomplete or incorrect specifications and translation. FormalMATH enhances robustness in semantic verification by introducing hybrid verification and negation-based disproving to address issues in (Ammanamanchi & Bhat). In addressing issues in (Ammanamanchi & Bhat), we build a large and comprehensive benchmark in formal math reasoning (See Table 5).

LLM-based Provers and Test-time Scaling. FormalMATH systematically evaluates frontier provers under test-time scaling, including short-CoT models (Xin et al., 2024; Dong & Ma, 2025; Lin et al., 2025; Tian et al., 2025; Xin et al., 2025; Wu et al., 2024), and long-CoT models (Wang et al., 2025; Ren et al., 2025). Advanced proof search algorithm including plain search (Xin et al., 2024; Dong & Ma, 2025; Lin et al., 2025; Ren et al., 2025), BFS (Xin et al., 2025; Lin et al., 2024; Yang et al., 2023), self-reflection (Yu et al., 2025), MCTS (Xin et al., 2024), and collaboration with informal reasoner (Jiang et al., 2022; Varambally et al., 2025). Our evaluation reveals important observations: unbalanced performance across domains, failure mode when inserting an informal solution written by humans, and highlighting common error patterns for proving FormalMATH (*e.g.*, improper use of automated tactics), etc.

Concluding Remarks We introduce FormalMATH, a novel and extensive benchmark for formal mathematical reasoning in LLMs. It provides a challenging and reliable testbed for evaluating theorem provers and advancing future research in formal reasoning. Comprising 5,560 semantically validated statements in Lean4, it spans a wide range of mathematical domains, including algebra, number theory, calculus, and discrete mathematics, from high-school Olympiad to undergraduate curricula.

6 Limitations

Mathlib and Lean 4 Versioning. FormalMATH focuses on Lean 4 theorem statements, so its coverage is shaped by the expressiveness of Lean 4,

the maturity of Mathlib, and the availability of suitable source problems. FormalMATH was originally formalized against Mathlib v4.15.0 and has since been updated to support the latest Mathlib version, v4.28.0. We will continue to update the benchmark to maintain compatibility with future Mathlib releases. In the current version, FormalMATH is primarily written in Lean 4 and does not directly support other formal systems, such as Isabelle and Coq. We plan to release a compact cross-system version in future work.

References

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, et al. Aristotle: Imo-level automated theorem proving. *arXiv preprint arXiv:2510.01346*, 2025.

Pawan Sasanka Ammanamanchi and Siddharth Bhat. Faults in our formal benchmarks. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic (TOCL)*, 9(1):2–es, 2007.

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.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq proof assistant reference manual: Version 6.1*. PhD thesis, Inria, 1997.

Riccardo Brasca, Christopher Birkbeck, Eric Rodriguez Boidi, Alex Best, Ruben van De Velde, and Andrew Yang. A complete formalization of fermat's last theorem for regular primes in lean. *Annals of Formalized Mathematics*, 1, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Konstantin Chernyshev, Vitaliy Polshkov, Ekaterina Artemova, Alex Myasnikov, Vlad Stepanov, Alexei Miasnikov, and Sergei Tilga. U-math: A university-level benchmark for evaluating mathematical skills in llms. *arXiv preprint arXiv:2412.03205*, 2024.

Yuri Chervonyi, Trieu H Trinh, Miroslav Olšák, Xiaomeng Yang, Hoang H Nguyen, Marcelo Menegali, Junehyuk Jung, Junsu Kim, Vikas Verma, Quoc V Le, et al. Gold-medalist performance in solving olympiad geometry with alphageometry2. *Journal of Machine Learning Research*, 26(241):1–39, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Kefan Dong and Tengyu Ma. Beyond limited data: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Fan Fan, Sarah Martinson, Erik Wang, Kaylie Hausknecht, Jonah Brenner, Danxian Liu, Nianli Peng, Corey Wang, and Michael Brenner. Hardmath: A benchmark dataset for challenging problems in applied mathematics. In *International Conference on Learning Representations*, volume 2025, pp. 13523–13556, 2025.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *International Conference on Learning Representations*, volume 2025, pp. 55720–55762, 2025.

Georges Gonthier et al. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

Moritz Hardt. The emerging science of machine learning benchmarks. *Manuscript. https://mlbenchmarks.org*, 2025.

Masayoshi Hata. *Problems and solutions in real analysis*, volume 14. World Scientific Publishing Company, 2016.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pp. 1–3, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Prithwish Jana, Kaan Kale, Ahmet Ege Tanriverdi, Cruise Song, Sriram Vishwanath, and Vijay Ganesh. Proofbridge: Auto-formalization of natural language proofs in lean via joint embeddings. *arXiv preprint arXiv:2510.15681*, 2025.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in nlp. In *Proceedings of the 2021 conference of the North American chapter of the Association for Computational Linguistics: human language technologies*, pp. 4110–4124, 2021.

K Rustan M Leino. Dafny: An automatic program verifier for functional correctness. In *International conference on logic for programming artificial intelligence and reasoning*, 2010.

Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Fan Yang, Xian Zhang, and Xiaoxing Ma. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. *Advances in Neural Information Processing Systems*, 37:53598–53625, 2024.

Jannis Limperg and Asta Halkjær From. Aesop: White-box best-first proof search for lean. In *ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2023.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, et al. Fimo: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*, 2023.

Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, Zhengfeng Yang, Jujian Zhang, Lihong Zhi, Jia Li, and Zhengying Liu. Combibench: Benchmarking llm capability for combinatorial mathematics, 2025a.

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.

Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. Formalalign: Automated alignment evaluation for autoformalization. *arXiv preprint arXiv:2410.10135*, 2024a.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, et al. Process-driven autoformalization in lean 4. *arXiv preprint arXiv:2406.01940*, 2024b.

Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35406–35430, 2025.

Mark Mandelkern. Constructive mathematics. *Mathematics Magazine*, 58(5):272–280, 1985.

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. minif2f-lean revisited: Reviewing limitations and charting a path forward. *arXiv preprint arXiv:2511.03108*, 2025.

Zhongyuan Peng, Yifan Yao, Kaijing Ma, Shuyue Guo, Yizhe Li, Yichi Zhang, Chenchen Zhang, Yifan Zhang, Zhouliang Yu, Luming Li, et al. Criticlean: Critic-guided reinforcement learning for mathematical formalization. *arXiv preprint arXiv:2507.06181*, 2025.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Christian Szegedy. A promising path towards autoformalization and general artificial intelligence. In *International Conference on Intelligent Computer Mathematics*, 2020.

Yuchen Tian, Ruiyuan Huang, Xuanwu Wang, Jing Ma, Zengfeng Huang, Ziyang Luo, Hongzhan Lin, Da Zheng, and Lun Du. Evolprover: Advancing automated theorem proving by evolving formalized problems via symmetry and difficulty. *arXiv preprint arXiv:2510.00732*, 2025.

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. *arXiv preprint arXiv:2407.11214*, 2024.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. *arXiv preprint arXiv:2509.22819*, 2025.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *NeurIPS*, 2022.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 2025.

Kaiyu Yang, Aidan 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 *NeurIPS*, 2023.

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. *Advances in Neural Information Processing Systems*, 37:105848–105863, 2024.

Zhouliang Yu, Yuhuan Yuan, Tim Z Xiao, Fuxiang Frank Xia, Jie Fu, Ge Zhang, Ge Lin, and Weiyang Liu. Generating symbolic world models via test-time scaling of large language models. *arXiv preprint arXiv:2502.04728*, 2025.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Appendix

Table of Contents

A Dataset in FormalMATH **12**
- A.1 Dataset Samples 12
- A.2 Data Source 13
- A.3 Evolving Benchmark in Formal Reasoning 13
- A.4 Data Format 14

B Details of Formalization Pipelines **14**
- B.1 Human Expert Validation. . . . 15
- B.2 Benchmark Reliability Audit . 16

C Prompts **17**
- C.1 Prompt for Vanilla Generation . 17
- C.2 Domain Classification 18
- C.3 Difficulty Classification 20
- C.4 Prompt for Verifier 21

D Common Errors in Proving FormalMATH **22**

E Typical Errors in Statement Autoformalization **22**
- E.1 Errors in Definition 22
- E.2 Errors in Expressions 24
- E.3 Errors in Constraint Condition . 25
- E.4 Errors in Proof Goals 26

F Case Study in Semantic Equivalence Checking **27**
- F.1 CriticLean ID 84 27
- F.2 CriticLean ID 148 28
- F.3 CriticLean ID 102 30
- F.4 Failed CriticLean ID 183 . . . 32
- F.5 CriticLean ID 210 34

G Fail Case in NL-augmented CoT **36**

H Smaller and Easier Subset for FormalMATH **36**

I LLM Usage Disclosure. **37**

A Dataset in FormalMATH

To support anonymous review, we include an anonymized supplementary material containing the benchmark statements, Lean evaluation commands, validation logs, and other supporting files needed for reliability and reproducibility checks. The full public release will be made after the anonymity period.

A.1 Dataset Samples

Example: The 27th derivative for $y = 2 \cdot x^2 \cdot \sin(x)$

u-math_915

Find the derivative of the 27th order $y^{(27)}$ for a function $y = 2 \cdot x^2 \cdot \sin(x)$. Prove that the answer is: $y^{(27)} = 1404 \cdot \cos(x) - 2 \cdot x^2 \cdot \cos(x) - 108 \cdot x \cdot \sin(x)$

Autoformalization:

```
import Mathlib

open Real Set
open scoped BigOperators

/-- Find the derivative of the 27th order y^(27) for a function
    y = 2 · x^2 · sin(x). Prove that the answer is: y^(27) =
    1404 · cos(x) − 2 · x^2 · cos(x) − 108 · x · sin(x) -/
theorem u_math_915 {f : ℝ → ℝ} (hf : f = λ x => 2 * x ^ 2 *
    sin x) :
  iteratedDeriv 27 f = λ x => 1404 * cos x – 2 * x ^ 2 * cos x
    – 108 * x * sin x := by
 sorry
```

Example: Find all positive integer solutions to a nonlinear divisibility equation

omni_theorem_4136

Determine all pairs (x, y) of positive integers such that $x^2y + x + y$ is divisible by $xy^2 + y + 7$.

Prove that the answer is: $(x, y) = (11, 1), (49, 1), (7t^2, 7t)$, t is an integer

Autoformalization:

```
import Mathlib

open Real

open scoped BigOperators

/-- Determine all pairs (x, y) of positive integers such that
    x^2y + x + y is divisible by xy^2 + y + 7.

Prove that the answer is:
    (x, y) = (11, 1), (49, 1), (7t^2, 7t), t is an integer

-/

theorem omni_theorem_4136 : ∀ x y : ℕ, x > 0 ∧ y > 0 → ((x ^
    2 * y + x + y) % (x * y ^ 2 + y + 7) = 0 ↔ (x = 11 ∧ y = 1)
    ∨ (x = 49 ∧ y = 1) ∨ ∃ t : ℤ, x = 7 * t ^ 2 ∧ y = 7 * t) :=
    by
 sorry
```

Example: Find the greatest positive integer

aime_all_2010_II_3

```
Let K be the product of all factors (b − a) (not
      necessarily distinct) where a and b are integers
      satisfying 1 ≤ a < b ≤ 20 . Find the greatest
      positive integer n such that 2^n divides K .
Prove that the answer is: 150
```

Autoformalization:

```
import Mathlib

open Finset

/--
Let K be the product of all factors (b − a) (not necessarily
      distinct) where a and b are integers satisfying
      1 ≤ a < b ≤ 20 . Find the greatest positive integer n
      such that 2^n divides K .
-/
def K : ℕ :=
  ∏ a in Icc 1 20, ∏ b in Icc 1 20, if a < b then (b − a) else 1

theorem aime_all_2010_II_3 :
  IsGreatest {n : ℕ | 0 < n ∧ 2 ^ n | K} 150 := by
 sorry
```

Example: Find the number of positive integers with three not necessarily distinct digits

aime_all_2010_I_1

```
Find the number of positive integers with three not
      necessarily distinct digits, abc , with a ≠ 0 and
      c ≠ 0 such that both abc and cba are multiples of 4 .
Prove that the answer is: 40
```

Autoformalization:

```
import Mathlib

/--
Find the number of positive integers with three not necessarily
      distinct digits, abc , with a ≠ 0 and c ≠ 0 such that both
      abc and cba are multiples of 4 .
-/
theorem aime_all_2012_I_1 :
  Set.ncard
   {n : ℕ | ∃ a b c : ℕ,
    a < 10 ∧ b < 10 ∧ c < 10 ∧
    a ≠ 0 ∧ c ≠ 0 ∧
    n = 100 ∗ a + 10 ∗ b + c ∧
    4 | (100 ∗ a + 10 ∗ b + c) ∧
    4 | (100 ∗ c + 10 ∗ b + a)} = 40 := by
 sorry
```

Example: Does there exist a function $f : \mathbb{N} \to \mathbb{N}$

olymid_ref_base_9735

```
22. (VIE 3) IMO4 Does there exist a function f : ℕ → ℕ,
      such that f(f(n)) = n + 1987 for every natural
      number n ?
```

Autoformalization:

```
import Mathlib

/--
22. (VIE 3) IMO4 Does there exist a function f : ℕ → ℕ, such
      that f(f(n)) = n + 1987 for every natural number n ?
-/

theorem olymid_ref_base_9735 :
  ¬ ∃ f : ℕ → ℕ, ∀ n : ℕ, f (f n) = n + 1987 := by
 sorry
```

Example: Find $f(1)$

olymid_ref_base_5198

```
If f(x) is a monic quartic polynomial such that
      f(−1) = −1, f(2) = −4, f(−3) = −9, and
      f(4) = −16, find f(1). Prove that the answer is: 23.
```

Autoformalization:

```
import Mathlib

open Polynomial

/--
If f(x) is a monic quartic polynomial such that
      f(−1) = −1, f(2) = −4, f(−3) = −9, and
      f(4) = −16, find f(1). Prove that the answer is: 23.
-/

theorem olymid_ref_base_5198
  (f : ℝ[X])
  (hf_monic : f.Monic)
  (hf_deg : f.natDegree = 4)
  (h₁ : f.eval (−1) = −1)
  (h₂ : f.eval 2 = −4)
  (h₃ : f.eval (−3) = −9)
  (h₄ : f.eval 4 = −16) :
  f.eval 1 = 23 := by
 sorry
```

A.2 Data Source

A.3 Evolving Benchmark in Formal Reasoning

Formal reasoning benchmarks face challenges such as semantic faults in formal statements that go beyond type checking (Ammanamanchi & Bhat), since no automatic mechanism can guarantee complete semantic alignment, correctness, and provability in all cases. We therefore advocate FormalMATH as a dynamic benchmark (Kiela et al., 2021; Hardt, 2025), allowing Lean 4 statements to be continually refined as human or future LLMs solve more problems and expose hidden flaws. When a problem is suspected to be flawed, a detailed judgment should be attached, together with an er-

Dataset	Level	#Domains	Size	#S.Formal	License
Omni-Math (Gao et al., 2024)	High School Olympiad	9	4.43k	1,210	CC BY-NC-SA 4.0
Numina-Olympiad	High School Olympiad	10	11.8k	2,409	Apache-2.0
AIME-Math	High School Olympiad	7	934	371	CC BY-NC-SA 4.0
BlueMO	High School Olympiad	8	3,024	1,099	CC BY-ND 4.0
U-Math (Chernyshev et al., 2024)	Undergraduate	6	1,100	358	MIT
HardMath (Fan et al., 2025)	Undergraduate	3	1,060	67	MIT
DEMIMATH (Hata, 2016)	Undergraduate	8	88	46	MIT

Table 6: Data sources used in FormalMATH. "#Domains" denotes the number of mathematical domains in the source dataset. "#S.Formal" denotes the number of samples in FormalMATH formalized from each source. We report source-level license information for attribution and release documentation.

ror tag indicating the fault type, and the statement may then be revised accordingly. Such judgments and refinements may be made by either humans or LLMs, provided they are fair and well-justified. To ensure fair evaluation under continuous updates, evaluators should report the exact benchmark version used, and every statement revision should be made transparent.

A.4 Data Format

Field	Description
ID	Stable statement identifier
Source	Source of the problem
Informal statement	Natural-language problem statement.
Formal statement	Lean 4 theorem statement.
Informal proof	Reference informal solution.
Domain	Domain tag.
Difficulty	Difficulty score following with rubric.
Dataset version	Dataset release tag.
Status	Item status in the living benchmark.
Changelog ref	Pointer to versioned change record.

Table 7: Format of the FormalMATH dataset.

B Details of Formalization Pipelines

Decontamination and Leakage Control. We conduct decontamination at Lean-level statements and natural-language problem statements level First, for Lean-level leakage, we compare FormalMATH against public Lean datasets released before the construction of our benchmark. We normalize Lean statements by removing theorem names, comments, imports, whitespace differences, and other non-semantic formatting artifacts, and then remove exact duplicates and high-similarity near-duplicates.

Second, because many source problems are originally written in natural language, we additionally check for natural-language statement overlap with Lean 4 benchmarks released before FormalMATH. We normalize informal statements by lowercasing, removing LaTeX formatting artifacts, standardizing mathematical symbols, and stripping source-specific metadata. We then perform lexical similarity matching using n-gram overlap and fuzzy matching, supplemented with semantic similarity search over sentence embeddings. Items with high similarity to public benchmark instances are manually inspected and removed if they correspond to the same mathematical problem.

We emphasize that these procedures reduce but cannot completely eliminate contamination risk. In particular, many frontier LLMs are trained on web-scale corpora, and we cannot fully determine whether they have seen natural-language versions of Olympiad, AIME, or undergraduate textbook problems. Therefore, FormalMATH should be viewed as controlling for observable benchmark overlap, especially Lean-level overlap, rather than guaranteeing absence of all pretraining exposure. For future evaluations, we recommend that model developers report whether FormalMATH source corpora or closely related natural-language problem collections were used during training.

Supervised Fine-tuning of Our Formalizer. During the development of FormalMATH, we find that mature, open-source autoformalization tools are scarce. To fill this gap, we build our own pipeline on top of two types of LLMs: coding-specialized LLMs (*e.g.*, Qwen2.5-7B-Coder (Bai et al., 2023)) and pre-trained theorem-proving LLMs (*e.g.*, Deepseek-prover-base (Xin et al., 2024)). We then generate training data by having a general-purpose LLM (*e.g.*, GPT-4 (OpenAI, 2023)) iteratively translate natural-language statements into Lean4 statements. Each candidate statement is then passed to the Lean4 compiler, and only those that are type-checked will be kept. This straightforward "compile-and-filter" strategy yields a high-quality corpus of 9,260 paired training examples, which is eventually used to finetune our own autoformalization models. Apart from our own formalizer we also use open-sourced Kimina-formalizer for multi-agents consistency.

Autoformalization. For each of the K autoformalizers (implemented by LLMs), we employ a best-of-N sampling strategy (Wang et al., 2022)

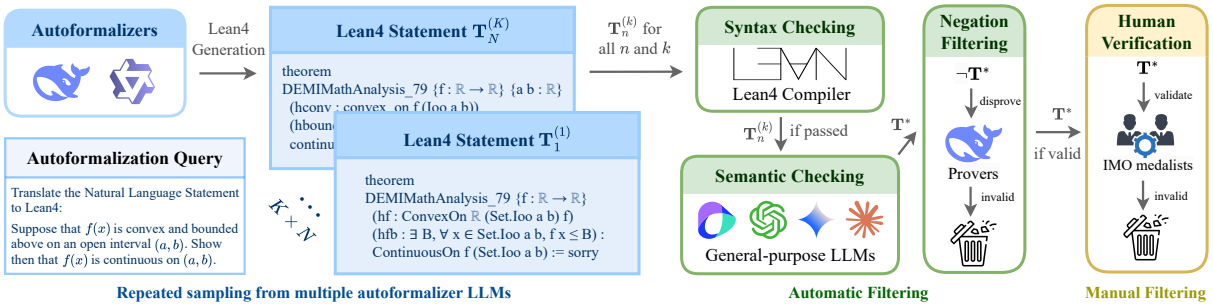


Figure 6: Formalization pipeline.

to generate N formal candidate statements $\mathbf{T}_n^{(k)}$, where $k \in \{1, \ldots, K\}$ denotes the autoformalizer index, and $n \in \{1, \ldots, N\}$ represents the candidate statement index of the k-th autoformalizer. All candidate statements $\mathbf{T}_n^{(k)}$ are first validated for syntactic correctness using the Lean4 compiler. Only syntactically valid statements are preserved for subsequent semantic verification.

Semantic Verification via LLMs. We implement a semantic verification strategy based on multiple powerful general-purpose LLMs (*e.g.*, o1-mini (Jaech et al., 2024), claude-3.5-Sonnet, GPT-5-Mini, DeepSeek-V3, etc.) with HSA to evaluate semantic alignment between natural language mathematics problems and their Lean4 formalizations. Importantly, only Lean4 statements that passed semantic verification performed by all the LLMs would be collected. This strategy is guided by the insight that translating Lean4 statements to natural language is a much easier task than the reverse process, and general-purpose LLMs excel at understanding natural language phrasings (Wu et al., 2022). Overall, this procedure filters out 60.7% of syntactically correct but semantically misaligned statements (*i.e.*, from 92.4% to 32.7%). Interestingly, we find distinct consensus patterns across problem difficulty levels – around 30% unanimous agreement rate for high school competition problems and significantly lower consensus for undergraduate-level formalizations (*e.g.*, 4.63% on HardMath).

Disproving a Statement by Proving Its Negation. Inspired by the Law of the Excluded Middle, we further filter out certain non-provable formalizations using off-the-shelf LLM-based provers (*e.g.*, DeepSeek-Prover-V1.5). For any formalized statement $\mathbf{T}_n^{(k)}$, we perform the following steps: (1) construct logical negation: construct its logical negation by applying transformation rules such as De Morgan dualization to generate $\neg\mathbf{T}_n^{(k)}$, and (2) automated proof attempts: perform automated proof attempts on $\neg\mathbf{T}_n^{(k)}$ within the formal system $\mathcal{S}$ (*i.e.*, Lean4 compiler). A successful proof of $\neg\mathbf{T}_n^{(k)}$ implies that the original statement $\mathbf{T}_n^{(k)}$ cannot hold on $\mathcal{S}$. Appendix B illustrates the Lean 4 formalization of a number-theoretic conjecture and its negation. By constructing the negation of a statement and applying an LLM-based prover for disproof, the system identifies inconsistencies through boundary case testing (*e.g.*, n = 7) and derives contradictions via systematic case analysis (*i.e.*, `interval_cases`). This strategy has filtered out a few unprovable statements, accounting for 1.6% of the total statements.

Negation-Based Disproof Protocol to Filter out Non-provable Statements

Original Lean4 Statement:

```
import Mathlib
def refBase (n : ℕ) : Prop :=
 ∀ k l, 0 < k → 0 < l → k < n → l < n →
 (k ∣ n → l ∣ n → (2 ∗ k − l ∣ n ∨ 2 ∗ l − k ∣ n))

theorem olymid_ref_base_1120 : {n ∣ l < n ∧ refBase n} = {6, 9, 15} := by
 sorry
```

Negation-based Disproof by Contradiction Construction:

```
theorem olymid_1120_negative : {n | l < n ∧ refBase n} ≠ {6, 9,
    15} := by
  simp (config := { decide := true }) [refBase]
  simp only [Set.ext_iff, Set.mem_setOf_eq, Set.mem_insert_iff,
    Set.mem_singleton_iff]
  intro h
  have h1 := h 7
  simp (config := { decide := true }) at h1
  obtain ⟨k, hk0, l, hl0, hk, hl, hkd, hld, h1, h2⟩ := h1
  interval_cases k <;> interval_cases l <;> simp_all (config :=
    {decide := true})
```

B.1 Human Expert Validation.

After automatic filtering, each remaining Lean statement was manually inspected by expert annotators. Each annotation instance contained the original natural-language problem, the candidate Lean4 theorem statement, the HSA-Verifier alignment evidence, and metadata such as source and domain. Annotators were asked to judge whether the Lean statement faithfully preserved the mathematical specification of the informal problem, rather

than whether the theorem was easy to prove.

We used a checklist-based protocol covering five aspects: (1) variables and domains/types, (2) atomic and quantified constraints, (3) structural assumptions, (4) target statement and answer encoding, and (5) extra, missing, or conflicting conditions. An item was accepted only if the formal statement preserved all specification-level information needed to define the same mathematical problem. Rejected items were required to include a free-text rationale, such as missing assumptions, incorrect quantifier scope, domain/type mismatch, expression error, or incorrect proof target.

The validation was performed by expert annotators with strong mathematical and formal verification backgrounds. Before annotation, annotators were given calibration examples and common Lean formalization error types. Items were assigned in batches, and annotator-level retention rates were monitored to detect abnormal labeling behavior. Ambiguous or rejected cases were further checked during cross-validation and adjudicated by the authors before inclusion in the final benchmark. For quality control, all rejected and uncertain items were reviewed by a second annotator. In addition, a stratified subset of accepted items was randomly sampled for secondary review. Disagreements were resolved through adjudication by the authors, where the final decision was made by checking the informal statement and Lean statement against the same five-channel checklist. For future releases, we will include a dedicated double-annotation audit split to support paired agreement analysis. Overall, the human validation stage involved 12 annotators and lasted 22 days. The final preservation rate of this stage was 72.09%, with an average annotation cost of $6.89 per statement. We also keep the rejection rationales and duplicate flags as part of the benchmark construction log to support future auditing and revision. The result of the cross-validation test for randomly sampled batches in FormalMATH can be seen in Table 10. Benchmark reliablity audit further shows 97.9% Gwet's AC1 agreement on

B.2 Benchmark Reliability Audit

To estimate the semantic reliability of FormalMATH, we conduct a post-hoc audit on 500 accepted FormalMATH statements in human review. The audited items are randomly sampled from the accepted benchmark with random seed 20260525. The goal of this audit is to evaluate whether the Lean4 theorem statements faithfully preserve the mathematical specifications of their corresponding natural-language problems.

Annotation setting. Two Lean-expert annotators independently review each item. Annotators are shown the natural-language problem, the Lean4 theorem statement, source metadata, and the reference solution when needed. They are instructed to judge only specification-level semantic faithfulness and to ignore proof difficulty, prover outputs, and the correctness of any generated proof. The annotators follow a five-channel checklist covering: variables and domains, atomic and quantified constraints, structural assumptions, target statement, and type/domain consistency.

Each item is assigned one of four labels: ACCEPT, MINOR, REJECT, or UNCERTAIN. ACCEPT indicates that the Lean statement faithfully formalizes the natural-language problem. MINOR indicates small issues that do not change the mathematical specification. REJECT indicates a semantic mismatch, and UNCERTAIN indicates that the item requires further adjudication. For computing semantic precision, we count ACCEPT and MINOR as semantically correct, and conservatively treat REJECT and UNCERTAIN as problematic before adjudication.

Adjudication and metrics. Items with disagreement between the two annotators, as well as uncertain cases, are reviewed by a third adjudicator. The adjudicator assigns a final action to each flagged item: keep, fix, or quarantine. Confirmed faulty or underspecified items are corrected or moved to a quarantine split before release.

We report semantic precision, Wilson 95% confidence intervals, exact four-way label agreement, binary semantic agreement, and Gwet's AC1 in Table 8. Importantly, the 10 / 500 confirmed mismatches are not residual errors in the final benchmark. They were audit-identified pre-release issues, and all of them were corrected before final release. Thus, the confirmed residual faulty items in the audited release sample are 0 / 500.

Metric	Result
Audited statements	500
Independent annotators	2
Adjudicator	1
P1 semantic precision	99.8%
P1 Wilson 95% CI	[98.9%, 100.0%]
P2 semantic precision	97.8%
P2 Wilson 95% CI	[96.1%, 98.8%]
Exact four-way label agreement	490 / 500 = 98.0%
Binary semantic agreement	490 / 500 = 98.0%
Gwet's AC1, four-way labels	0.9797
Gwet's AC1, binary semantic labels	0.9795
Items requiring adjudication	11 / 500 (2.2%)
Confirmed semantic mismatches	10 / 500 (2.0%)
Underspecified / uncertain items	1 / 500 (0.2%)
Release action for flagged items	corrected
Confirmed residual faulty items in audit sample	0 / 500

Table 8: Reliability audit on 500 accepted FormalMATH statements. Each item is independently reviewed by two Lean-expert annotators. ACCEPT and MINOR are counted as semantically correct, while REJECT and UNCERTAIN are treated as problematic before adjudication. Disagreements are resolved by a third adjudicator. All confirmed faulty or underspecified items are corrected or moved to a quarantine split before release.

Item	Description
Annotators	12 expert annotators
Qualification	Olympiad-level mathematical background; Lean-reading calibration
Input	NL problem, Lean statement, HSA evidence, metadata
Label space	Accept / Reject / Duplicate / Uncertain
Checklist	Variables, constraints, structure, target, type/domain consistency
Rejected items	Required free-text rationale
Quality control	Batch-level retention monitoring and cross-validation
Adjudication	Disagreements resolved by author review
Duration	22 days
Preservation rate	72.09%
Cost	$6.89 per statement

Table 9: Human validation protocol for FormalMATH.

C Prompts

C.1 Prompt for Vanilla Generation

Example: Prompt for Vanilla Generation

Complete the following Lean4 code:

```
import Mathlib
/-- If x = 2018, what is the value of the expression
    x^2 + 2 * x - x * (x + 1)? Prove that the answer is
    2018. -/
theorem omni\_theorem\_2669
(x : \mathbb{Z}) (hx : x = 2018) : x^2 + 2 * x - x * (x + 1) = 2018
    := by
```

Example: Prompt for CoT Generation

Complete the following Lean4 code with explanatory comments preceding each line of code:

```
import Mathlib
/-- If x = 2018, what is the value of the expression
    x^2 + 2 * x - x * (x + 1)? Prove that the answer is
    2018. -/
theorem omni\_theorem\_2669
(x : \mathbb{Z}) (hx : x = 2018) : x^2 + 2 * x - x * (x + 1) = 2018
    := by
```

Example: Prompt for NL-augmented CoT

Complete the following Lean4 code with explanatory comments preceding each line of code:

```
import Mathlib
open Finset
theorem omni\_theorem\_4199 :
\(\exists n \in \mathbb{N}, \{s : \text{Finset} \mathbb{N}
    \mid \text{s.card} = 2017 \land \sum_{i \in s} i^2 = n
    \}. \text{ncard} \geq 2017 \text{:= by}\)
/-To determine if there exists a number \( n \) that can be
    expressed as the sum of 2017 perfect squares in at least
    2017 distinct ways, we consider the properties and
    combinations of perfect squares.
\#\#\# Step 1: Understanding the Problem
The problem asks us to express a number \( n \) as the sum of 2017
    perfect squares, \( n = a_1^2 + a_2^2 + \cdots +
    a_{2017}^2 \), where \( a_i \) are integers. Moreover, this
    can be done in at least 2017 different ways, meaning there
    are at least 2017 distinct sets of such integers.
\#\#\# Step 2: Exploring Perfect Squares
Perfect squares are non-negative numbers of the form \( k^2 \),
    where \( k \) is an integer. To construct different sums, we
    need to evaluate how the combinations of these squares can
    vary and still yield distinct sums that equate to the same \( n
    \).
\#\#\#  Step 3: Existence of Solutions
1. **Many Small Squares**: By choosing different arrangements
    of small perfect squares (like 0, 1, 4, 9, etc.), we can vary
    them freely since they don't drastically alter the cumulative
    sum quickly. For instance, using 0 is trivial as it adds
    nothing to sums; including or excluding it in varying
    positions introduces variety.
2. **Adjusting a Larger Value**: Consider including a larger
    square, say \((k+1)^2\), and adjusting the rest of the terms
```

Group	ID	Batch 1	Batch 2	Batch 3	Overall
A	A1	63.00	81.33	76.57	75.82
A	A2	76.53	79.00	82.55	81.62
A	A3	77.00	82.00	80.00	80.14
A	A4	59.00	62.00	53.52	55.76
A	A5	73.00	90.00	82.00	81.75
A	A6	85.00	–	77.26	79.20
A	A7	–	–	72.86	72.86
A mean	–	72.25	78.87	74.97	75.31
A std.	–	9.65	10.30	10.04	9.20

Table 10: Batch-level acceptance rates (%) in human cross-validation for randomly sampled batches (batch size = 200). Annotator identities are anonymized. The table reports the percentage of candidate Lean statements accepted by each annotator in each assigned batch. These statistics were used for batch-level quality monitoring and abnormal labeling detection

Error Category	Percentage (%)
Condition Error	83.3
Expression Error (Lean Syntax)	9.6
Definition Error (No Mathematical Meaning)	3.3
Domain Error	1.7
Propositional Logic Error	0.9
Lack of Geometric Background	0.6
Condition Redundancy	0.5
Algebraic Expression Error	0.2

Table 11: Error types of filtered data. (%)

accordingly. This diversity of combinations even with fixed values of \(a_i = 0 \) (i.e., not all contributing to sum) provides additional distinct setups.
\#\#\# Step 4: Conclusion
Given the vast number of combinations possible with 2017 variables, it is feasible to achieve at least 2017 distinct sums since:
– Choosing different subsets of minimal contributions (e.g., many zeros and small numbers) can still lead to varying sums.
– Incremental adjustments in a few selections using larger squares or varied middle–range integers allow differential assembly leading to the target sum.
Thus, there is indeed a number \(n \) that can be expressed as the sum of 2017 perfect squares in at least 2017 distinct ways.
Hence, the answer is:
\(\boxed{\text{Yes}}\)–/

C.2 Domain Classification

Prompt for Domain Classification

```
# CONTEXT #
I am a teacher, and I have some high-level math problems.
I want to categorize the domain of these math problems.

# OBJECTIVE #
A. Summarize the math problem in a brief sentence, describing the concepts involved in the math problem.
B. Categorize the math problem into specific mathematical domains. Please provide a classification chain, for example: Mathematics -> Applied Mathematics -> Probability -> Combinations. The following is a basic classification framework in the field of mathematics.
<math domains>
Mathematics
|
|- Applied Mathematics
| |- Math Word Problems
| `- Statistics
| |- Mathematical Statistics
| `- Probability
| `- Counting Methods
| |- Permutations
| `- Combinations
|
|- Algebra
| |- Prealgebra
| | |- Integers
| | |- Fractions
| | |- Decimals
| | `- Simple Equations
| |- Algebra
| | |- Algebraic Expressions
| | |- Equations and Inequalities
| | |- Factoring
| | `- Polynomial Operations
| |- Intermediate Algebra
| | |- Quadratic Functions
| | |- Exponential Functions
| | |- Logarithmic Functions
| | `- Complex Numbers
| |- Linear Algebra
| | |- Vectors
| | |- Matrices
| | |- Determinants
| | `- Linear Transformations
| `- Abstract Algebra
| |- Group Theory
| |- Ring Theory
| `- Field Theory
|
|- Geometry
| |- Plane Geometry
| | |- Polygons
| | |- Angles
| | |- Area
| | |- Triangulations
| | `- Perimeter
| |- Solid Geometry
| | |- 3D Shapes
| | |- Volume
| | `- Surface Area
| |- Differential Geometry
| | |- Curvature
| | |- Manifolds
| | `- Geodesics
| `- Non-Euclidean Geometry
```

```
| |- Spherical Geometry
| `- Hyperbolic Geometry
|
|- Number Theory
| |- Prime Numbers
| |- Factorization
| |- Congruences
| |- Greatest Common Divisors (GCD)
| `- Least Common Multiples (LCM)
|
|- Precalculus
| |- Functions
| |- Limits
| `- Trigonometric Functions
|
|- Calculus
| |- Differential Calculus
| | |- Derivatives
| | |- Applications of Derivatives
| | `- Related Rates
| `- Integral Calculus
| |- Integrals
| |- Applications of Integrals
| `- Techniques of Integration
| |- Single-variable
| `- Multi-variable
|
|- Differential Equations
| |- Ordinary Differential Equations (ODEs)
| `- Partial Differential Equations (PDEs)
|
`- Discrete Mathematics
|- Graph Theory
|- Combinatorics
|- Logic
`- Algorithms
</math domains>

# STYLE #
Data report.

# TONE #
Professional, scientific.

# AUDIENCE #
Students. Enable them to better understand the domain and difficulty of the math problems.

# RESPONSE: MARKDOWN REPORT # ## Summarization [Summarize the math problem in a brief paragraph.] ## Math domains [Categorize the math problem into specific mathematical domains, including major domains and subdomains.]`
# ATTENTION # - The math problem can be categorized into multiple domains, but no more than three. Separate the classification chains with semicolons(;).
- Your classification MUST fall under one of the aforementioned subfields; if it really does not fit, please add "Other" to the corresponding branch. For example: Mathematics -> Algebra -> Intermediate Algebra -> Other. Only the LAST NODE is allowed to be "Other"; the preceding nodes must strictly conform to the existing framework.
- The math domain must conform to a format of classification chain, like "Mathematics -> Applied Mathematics -> Probability -> Combinations".
- Add "=== report over ===" at the end of the report.

<example math problem>

[Question]:
If $\frac{1}{9}+\frac{1}{18}=\frac{1}{square}$, what is the number that replaces the $square$ to make the equation true?
[Solution]:
We simplify the left side and express it as a fraction with numerator 1: $\frac{1}{9}+\frac{1}{18}=\frac{2}{18}+\frac{1}{18}=\frac{3}{18}=\frac{1}{6}$. Therefore, the number that replaces the $square$ is 6.
[Source]: 2010_Pascal:

</example math problem>
## Summarization The problem requires finding a value that makes the equation $\frac{1}{9}+\frac{1}{18}=\frac{1}{square}$. This involves adding two fractions and determining the equivalent fraction.
## Math domains Mathematics -> Algebra -> Prealgebra -> Fractions;

=== report over ===

</example math problem>
[Question]:
Let $\mathcal{P}$ be a convex polygon with $n$ sides, $n \geq 3$. Any set of $n-3$ diagonals of $\mathcal{P}$ that do not intersect in the interior of the polygon determine a triangulation of $\mathcal{P}$ into $n-2$ triangles. If $\mathcal{P}$ is regular and there is a triangulation of $\mathcal{P}$ consisting of only isosceles triangles, find all the possible values of $n$.
[Solution]:
We label the vertices of $\mathcal{P}$ as $P_0, P_1, P_2, \ldots, P_n$. Consider a diagonal $d = \overline{P_a P_{a+k}}$, $k \leq n/2$ in the triangulation. We show that $k$ must have the form $2^m$ for some nonnegative integer $m$. This diagonal partitions $\mathcal{P}$ into two regions $\mathcal{Q}$, $\mathcal{R}$, and is the side of an isosceles triangle in both regions. Without loss of generality suppose the area of $Q$ is less than the area of $R$ (so the center of $P$ does not lie in the interior of $Q$); it follows that the lengths of the edges and diagonals in $Q$ are all smaller than $d$. Thus $d$ must the be the base of the isosceles triangle in $Q$, from which it follows that the isosceles triangle is $\triangle P_a P_{a+k/2} P_{a+k}$, and so $2|k$. Repeating this process on the legs of isosceles triangle $(\overline{P_a P_{a+k/2}}, \overline{P_{a+k} P_{a+k/2}})$, it follows that $k = 2^m$ for some positive integer $m$ (if we allow degeneracy, then we can also let $m = 0$). Now take the isosceles triangle $P_x P_y P_z$, $0 \leq x < y < z < n$ in the triangulation that contains the center of $\mathcal{P}$ in its interior; if a diagonal passes through the center, select either of the isosceles triangles with that diagonal as an edge. Without loss of generality, suppose $P_x P_y = P_y P_z$. From our previous result, it follows that there are $2^a$ edges of $P$ on the minor arcs of $P_x P_y$, $P_y P_z$ and $2^b$ edges of $P$ on the minor arc of $P_z P_x$, for positive integers $a, b$. Therefore, we can write

$$n = 2 \cdot 2^a + 2^b = 2^{a+1} + 2^b,$$

so $n$ must be the sum of two powers of 2. We now claim that this condition is sufficient. Suppose without loss of generality that $a+1 \geq b$; then we rewrite this as

$$n = 2^b(2^{a-b+1} + 1).$$

Lemma 1: All regular polygons with $n = 2^k + 1$ or $n = 4$ have triangulations that meet the conditions. By induction, it follows that we can cover all the desired $n$. For $n = 3, 4$, this is trivial. For $k > 1$, we construct the diagonals of equal length $\overline{P_0 P_{2^{k-1}}}$ and $\overline{P_{2^{k-1}+1} P_0}$. This partitions $\mathcal{P}$ into 3 regions: an isosceles $\triangle P_0 P_{2^{k-1}} P_{2^{k-1}+1}$, and two other regions. For these two regions, we can recursively construct the isosceles triangles defined above in the second paragraph. It follows that we have constructed $2(2^{k-1}-1)+(1) = 2^k - 1 = n-2$ isosceles triangles with non-intersecting diagonals, as desired.
Lemma 2: If a regular polygon with $n$ sides has a working triangulation, then the regular polygon with $2n$ sides also has a triangulation that meets the conditions. We construct the diagonals $\overline{P_0 P_2}$, $\overline{P_2 P_4}$, $\ldots \overline{P_{2n-2} P_0}$. This partitions $\mathcal{P}$ into $n$ isosceles triangles of the form $\triangle P_{2k} P_{2k+1} P_{2k+2}$, as well as a central regular polygon with $n$ sides. However, we know that there exists a triangulation for the $n$-sided polygon that yields $n-2$ isosceles triangles. Thus, we have created $(n)+(n-2) = 2n-2$ isosceles triangles with non-intersecting diagonals, as desired. In summary, the answer is all $n$ that can be written in the form $2^{a+1} + 2^b$, $a, b \geq 0$. Alternatively, this condition can be expressed as either $n = 2^k$, $k \geq 2$ (this is the case when $a+1 = b$) or $n$ is the sum of two distinct powers of 2, where $1 = 2^0$ is considered a power of 2.
[Source]:
USAMO 2008
</example math problem>

## Summarization
The problem asks for the possible values of $n$ for a regular $n$-sided polygon that can be completely triangulated into isosceles triangles using non-intersecting diagonals. The solution involves
```

analyzing the properties of the diagonals forming isosceles triangles and deducing that n can be expressed in terms of powers of 2.
Math domains
Mathematics -> Geometry -> Plane Geometry -> Polygons;

=== report over ===

C.3 Difficulty Classification

Prompt for Difficulty Classification

You are an expert mathematical problem evaluator. Your task is to assess the intrinsic difficulty of the given informal problem.
I am working with natural language statements of advanced mathematics problems, which are intended to be later formalized in Lean. Before that, we aim to assess the **intrinsic difficulty** of the problem in its current informal (natural language) form.
#OBJECTIVE
Assign a **difficulty score** to the problem, on a scale from 0 to 10.
Your rating should reflect the mathematical reasoning required to solve the problem - including the level of abstraction, creativity, number of steps, and familiarity with advanced techniques.
#DIFFICULTY REFERENCE
##Examples for difficulty levels
For reference, here are problems from each of the difficulty levels 1-10:
1: How many integer values of x satisfy $|x| < 3\pi$? (2021 Spring AMC 10B, Problem 1)
1.5: A number is called flippy if its digits alternate between two distinct digits. For example, 2020 and 37373 are flippy, but 3883 and 123123 are not. How many five-digit flippy numbers are divisible by 15? (2020 AMC 8, Problem 19)
2: A fair 6-sided die is repeatedly rolled until an odd number appears. What is the probability that every even number appears at least once before the first occurrence of an odd number? (2021 Spring AMC 10B, Problem 18)
2.5: A, B, C are three piles of rocks. The mean weight of the rocks in A is 40 pounds, the mean weight of the rocks in B is 50 pounds, the mean weight of the rocks in the combined piles A and B is 43 pounds, and the mean weight of the rocks in the combined piles A and C is 44 pounds. What is the greatest possible integer value for the mean in pounds of the rocks in the combined piles B and C? (2013 AMC 12A, Problem 16)
3: Triangle $\triangle ABC$ with $AB = 50$ and $AC = 10$ has area 120. Let D be the midpoint of AB, and let E be the midpoint of AC. The angle bisector of $\angle BAC$ intersects DE and BC at F and G, respectively. What is the area of quadrilateral FDBG? (2018 AMC 10A, Problem 24)
3.5: Find the number of integer values of k in the closed interval $[-500, 500]$ for which the equation $\log(kx) = 2\log(x+2)$ has exactly one real solution. (2017 AIME II, Problem 7)
4: Define a sequence recursively by $x_0 = 5$ and

$$x_{n+1} = \frac{x_n^2 + 5x_n + 4}{x_n + 6}$$

for all nonnegative integers n. Let m be the least positive integer such that

$$x_m \le 4 + \frac{1}{2^{20}}.$$

In which of the following intervals does m lie? (A) [9, 26] (B) [27, 80] (C) [81, 242] (D) [243, 728] (E) [729, ∞) (2019 AMC 10B, Problem 24 and 2019 AMC 12B, Problem 22)
4.5: Find, with proof, all positive integers n for which $2^n + 12^n + 2011^n$ is a perfect square. (USAJMO 2011/1)
5: Find all triples (a, b, c) of real numbers such that the following system holds:

$$a + b + c = \frac{1}{a} + \frac{1}{b} + \frac{1}{c},$$

$$a^2 + b^2 + c^2 = \frac{1}{a^2} + \frac{1}{b^2} + \frac{1}{c^2}.$$

(JBMO 2020/1)
5.5: Triangle ABC has $\angle BAC = 60^\circ$, $\angle CBA \le 90^\circ$, $BC = 1$, and $AC \ge AB$. Let H, I, and O be the orthocenter, incenter, and circumcenter of $\triangle ABC$, respectively. Assume that the area of pentagon BCOIH is the maximum possible. What is $\angle CBA$? (2011 AMC 12A, Problem 25)
6: Let $\triangle ABC$ be an acute triangle with circumcircle ω, and let H be the intersection of the altitudes of $\triangle ABC$. Suppose the tangent to the circumcircle of $\triangle HBC$ at H intersects ω at points X and Y with $HA = 3$, $HX = 2$, and $HY = 6$. The area of $\triangle ABC$ can be written in the form $m\sqrt{n}$, where m and n are positive integers, and n is not divisible by the square of any prime. Find $m + n$. (2020 AIME I, Problem 15)
6.5: Rectangles BCC_1B_2, CAA_1C_2, and ABB_1A_2 are erected outside an acute triangle ABC. Suppose that $\angle BC_1C + \angle CA_1A + \angle AB_1B = 180^\circ$. Prove that lines B_1C_2, C_1A_2, and A_1B_2 are concurrent. (USAMO 2021/1, USAJMO 2021/2)
7: We say that a finite set S in the plane is balanced if, for any two different points A, B in S, there is a point C in S such that $AC = BC$. We say that S is centre-free if for any three points A, B, C in S, there is no point P in S such that $PA = PB = PC$. Show that for all integers $n \ge 3$, there exists a balanced set consisting of n points. Determine all integers $n \ge 3$ for which there exists a balanced centre-free set consisting of n points. (IMO 2015/1)
7.5: Let $\mathbb{Z}$ be the set of integers. Find all functions $f : \mathbb{Z} \to \mathbb{Z}$ such that

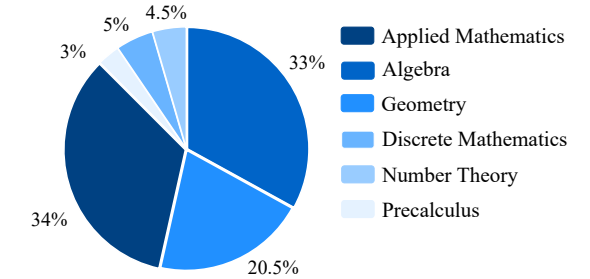
$$xf(2f(y) - x) + y^2 f(2x - f(y)) = f(x)^2 + f(yf(y))$$

for all $x, y \in \mathbb{Z}$ with $x \ne 0$. (USAMO 2014/2)
8: For each positive integer n, the Bank of Cape Town issues coins of denomination $1/n$. Given a finite collection of such coins (of not necessarily different denominations) with total value at most $99 + 1/2$, prove that it is possible to split this collection into 100 or fewer groups, such that each group has total value at most 1. (IMO 2014/5)
8.5: Let I be the incentre of acute triangle ABC with $AB \ne AC$. The incircle ω of ABC is tangent to sides BC, CA, and AB at D, E, and F, respectively. The line through D perpendicular to EF meets ω at R. Line AR meets ω again at P. The circumcircles of triangle PCE and PBF meet again at Q. Prove that lines DI and PQ meet on the line through A perpendicular to AI. (IMO 2019/6)
9: Let k be a positive integer and let S be a finite set of odd prime numbers. Prove that there is at most one way (up to rotation and reflection) to place the elements of S around the circle such that the product of any two neighbors is of the form $x^2 + x + k$ for some positive integer x. (IMO 2022/3)
9.5: An anti-Pascal triangle is an equilateral triangular array of numbers such that, except for the numbers in the bottom row, each number is the absolute value of the difference of the two numbers immediately below it. For example, the following is an anti-Pascal triangle with four rows which contains every integer from 1 to 10.

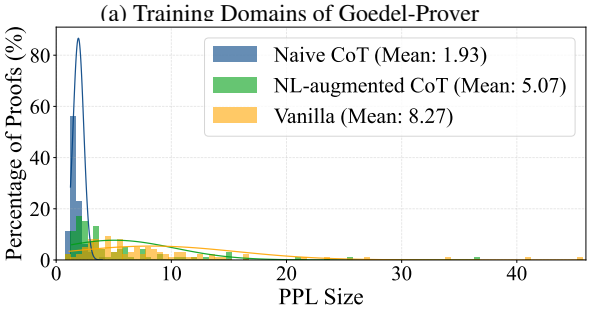
```
        4
      2   6
    5   7   1
  8   3   10   9
```

Does there exist an anti-Pascal triangle with 2018 rows which contains every integer from 1 to $1 + 2 + 3 + \cdots + 2018$? (IMO 2018/3)
10: Prove that there exists a positive constant c such that the following statement is true: Consider an integer $n > 1$, and a set S of n points in the plane such that the distance between any two different points in S is at least 1. It follows that there is a line ℓ separating S such that the distance from any point of S to ℓ is at least $cn^{-1/3}$.
#OBJECTIVE

1. Summarize the math problem in a brief sentence, describing the concepts involved in the math problem.



(a) Training Domains of Goedel-Prover



(b) Perplexity of DeepSeek-V1.5-SFT

Figure 7: (a) The mathematical domain distribution of Goedel-Prover's training dataset. (b) The perplexity distribution of Deepseek-V1.5-SFT across various proof generation modes.

```
    2. Based on the source of the given problem, as well
       as the difficulty of the problems referenced in
       these materials and the solution to the current
       problem, please provide an overall difficulty
       score for the current problem. The score should be
       a number between 1 and 10, with increments of 0.5,
       and should align perfectly with the materials.

#STYLE#
Data report.
#TONE#
Professional, scientific.
#AUDIENCE#
Students.  Enable them to better understand the
difficulty of the math problems.
#RESPONSE: MARKDOWN REPORT# ##Summarization
[Summarize the math problem in a brief paragraph.]
##Difficulty
[Rate the difficulty of the math problem and give the
reason.]
#ATTENTION#

    • Add "=== report over ===" at the end of the report.

#INPUT# Below is the original natural language math
problem statement:
<statement>{statement}</statement>
#OUTPUT FORMAT#
You must respond with a JSON object:

{
  "Difficulty": float (between 0 and 10),
 "Rationale": "Explain your score in 1-3 sentences. Mention struct
  elements or comparison to benchmark problems."
}
```

C.4 Prompt for Verifier

Example: Prompt for Normal LLM-based Verifier in CriticLean.

```
Role: Lean & Formal Verification Expert

Input:
- Mathematical_Text: A math problem and its answer (no
proof).
- Lean4Code: A Lean 4 theorem statement formalizing the
problem. Proof is intentionally omitted (e.g., sorry).

Goal:
Determine if the Lean theorem statement is an exact and
faithful formalization of the mathematical problem.
**Do not evaluate or consider the answer or the proof.
Your sole task is to verify the correctness of the
formalization.**

Evaluation Stages (All required):

1. Math Assertion Analysis
Identify all structurally and semantically relevant
components of the mathematical problem, including
variables, types, quantifiers, constraints, logic
structure, conclusion, and so on. The analysis should
be based on the actual content of the text.

2. Lean Statement Analysis (ignore proof part)
Extract all structurally and semantically relevant
components from the Lean statement, including variables,
types, conditions, quantifiers, constraints, the final
claim, and so on.  The analysis should reflect the
actual content present in the Lean code.

3. Comparative Verification
Check for exact correspondence between the math and
Lean statements; you may refer to aspects like:
- Semantic alignment, logic structure, and quantifier
correctness.
- Preservation of constraints and boundary assumptions.
- Accurate typing and use of variables.
- Syntactic validity and proper Lean usage (free from
errors).
- Use of symbols and constructs without semantic drift.
- No missing elements, no unjustified additions, and no
automatic corrections or completions.

4.   Final Judgement Based solely on the above
analysis, judge whether the Lean statement is a correct
and exact formalization of the mathematical problem.

5. Accuracy Confirmation
If correct: clearly confirm why all elements match.
If incorrect: list all mismatches and explain how each
one affects correctness.

Note:  While the analysis may be broad and open
to interpreting all relevant features, the final
judgment must be based only on what is explicitly and
formally expressed in the Lean statement.
**Do not consider or assess any part of the proof. Your
judgment should be entirely about the accuracy of the
statement formalization.**

Output Format:
Return exactly one JSON object:
"'json

{
  "reasons": "Your detailed CoT analysis:
1. Math Assertion Analysis: [...]
2. Lean Statement Analysis (Proof Ignored): [...]
3. Comparative Verification: [...]
4. Conclusion: [...]
5. Accuracy Confirmation: [match confirmation or list of discrepancies...]",
  "is_assistant_correct": "[Correct/Incorrect]"
}
"'

Input Data:
- Start of Mathematical_Text -
{mathematical_statement}
- End of Mathematical_Text -

- Start of Lean4Code -
{autoformalization_placeholder}
- End of Lean4Code -
```

Example: Prompt for HSA-Verifier.

```
Role: Lean & Formal Verification Expert

Input:
- Mathematical_Text: A math problem and its answer (no
proof).
- Lean4Code: A Lean 4 theorem statement formalizing the
```

problem. Proof is intentionally omitted (e.g., sorry).

Goal:
Determine if the Lean theorem statement is an exact and faithful formalization of the mathematical problem.
Do not evaluate or consider the answer or the proof. Your sole task is to verify the correctness of the formalization.

Evaluation Stages (All required):

1. Extract and compare all conditions from informal and formal statements
Before doing semantic judgement, first extract structured evidence from Mathematical_Text, using Lean4Code only as disambiguation context (domain/scope/symbol role), not as ground truth override.
Extraction objective: Extract every condition that materially changes admissible solution space.
A condition is HSA-relevant if removing it enlarges/shrinks/changes valid interpretations.
You must extract for informal and formal statements and comparing their consistency:
- variables and domains/types
- atomic constraints (domain/bound/equality/inequality/cardinality/distinctness divisibility/parity/relational/existence/uniqueness)
- quantified constraints (quantifier type, bound variables, range, body, scope)
- structural assumptions (graph/geometry/algebra/combinatorics/global properties)
- target statement (separate from assumptions)
- implicit_or_derived_requirements (only if semantically necessary and clearly justified)
- ambiguities (if wording is unclear; do not silently guess)

Hard rules: - Do NOT treat target claim as assumption. - Do NOT invent unsupported assumptions. - Do NOT flatten quantified conditions into vague summaries. - If uncertain, keep conservative normalization and record ambiguity.
think in JSON schema:

```
  {
    "variables": [...],
    "atomic_constraints": [...],
    "quantified_constraints": [...],
    "structural_assumptions": [...],
    "target_statement": {...},
    "implicit_or_derived_requirements": [...],
    "ambiguities": [...]
  }
```

Then use this extracted JSON as mandatory evidence in Stages 2-6.
2. Math Assertion Analysis Identify all structurally and semantically relevant components of the mathematical problem, including variables, types, quantifiers, constraints, logic structure, conclusion, and so on. The analysis should be based on the actual content of the text. 3. Lean Statement Analysis (ignore proof part) Extract all structurally and semantically relevant components from the Lean statement, including variables, types, conditions, quantifiers, constraints, the final claim, and so on. The analysis should reflect the actual content present in the Lean code. 4. Comparative Verification Check for exact correspondence between the math and Lean statements; you may refer to aspects like: - Semantic alignment, logic structure, and quantifier correctness. - Preservation of constraints and boundary assumptions. - Accurate typing and use of variables. - Syntactic validity and proper Lean usage (free from errors). - Use of symbols and constructs without semantic drift. - No missing elements, no unjustified additions, and no automatic corrections or completions. 5. Final Judgement Based solely on the above analysis, judge whether the Lean statement is a correct and exact formalization of the mathematical problem. 6. Accuracy Confirmation If correct: clearly confirm why all elements match. If incorrect: list all mismatches and explain how each one affects correctness. Note: While the analysis may be broad and open to interpreting all relevant features, the final judgment must be based only on what is explicitly and formally expressed in the Lean statement.
Do not consider or assess any part of the proof. Your judgment should be entirely about the accuracy of the statement formalization.

Output Format:
Return exactly one JSON object:
“‘**json**

```
{
  "reasons": "Your detailed CoT analysis:
1. Math Assertion Analysis: [...]
2. Lean Statement Analysis (Proof Ignored): [...]
3. Comparative Verification: [...]
4. Conclusion: [...]
5. Accuracy Confirmation: [match confirmation or list of discrepancies...]",
  "is_assistant_correct": "[Correct/Incorrect]"
}
“‘
```

Input Data:
- Start of Mathematical_Text -
{mathematical_statement}
- End of Mathematical_Text -

- Start of Lean4Code -
{autoformalization_placeholder}
- End of Lean4Code -

D Common Errors in Proving FormalMATH

E Typical Errors in Statement Autoformalization

E.1 Errors in Definition

Definition Errors mainly occur in the failure to semantically correspond the mathematical entities in autoformalized statements one-to-one with those in the refined statement. For instance, in a geometry problem, if the defined entities do not pertain to geometric concepts such as points, lines, and angles, it indicates a lack of domain relevance.

Mismatched Variable type is also a common issue. This type of error can lead to failure when applying tactics that are specific to a particular variable type.

Furthermore, the defined variables may lack actual mathematical significance so that the statement cannot pass the compiler.

Lean4 Statement	Prover Output	Explanation

`theorem omni_theorem_4000 (f : ℝ → ℝ) :`
`(∀ x y, (x + y ^ 2) * f (y * f x) = x * y * f (y ^ 2 + f x)) ↔`
`∀ x, f x = x ∨ f x = 0 ∨ ∃ a, a ≤ −1 ∨ a > 0 ∧ f x = if x ≠ −a ^ 2 then 0 else a := by`
	`-- Proof Before`
	`-- If f(x) != 0, we need to show that there exists an a such that the conditions hold.`
	`right`
	`right`
	`use 0`
	`-- We choose a = 0 to satisfy the conditions.`
	`simp`
	`--Proof After`
		Improper use of Automation Tactics. The prover selects the witness $a = 0$ without checking the original side-condition and then calls `simp` to close off the goal. This bypasses the essential case-by-case analysis of the core functional equation, resulting in an invalid proof that never constructs the required nonzero witness when needed.

`theorem DEMIMathAnalysis_50 :`
`Tendsto (λ n => sqrt n * ∫ x in univ, 1 / (1 + x ^ 2) ^ n) atTop (N (sqrt π)) := by`
	`rw [show sqrt = fun x => sqrt x by funext; rfl]`
	`apply tendsto_atTop_mono (fun n => sqrt_le_sqrt <|`
	`integral_mono_on _ _ <| fun x _ => by`
	`gcongr`
	`all_goals norm_num)`
	`refine'`
	`tendsto_atTop_of_eventually_const (by simp)`
	`all_goals`
	`filter_upwards with _`
	`<;> simp`
		Incomplete Proof. The proof invokes `tendsto_atTop_mono` without establishing any of the analytical prerequisites. The final `simp` steps trivially handle eventual constancy but leave the main asymptotic argument unresolved.

`theorem aime_all_2005_II_1 (n : ℕ) (hn : n ≥ 6) (h : {s : Finset (Fin n) | s.card = 6}.ncard = 6 * {s : Finset (Fin n) | s.card = 3}.ncard) :`
`n = 13 := by`
	`norm_num [Nat.choose] at h`
	`revert n hn h`
	`rintro n hn h`
	`ring_nf at h`
	`nlinarith`
		Redundant Hypothesis Introduction. After `norm_num` already rewrites the binomial coefficients, the `revert ; rintro` sequence merely reintroduces n, hn, and h in the same form, adding no new information and cluttering the context.

`theorem algebra_528739 (a b c : ℝ) (ha : 0 < a) (hb : 0 < b) (hc : 0 < c) :`
`1 / (a^3 + b^3 + a * b * c) + 1 / (b^3 + c^3 + a * b * c) +`
`1 / (c^3 + a^3 + a * b * c) ≤ 1 / (a * b * c) := by`
	`-- We start by simplifying the left-hand side using the fact that each term is non-negative.`
	`have $h_0$ : 0 < a * b * c := by positivity`
	`...`
	`-- We then apply the inequality to each term.`
	`rw [div_le_div_iff] <;>`
	`nlinarith , [sq_nonneg (a − b), sq_nonneg (b − c), sq_nonneg (c − a)]`
	`...`
		Inadequate Handling of Inequalities. The solver attempts to apply `nlinarith` after a single `div_le_div_iff`, but the cyclic, high-degree fractional structure exceeds its linear-and-quadratic reasoning scope.

Table 12: Examples of common Lean4 error patterns in LLM-based provers.

Example E.1: Errors in Definition: Lack of Specific Mathematical Background (Geometry)

omni_theorem_2599

Points A, B, C, D lie on a circle in that order such that $\frac{AB}{BC} = \frac{DA}{CD}$. If $AC = 3$ and $BD = BC = 4$, find AD.

Prove that the answer is: $\frac{3}{2}$

Autoformalization:

```
import Mathlib

open EuclideanGeometry Real
open scoped BigOperators

theorem omni_theorem_2599
 {A B C D : EuclideanSpace ℝ (Fin 2)}
 (h₀ : ∃ O, dist A O = dist B O ∧ dist B O = dist C O ∧ dist C O
      = dist D O)
 (h₁ : dist A B / dist B C = dist D A / dist C D)
 (h₂ : dist A C = 3)
 (h₃ : dist B D = 4)
 (h₄ : dist B C = 4) :
 dist A D = 3 / 2 := by sorry
```

The model's formalization is deficient because it only specifies the lengths of line segments AB, BC, CD, and DA, without explicitly stating that points A, B, C, and D are in that order along circle O. This omission is critical as only segment lengths is insufficient to uniquely define a quadrilateral inscribed in a circle. Different orderings of these points on the circle, even with the same segment lengths.

Example E.2: Errors in Definition: Mismatched variable Type

omni_theorem_66

Determine whether or not there exist two distinct sets A and B, each consisting of at most 2011^2 positive integers, such that for every x with $0 < x < 1$, the following inequality holds:
$\left|\sum_{a\in A} x^a - \sum_{b\in B} x^b\right| < (1-x)^{2011}$.
Prove that the answer is Yes.

Autoformalization:

```
import Mathlib

theorem omni_theorem_66 :
  ∃ A B : Finset ℕ,
   A ≠ B ∧
   A.card ≤ 2011^2 ∧
   B.card ≤ 2011^2 ∧
   (∀ x ∈ Set.Ioo 0 1, |∑ a ∈ A, (x : ℝ)^a − ∑ b ∈ B, (x : ℝ
     )^b| < (1 − x)^2011) := by sorry
```

While positive integer mappings and positive integers m and n were required, the autoformalization mistakenly used natural numbers instead.

Example E.3: Errors in Definition: Meaningless Definition in Mathematical Context

omni_theorem_1024

For a positive integer n, denote by $\tau(n)$ the number of positive integer divisors of n, and denote by $\phi(n)$ the number of positive integers that are less than or equal to n and relatively prime to n. Call a positive integer n good if $\varphi(n) + 4\tau(n) = n$. For example, the number 44 is good because $\varphi(44) + 4\tau(44) = 44$. Find the sum of all good positive integers n.
Prove that the answer is: 172

Autoformalization:

```
import Mathlib

open Finset

def isGood (n : ℕ) := n.totient + 4 ∗ n.divisors.card = n

theorem omni_theorem_1024 :
  ∑ n ∈ {n | 0 < n ∧ isGood n}, n = 172 := by sorry
```

Euler's totient function, denoted as $\phi(n)$ or in Lean4 as `ntotient`, counts the number of positive integers less than or equal to n that are relatively prime to n. For $n = 0$, the function `n.totient` is not defined, which results in isGood n being undefined.

E.2 Errors in Expressions

This type of errors include incorrect arithmetic or logical operations, improper use of variables, or incorrect application of mathematical or logical rules.

Example E.4: Errors in Proposition Construction (Wrong Position of Existential Qualifier)

omni_theorem_4

Let p be a prime number. Prove the following theorem of Euler: the equation $p = x^2 + 3y^2$ has a solution with $x, y \in \mathbb{Z}$ if and only if $p = 3$ or $p \equiv 1 \pmod 3$. (You may use the fact that the ring of integers of $\mathbb{Q}(\sqrt{-3})$ is a principal ideal domain.)
Prove that the answer is: The equation $p = x^2 + 3y^2$ has a solution with $x, y \in \mathbb{Z}$ if and only if p = 3 or p \equiv 1 \pmod{3}

Autoformalization:

```
import Mathlib

open Real
open scoped BigOperators

theorem omni_theorem_401 {p : ℕ} (hp : Nat.Prime p) :
  ∃ x y : ℤ, p = x^2 + 3∗y^2 ↔ p = 3 ∨ p % 3 = 1 := by sorry
```

In the Lean4 code, the existential quantifier is misplaced. $\exists x, y \in \mathbb{Z}$ should quantify the entire left-hand side of the biconditional. Instead of asserting that the existence of a solution (x, y) is equivalent to the condition $p = 3 \vee p \equiv 1 \pmod 3$, the incorrect code unintentionally suggests the existence of x and y is linked to the entire biconditional statement being true. the existential quantifier should be correctly scoped only to the left-hand side of the biconditional, as in the corrected form:

$$(\exists x, y \in \mathbb{Z}, p = x^2 + 3y^2) \leftrightarrow (p = 3 \vee p \equiv 1 \pmod 3),$$

which ensures that the quantifier only applies to the equation $p = x^2 + 3y^2$.

Example E.5: Errors in Expression: Misleading Scope of the Universal Quantifier

omni_theorem_219

Find, with proof, all nonconstant polynomials $P(x)$ with real coefficients such that, for all nonzero real numbers z with $P(z) \neq 0$ and $P\left(\frac{1}{z}\right) \neq 0$, we have

\frac{1}{P(z)}+\frac{1}{P\left(\frac{1}{z}\right)}=z+\frac{1}{z}

Prove that the answer is:

P(x) = \frac{x(x^{4k+2}+1)}{x^{2}+1} \quad \text{or} \quad P(x) = \frac{x(1−x^{4k})}{x^{2}+1}

Autoformalization:

```
import Mathlib

open Fintype Set Real Ideal Polynomial
open scoped BigOperators

theorem omni_theorem_219 {P : Polynomial ℝ} (hP : P.degree
    > 0) :
  ∀ z : ℝ, z ≠ 0 → P.eval z ≠ 0 → P.eval (1/z) ≠ 0 →
   1/(P.eval z) + 1/(P.eval (1/z)) = z + 1/z ↔
 ∃ k : ℕ, P = X ∗ (X^(4∗k + 2) + 1)/(X^2 + 1) ∨
  P = X ∗ (1 − X^(4∗k))/(X^2 + 1) := by sorry
```

The problem is one of logical scope and intended semantic meaning. While Lean4's type system is powerful and can often handle somewhat flexible syntax, the initial phrasing can lead to a misinterpretation of the theorem's claim. While Lean4 might parse this code without immediate syntax errors due to the right-associativity of implication, this placement leads to a misinterpretation of the theorem's intended logical structure and meaning. The original code is effectively parsed as if it were written:

Example E.6: Logical Parse in Original Autoformalized Statement

omni_theorem_219-autoformalization

```
∀ z : ℝ, (z ≠ 0 → (P.eval z ≠ 0 → (P.eval (1/z) ≠ 0 →
    (1/(P.eval z) + 1/(P.eval (1/z)) = z + 1/z ↔
  ∃ k : ℕ, P = X ∗ (X^(4∗k + 2) + 1)/(X^2 + 1) ∨
   P = X ∗ (1 − X^(4∗k))/(X^2 + 1) ))))
```

The theorem should state: A nonconstant polynomial $P(x)$ satisfies the property that for all relevant nonzero z, the equation holds if and only if $P(x)$ takes one of the specified forms. To accurately reflect the intended meaning and correct the quantifier placement, we must use parentheses to explicitly define the scope of the universal quantifier.

Example E.7: Enhanced Autoformalized Statement

omni_theorem_219-autoformalization

```
theorem omni\_theorem\_219 \{P : Polynomial ℝ\} (hP :
      P.degree > 0) : \\
  (∀ z : ℝ, z ≠ 0 → P.eval z ≠ 0 → P.eval (1/z) ≠ 0 →
   1/(P.eval z) + 1/(P.eval (1/z)) = z + 1/z) ↔
 ∃ k : ℕ, P = X ∗ (X^(4∗k + 2) + 1)/(X^2 + 1) ∨
  P = X ∗ (1 − X^(4∗k))/(X^2 + 1) := by sorry
```

E.3 Errors in Constraint Condition

The content of the constraints may not match the requirements stated in the problem or be left out. We also notice that constraints that are not mentioned in the original problem statement may be added during formalization.

Example E.8: Errors in Constraint Condition: Mismatch with known conditions

omni_theorem_748:

Let $f(x) = x^2 + ax + b$ and $g(x) = x^2 + cx + d$ be two distinct real polynomials such that the x −coordinate of the vertex of f is a root of g, the x −coordinate of the vertex of g is a root of f and both f and g have the same minimum value. If the graphs of the two polynomials intersect at the point (2012, − 2012), what is the value of $a + c$?
Prove that the answer is: −8048

Autoformalization:

```
import Mathlib

theorem omni_theorem_748 {a b c d : ℝ} (f g : ℝ → ℝ)
  (hf : f = fun x => x^2 + a ∗ x + b) (hg : g = fun x => x^2 + c ∗
      x + d)
  (hfg : f ≠ g) (hvertex : ∃ x, x = −a / 2 ∧ g x = 0) (hvertex' :
      ∃ x, x = −c / 2 ∧ f x = 0)
  (hmin : ∃ y, ∀ x, f x ≥ y ∧ g x ≥ y) (hintersect : f 2012 =
      −2012 ∧ g 2012 = −2012) :
  a + c = −8048 := by sorry
```

In Autoformalization, hmin: $\exists y, \forall x, f(x) \geq y \wedge g(x) \geq y$ means that f and g have a common lower bound, rather than: f and g have the same minimum value.

Example E.8: Errors in Constraint Condition: Lack of a key condition

omni_theorem_10:

Given 30 students such that each student has at most 5 friends and for every 5 students there is a pair of students that are not friends, determine the maximum k such that for all such possible configurations, there exists k students who are all not friends.
Prove that the answer is: 6

Autoformalization:

```
import Mathlib

open Finset

theorem omni_theorem_10 (n : ℕ) (hn : n = 30) (f : Fin n → Fin
      n → Prop)
  (hf : ∀ i j, f i j ↔ f j i) (hfn : ∀ s : Finset (Fin n), s.card
      = 5 →
  ∃ i ∈ s, ∃ j ∈ s, i ≠ j ∧ ¬f i j) :
  IsGreatest {k | ∃ t : Finset (Fin n), t.card = k ∧ ∀ i ∈ t, ∀
      j ∈ t, i ≠ j → ¬f i j} 6 := by sorry
```

The formalization fails to represent the condition "Each student has at most 5 friends". This crucial condition is necessary for accurately capturing the problem's constraints, and its absence may lead to a solution space that does not match the requirements of the original problem. Without this condition, the formalized statement does not fully reflect the situation described in the refined statement, potentially resulting in an incorrect or incomplete proof for determining the maximum

number k of students who are all not friends.

Example E.9: Errors in Constraint Condition: Redundant conditions

omni_theorem_1245

```
The fraction 1/2015 has a unique ``(restricted) partial
    fraction decomposition'' of the form
    1/2015 = a/5 + b/13 + c/31 where a, b, c are integers
    with 0 ≤ a < 5 and 0 ≤ b < 13. Find a + b.
Prove that the answer is: 14
```

Autoformalization:

```
import Mathlib

theorem omni_theorem_1245 {a b c : ℤ} (ha : 0 ≤ a ∧ a < 5)
    (hb : 0 ≤ b ∧ b < 13)
  (hc : 0 ≤ c ∧ c < 31) (h : 1 / 2015 = a / 5 + b / 13 + c / 31) : a +
    b = 14 := by sorry
```

In autoformalization, the model introduced a spurious condition: `(hc : 0 ≤ c ∧ c < 31)`.

E.4 Errors in Proof Goals

The stated proof goals may be incorrect. They could be misaligned with the original problem statement, overly simplified, too general or even omitted.

Example E.10: Errors in Proof Goals: Incorrect Proposition Logic

omni_theorem_1362

```
The Fibonacci sequence is defined as follows:
    F_0 = 0, F_1 = 1, and F_n = F_{n-1} + F_{n-2} for all
    integers n ≥ 2. Find the smallest positive integer
    m such that F_m ≡ 0(mod 127) and
    F_{m+1} ≡ 1(mod 127).
Prove that the answer is: 256
```

Autoformalization:

```
import Mathlib

def fib (n : ℕ) : ℕ := match n with
| 0 => 0
| 1 => 1
| n + 2 => fib (n + 1) + fib n

theorem omni_theorem_1362 {m : ℕ} (hm : m > 0) :
  fib m ≡ 0 [MOD 127] ∧ fib (m + 1) ≡ 1 [MOD 127] ↔ m = 256
    := by sorry
```

The fundamental error in this autoformalization lies in framing the problem as an equivalence verification. The autoformalization, by using $\leftrightarrow m = 256$, transforms the problem into a claim that the given congruences hold if and only if m is exactly 256. The correct approach to formalize the original problem would involve demonstrating that The value $m = 256$ satisfies the conditions:fib$256 \equiv 0 \pmod{127} \wedge$fib$(256+1) \equiv 1 \pmod{127}$,and that no smaller positive integer $m' < 256$ also satisfies these conditions.

Example E.11: Errors in Proof Goals: Incomplete Goal

olymidrefbase_551

```
Let a, b, c, d, e, f be non-negative real numbers satisfying
    a + b + c + d + e + f = 6. Find the maximal
    possible value of  a b c+b c d+c d e+d e f+e f a+f a b
     and determine all 6-tuples (a, b, c, d, e, f) for
    which this maximal value is achieved. Answer: 8 .
    Prove that the answer is: 8
```

Autoformalization:

```
import Mathlib  open Real  theorem olymid_ref_base_551 :
    IsGreatest {x | ∃ a b c d e f : ℝ, a ≥ 0 ∧ b ≥ 0 ∧ c ≥ 0
    ∧ d ≥ 0 ∧ e ≥ 0 ∧ f ≥ 0 ∧ a + b + c + d + e + f = 6 ∧ x
    = a * b * c + b * c * d + c * d * e + d * e * f + e * f * a +
    f * a * b} 8 := by sorry
```

The core issue lies in how the autoformalization treats the problem's objective – finding the maximal possible value – and the request to determine

all 6-tuples that achieve this maximum. The original problem requires the solver to not only find the maximum value but also to characterize the set of inputs that lead to this maximum. The provided autoformalization using IsGreatest completely omits any formalization of the requirement to determine all 6-tuples. It focuses solely on verifying the maximal value (8).

F Case Study in Semantic Equivalence Checking

F.1 CriticLean ID 84

Example Case 84 in CriticLean: Why HSA Correct, CriticLean Wrong

ID: 84
Gold tag: `compile_right_is_false`
Source: AoPs
CriticLean prediction: Correct
HSA prediction: Incorrect

Core mismatch. The Lean statement omits the implicit domain guard induced by cotangent (sin x != 0), which changes the admissible domain.
Why HSA is correct. HSA expands implicit domain constraints and flags missing cot-domain restriction, so it correctly marks Incorrect.
Why CriticLean is wrong. CriticLean relies on surface formula identity and treats shared omission as harmless, yielding a false positive.

Informal statement (full).

Prove the trigonometric identity:
$$\cos x - \sin x = \sin 2x(\cot x - 1).$$

Lean statement (full).

```
import Mathlib

open Real Set
open scoped Real


theorem trigonometric (x : ℝ) :
 cos x − sin x = sin (2 ∗ x) ∗ (cot x − 1) := by sorry
```

CriticLean output (full raw). `is_assistant_correct = Correct`

1. Math Assertion Analysis:
– Mathematical text: Prove the trigonometric identity: cos{x} − sin{x} = sin{2x}(cot{x} − 1).
– Implicit structure: a single equality between two trigonometric expressions, intended as a universal statement in the variable x (common reading: for all real x, or at least for all x where all expressions are defined).
– Components: variable x (real number in ordinary context), left−hand side LHS = cos x − sin x, right−hand side RHS = sin(2x) ∗ (cot x − 1). The expression cot x invokes the cotangent function (cot x = cos x / sin x), which is ordinarily undefined when sin x = 0; the mathematical statement as written does not explicitly restrict x to avoid those points.

2. Lean Statement Analysis (Proof Ignored):
– Lean theorem: theorem trigonometric (x : ℝ) : cos x − sin x = sin (2 ∗ x) ∗ (cot x − 1) := by sorry
– Quantification and typing: x is explicitly typed as a real number (x : ℝ), so the statement asserts the equality for every real x.
– LHS in Lean: cos x − sin x (matches mathematical LHS).
– RHS in Lean: sin (2 ∗ x) ∗ (cot x − 1) (matches mathematical RHS, with explicit multiplication and parentheses).
– The code uses Mathlib and opens Real, so sin, cos, cot refer to the standard real trigonometric functions from Mathlib.
– No additional assumptions or domain restrictions are present in the Lean statement (e.g., there is no hypothesis sin x ≠ 0).

3. Comparative Verification:
– Logical structure and quantifiers: Both the mathematical statement (as typically read) and the Lean statement are universal claims about a real variable x; the Lean statement makes this explicit by (x : ℝ). This matches the typical interpretation of the problem.
– Expressions and symbols: The LHS and RHS are identical in structure and symbols (cos, sin, sin(2x), cot, subtraction and multiplication). Parentheses and multiplication are used in Lean in the natural way and correspond exactly to the mathematical expression.
– Domain/definedness subtlety: The mathematical text does not explicitly state the domain restriction that cot x requires sin x ≠ 0. The Lean statement likewise contains no domain restriction and asserts the equality for all real x. Thus Lean faithfully mirrors the omission in the problem statement. If one wished to be stricter (i.e., require sin x ≠ 0), the original math problem would need to state that; the provided Lean code does not add or remove such a restriction relative to the given math text.
– Typing and validity: The Lean statement is syntactically and semantically a well−formed theorem statement in Mathlib style. It uses standard names and types (x : ℝ, cos, sin, cot). There are no extra hypotheses or extraneous constructs.

4. Conclusion:
– The Lean theorem statement is an exact formalization of the given mathematical text: it states the same equality, for the same domain as implicitly intended by the text (every real x), and uses the same trigonometric functions without adding or removing assumptions.

5. Accuracy Confirmation:
– Match confirmation: All elements match. The variable, its type (ℝ), the left− and right−hand sides, the trig functions used, and the universal nature of the claim correspond precisely between the mathematical problem and the Lean formalization. The only subtlety (the undefinedness of cot at sin x = 0) is present in both the informal problem and the Lean statement through omission of any restriction, so there is no mismatch introduced by the formalization.

HSA output (full raw). `is_assistant_correct = Incorrect`


```
1. Condition extraction (from Mathematical_Text)
{
 "variables": ["x"],
 "atomic_constraints": [],
 "quantified_constraints": [
  {
   "quantifier": "implicit universal",
   "bound_variables": ["x"],
   "range": "unspecified (natural reading: real angles)",
   "body": "cos x - sin x = sin(2x) * (cot x - 1)"
  }
 ],
 "structural_assumptions": [
  "standard trigonometric functions (cos, sin, sin(2x),
     cot) are in scope",
  "cot x denotes cos x / sin x (standard meaning)"
 ],
 "target_statement": {
  "type": "equality",
  "formula": "cos x - sin x = sin(2x) * (cot x - 1)",
  "intended_scope": "identity (i.e. for all admissible x)"
 },
 "implicit_or_derived_requirements": [
  "Because cot x appears, one normally requires sin x ≠
     0 for cot x to be defined (unless an extended
     definition is intended)",
  "Therefore the natural precise reading is: for all x in
     the domain (e.g., real x) with sin x ≠ 0, the
     equality holds. Alternatively the author might mean
     the identity of functions where the RHS is only
     considered where defined."
 ],
 "ambiguities": [
  "No explicit domain for x is given (real numbers, reals
     mod 2π, or complex?).",
  "No explicit statement whether the identity is asserted
     only where cot is defined (i.e., sin x ≠ 0) or for
```

```
      all x by continuity/extension."
 ]
}

Condition extraction (from Lean4Code)
{
 "variables": ["x : ℝ"],
 "atomic_constraints": [],
 "quantified_constraints": [
  {
   "quantifier": "universal",
   "bound_variables": ["x"],
   "range": "ℝ",
   "body": "cos x - sin x = sin (2 * x) * (cot x - 1)"
  }
 ],
 "structural_assumptions": [
  "Uses Mathlib; cos, sin, cot are the library functions
      on reals",
  "No additional hypotheses (e.g., sin x ≠ 0) are
      assumed in the theorem statement"
 ],
 "target_statement": {
  "type": "unconditional equality for all real x",
  "formula": "cos x - sin x = sin (2 * x) * (cot x - 1)"
 },
 "implicit_or_derived_requirements": [
  "By the theorem shape, the claim is for every real x
      without restriction. Any domain restriction
      implicit in the informal text is not present in the
      Lean statement."
 ],
 "ambiguities": [
  "Depends on Mathlib's definition of cot: if cot is a
      partial notion requiring sin x ≠ 0, then the
      formal statement should have included sin x ≠ 0.
      The Lean code does not include such a hypothesis."
 ]
}

2. Math Assertion Analysis:
- Variables: one variable x; informal text does not state its
      type, but context (trigonometric identity) suggests x is
      a real angle (or real-valued), and cot x implies sin x ≠ 0
      where RHS is formed.
- Quantification: implicit universal quantifier "for all x
      (in an appropriate domain)"; not explicitly stated.
- Key constraints: presence of cot x typically imposes a domain
      restriction sin x ≠ 0. The identity is normally
      interpreted pointwise where both sides are defined.
- Target: show equality cos x − sin x = sin(2x) ∗ (cot x − 1) under
      the natural domain assumptions.
- Ambiguity that matters: whether the author intended the
      equality for all real x without proviso (using some
      extension of cot) or only for x with sin x ≠ 0.

3. Lean Statement Analysis (proof ignored):
- Variables/types: the theorem introduces x : ℝ, so x is
      explicitly a real number.
- Quantifier: the theorem is universally quantified over all
      real x (no hypotheses).
- Formula exactly: cos x − sin x = sin (2 ∗ x) ∗ (cot x − 1).
- Preconditions: none. In particular, there is no sin x ≠ 0
      hypothesis.
- Syntactic validity: the Lean statement is syntactically
      valid Lean 4 code (imports Mathlib etc.). It uses cot from
      Mathlib; how Mathlib treats cot w.r.t. division-by-zero
      is relevant but not asserted in the statement.

4. Comparative Verification:
- Matching parts:
  - Both informal text and Lean statement assert the same
      algebraic equality as the target formula.
  - Both intend a universal statement over some domain of x;
      Lean makes it explicit (x : ℝ).
- Mismatches / Issues:
  - Domain restriction for cot: The informal problem, by using
      cot x, implicitly restricts to x where cot x is defined
      (i.e., sin x ≠ 0) unless the author meant an identity of
      functions with domain qualification. The Lean statement
      asserts the equality for every real x without any
      hypothesis sin x ≠ 0. This is a semantic difference: Lean
      removes an implicit but material domain restriction.
  - Ambiguity resolution: The Lean code fixes the domain to all
      real numbers. The informal statement did not explicitly
      fix that; the more conservative precise reading would
      require "for all real x with sin x ≠ 0". Because that
      condition materially changes admissible x, omitting it
      is a mismatch.
  - No other structural assumptions or quantifier changes
      occur: Lean is explicit about ℝ and universal
      quantification, which is acceptable provided the
      intended informal domain was ℝ and one meant
      unconditional identity. But since cot introduces a
      definitional domain issue, the omission is important.

5. Conclusion:
- The Lean statement is not an exact and faithful
      formalization of the informal mathematical text under
      the conservative reading that cot x requires sin x ≠ 0.
      The Lean theorem asserts the identity for all real x
      without any restriction; the informal statement at
      minimum implicitly restricts to x where cot is defined.
      Therefore the formalization is
      incorrect/insufficiently faithful.

6. Accuracy Confirmation / Discrepancies:
- Discrepancy 1 (critical): Missing domain restriction sin x ≠
      0. The informal statement implicitly requires cot x to be
      defined (sin x ≠ 0) or at least leaves that as an intended
      domain qualification; the Lean theorem states the
      equality for every real x with no proviso. Omitting sin x
      ≠ 0 changes the set of x for which the statement is
      claimed and is therefore not faithful.
- Discrepancy 2 (minor/clarifying): The informal text does not
      name the domain of x explicitly (could be reals, angles
      modulo 2π, etc.); Lean fixes x : ℝ. This is acceptable if
      the intended domain was reals, but since the informal
      text is ambiguous, this choice should be noted. The main
      correctness issue remains the missing cot-domain
      restriction.

Overall judgment: the formalization is Incorrect because it
      omits the implicit definitional/domain constraint (sin
      x ≠ 0) that materially affects where the identity can be
      interpreted.
```

F.2 CriticLean ID 148

Example Case 148: Why HSA Correct, CriticLean Wrong

ID: 148
Gold tag: `compile_right_is_false`
Source: `BlueMO`
CriticLean prediction: `Correct`
HSA prediction: `Incorrect`

Core mismatch. The target-level extremal claim is misformalized (hard-coded maximum value assertion), not merely a premise-level wording issue.
Why HSA is correct. HSA isolates target semantics in channel T and detects inconsistency in the extremal objective, correctly predicting Incorrect.
Why CriticLean is wrong. CriticLean mainly checks local expression/premise alignment and misses the target-semantic error, causing false positive acceptance.

Informal statement (full).

```
Algebraic expression r v z − r w y − s u z + s w x + t u y − t v x.
      \label{eq1} In this expression, \(r, s, t, u, v, w, x, y, z\)
      can take either \( + 1\) or \( − 1\) respectively. (1) Prove
      that the value of the algebraic expression is always an
      even number. (2) Find the maximum value that this
      algebraic expression can take.
```

Lean statement (full).

```
import Mathlib theorem algebra_486948 : (∀ r s t u v w x y z : ℤ,
      (r = 1 ∨ r = −1) ∧ (s = 1 ∨ s = −1) ∧ (t = 1 ∨ t = −1) ∧     (u
      = 1 ∨ u = −1) ∧ (v = 1 ∨ v = −1) ∧ (w = 1 ∨ w = −1) ∧ (x = 1 ∨
      x = −1) ∧ (y = 1 ∨ y = −1) ∧     (z = 1 ∨ z = −1) → Even (r ∗
      v ∗ z − r ∗ w ∗ y − s ∗ u ∗ z + s ∗ w ∗ x + t ∗ u ∗ y − t ∗ v ∗ x))
      ∧   IsGreatest {n | ∃ r s t u v w x y z : ℤ, (r = 1 ∨ r = −1) ∧
      (s = 1 ∨ s = −1) ∧ (t = 1 ∨ t = −1) ∧     (u = 1 ∨ u = −1) ∧ (v
      = 1 ∨ v = −1) ∧ (w = 1 ∨ w = −1) ∧ (x = 1 ∨ x = −1) ∧ (y = 1 ∨
      y = −1) ∧     (z = 1 ∨ z = −1) ∧ n = r ∗ v ∗ z − r ∗ w ∗ y − s ∗
      u ∗ z + s ∗ w ∗ x + t ∗ u ∗ y − t ∗ v ∗ x} 18 := by sorry
```

CriticLean output (full raw). `is_assistant_correct = Correct`

1. Math Assertion Analysis: The mathematical problem has two assertions about the expression E = r∗v∗z − r∗w∗y − s∗u∗z + s∗w∗x + t∗u∗y − t∗v∗x with nine variables r,s,t,u,v,w,x,y,z each taking values in {+1, −1}. (1) For every choice of these nine signs, E is even. (2) The maximum possible value of E over all such sign choices is requested.

2. Lean Statement Analysis (Proof Ignored): The Lean statement is a single theorem asserting a conjunction of two claims: (A) ∀ r s t u v w x y z : ℤ, (r = 1 ∨ r = −1) ∧ ... ∧ (z = 1 ∨ z = −1) → Even (r ∗ v ∗ z − r ∗ w ∗ y − s ∗ u ∗ z + s ∗ w ∗ x + t ∗ u ∗ y − t ∗ v ∗ x). This quantifies over integers and constrains each variable to be 1 or −1, then claims Even on the same algebraic expression. (B) IsGreatest {n | ∃ r s t u v w x y z : ℤ, (r = 1 ∨ r = −1) ∧ ... ∧ (z = 1 ∨ z = −1) ∧ n = expression} 18. This states that 18 is the greatest element of the set of integers n that occur as values of the expression for some allowed choice of the nine variables (IsGreatest includes membership and upper bound properties).

3. Comparative Verification: Semantic alignment: The Lean statement precisely captures "each variable is ±1" by equalities r = 1 ∨ r = −1, etc., and quantifies over all such choices to assert Even of the expression, matching (1). For (2) the use of the set {n | ∃ r ... n = expression} exactly describes the multiset/range of possible integer values of the expression, and IsGreatest ... 18 asserts that 18 is in that set and that no element of the set exceeds 18, i.e. 18 is the maximum. Typing: variables are ℤ, matching integer ±1 choices; Even is the standard integer−even predicate; IsGreatest is appropriate on a Set ℤ with the usual ≤. There are no added constraints or missing quantifiers: existence quantifier is used for the set definition, and universal quantifier for the evenness claim. No extraneous variables or assumptions are introduced. Concerning syntactic/formal Lean usage: the constructs used (∀, ∨ equalities, Even, set comprehension, ∃, IsGreatest) are appropriate for formalizing the intended meaning. (Minor note: typical Lean style would place the import and theorem on separate lines, but that is a formatting issue rather than a semantic mismatch.)

4. Conclusion: The Lean theorem statement is an accurate and faithful formalization of the two parts of the mathematical problem: (1) universal evenness for all choices of ±1, and (2) that the maximum attainable value of the expression is 18.

5. Accuracy Confirmation: All elements match: variables, their domain and constraints (±1), the exact algebraic expression, the claim of evenness for all assignments, and the maximum being 18 expressed via IsGreatest on the set of attained values. No essential aspect of the mathematical statement is missing or misstated in the Lean formalization.

HSA output (full raw). `is_assistant_correct = Incorrect`

```
extraction (from Mathematical_Text): {
 "variables": ["r","s","t","u","v","w","x","y","z"],
 "atomic_constraints": [
  "for each v in {r,s,t,u,v,w,x,y,z}: v ∈ {+1, -1}"
 ],
 "quantified_constraints": [
  "(1) For statement (1): implicitly ∀ r,s,t,u,v,w,x,y,z
      with each ∈ {+1,-1}, consider E(r,...,z) and prove
      E(r,...,z) is even.",
  "(2) For statement (2): consider the set of values
      {E(r,...,z) | each variable ∈ {+1,-1}} and find its
      maximum."
 ],
 "structural_assumptions": [
  "No further structure (variables independent,
      arithmetic over integers is intended)."
 ],
 "target_statement": {
  "part1": "For all assignments of r,s,t,u,v,w,x,y,z ∈
      {+1,-1}, the expression E = r*v*z - r*w*y - s*u*z +
      s*w*x + t*u*y - t*v*x is even.",
  "part2": "Determine the maximum value of E over all
      assignments r,s,t,u,v,w,x,y,z ∈ {+1,-1}."
 },
 "implicit_or_derived_requirements": [
  "Variables are independent and can each be chosen ±1;
      arithmetic parity is meaningful (treating ±1 as
      integers)."
 ],
 "ambiguities": [
  "The problem statement does not explicitly declare the
      ambient number type (integers vs reals), but ±1
      suggests integer domain; this is a reasonable and
      conventional interpretation."
 ]
}

extraction (from Lean4Code): {
 "variables": ["r","s","t","u","v","w","x","y","z","n"],
 "types": {
  "r..z": "ℤ",
  "n": "ℤ (as element of set comprehension)"
 },
 "atomic_constraints": [
  "for each v in {r,s,t,u,v,w,x,y,z}: (v = 1) ∨ (v = -1)",
  "n = r*v*z - r*w*y - s*u*z + s*w*x + t*u*y - t*v*x  (in
      the set comprehension)"
 ],
 "quantified_constraints": [
  "Conjunct 1: ∀ r s t u v w x y z : ℤ, P(r..z) →
      Even(expr(r..z)).",
  "Conjunct 2: IsGreatest { n | ∃ r..z : ℤ, P(r..z) ∧ n
      = expr(r..z) } 18.",
  "where P(r..z) is the conjunction of (v = 1 ∨ v = -1)
      for all variables."
 ],
 "structural_assumptions": [
  "Uses Lean's Even predicate and IsGreatest on a set of
      integers.",
  "Conjunction combines two claims: universal parity
      claim and a claim that 18 is the maximum value
      realized."
 ],
 "target_statement": {
  "part1": "∀ r..z with each = ±1 (integers),
      Even(expr(r..z)).",
  "part2": "18 is the greatest element of the set of
      integer values of the expression over such
      assignments (i.e., maximum = 18)."
 },
 "implicit_or_derived_requirements": [
  "IsGreatest requires 18 ∈ the set (∃ assignment giving
      value 18) and all other values ≤ 18.",
  "Using ℤ domain for variables and result."
 ],
 "ambiguities": [
  "Choice of ℤ is explicit; parity predicate 'Even' is
      used on integers. No other ambiguities in the Lean
      encoding itself."
 ]
}

2. Math Assertion Analysis:
– Variables: nine symbols r,s,t,u,v,w,x,y,z; each takes either +1
    or −1 independently.
– Expression E = r∗v∗z − r∗w∗y − s∗u∗z + s∗w∗x + t∗u∗y − t∗v∗x.
    Each term is a product of three ±1's, hence each term ∈
    {+1,−1}.
– Claims to be formalized: (A) For all assignments, E is even. (B)
    The maximum possible value of E across all assignments.
– Logical shape: (A) universal parity property; (B) an
    optimization / existence of maximum over finite domain
    of 2^9 assignments.
– Implicit: arithmetic over integers (parity meaningful). The
    math text does not provide the numeric maximum
    explicitly.

3. Lean Statement Analysis (proof ignored):
– The Lean statement is a conjunction of two pieces:
  ∗ (∀ r s t u v w x y z : ℤ, P(r..z) → Even(expr(r..z))) where P
      enforces each variable equals 1 or −1.
  ∗ IsGreatest S 18 where S = { n | ∃ r..z : ℤ, P(r..z) ∧ n = expr(r..z)
      }.
– Variables are explicitly integers and constrained to be ±1
    via disjunctions.
– The first conjunct matches the universal−even property
    structure.
```

– The second conjunct asserts that 18 is the greatest element of the set of achievable values; i.e., Lean claims maximum = 18.
– Syntactic/formal notes: uses Even and IsGreatest from Mathlib; quantification and set comprehension are properly formed.

4. Comparative Verification:
– Agreement:
∗ Both the informal statement and the Lean code quantify over nine variables that must be ±1.
∗ Both target the same two kinds of claims: universal parity and existence of a greatest value (maximum) for the expression over those assignments.
∗ The Lean encoding faithfully represents the logical forms: universal implication for parity, and IsGreatest on the image set for maximum.
∗ Using integers (ℤ) and Even is a natural, acceptable formalization of ±1 and parity.
– Mismatch / Critical issue:
∗ The mathematical problem asks to 'find the maximum value' but does not itself state the numeric maximum. The Lean statement asserts that the maximum is 18 (IsGreatest ... 18).
∗ This numeric claim (18) is an extra specific numeric assertion that is not present in the problem statement. Unless the informal Mathematical_Text specified 18 as the answer (it does not), the Lean formalization has introduced a concrete numeric value as the claimed maximum.
∗ Semantically, the correct maximum (from direct reasoning: expression is sum of six ±1 terms so max ≤ 6) is not 18; thus the Lean statement's numeric conclusion is not faithful to the (intended) problem. Regardless of the actual correct numeric value, the problem statement required finding the maximum; the formalization must assert the correct maximum or leave it as an existential/maximization claim. The Lean code instead asserts 18, which is an unjustified and likely incorrect addition.
– Other minor points (not affecting correctness of formalization except for the above):
∗ The use of ℤ is acceptable though the problem didn't name the ambient type. No other assumptions were added/omitted.

5. Conclusion:
– The Lean theorem mostly mirrors the structure of the mathematical problem: same variables, same constraints (each variable ±1), parity claim universal, and a maximum claim formulated as IsGreatest on the image set.
– However, it departs in a critical and explicit way by asserting that the maximum value is 18. The mathematical prompt did not contain that specific numeric value; the formalization must either prove a maximum value or assert the correct maximum. By hard–coding 18 as the maximum, the Lean statement does not faithfully formalize the problem as given.

6. Accuracy Confirmation / Discrepancies:
– Verdict: Incorrect.
– Discrepancy list:
1) Incorrect numeric maximum: The Lean statement asserts IsGreatest ... 18, i.e., that the maximum value of the expression over all assignments is 18. The original problem did not state a numeric maximum and required finding it; the formalization must reflect the correct maximum. The assertion that the maximum equals 18 is an unjustified and incorrect addition to the problem formalization.

Summary: The encoding of variables, domain (±1), the parity claim, and the overall logical structure are correct. The single critical error is the specific numeric claim that the maximum is 18; this misformalizes the 'find the maximum' part. Therefore the Lean statement is not a faithful and exact formalization of the mathematical problem.

F.3 CriticLean ID 102

Example Case 102: Why HSA Correct, CriticLean Wrong

ID: 102
Gold tag: compile_right_is_false
Source: AoPs
CriticLean prediction: Correct
HSA prediction: Incorrect

Core mismatch. The final inequality has a typed-constraint inconsistency (cardinality terms vs real-valued log expression handling), which is semantically non-trivial at formal level.
Why HSA is correct. HSA checks typed constraint alignment and detects this mismatch in the target inequality, correctly rejecting the formalization.
Why CriticLean is wrong. CriticLean dismisses the type interaction as benign coercion, resulting in false positive acceptance.

Informal statement (full).

Given an integer $n \geq 4$, let $S = \{1, 2, \ldots, n\}$. Consider two subsets A and B of S such that for every pair (a, b) with $a \in A$ and $b \in B$, the expression $ab + 1$ is a perfect square. Prove that $\min\{|A|, |B|\} \leq \log_2 n$.

Lean statement (full).

```
import Mathlib


theorem log_n (n : ℕ) (hn : 4 ≤ n) (A B : Finset ℕ)
 (hA : A ⊆ Finset.Icc 1 n) (hB : B ⊆ Finset.Icc 1 n)
 (hAB : ∀ x ∈ A, ∀ y ∈ B, ∃ m, m^2 = x ∗ y + 1) :
 min (A.card) (B.card) ≤ Real.logb 2 n := by sorry
```

CriticLean output (full raw). is_assistant_correct = Correct

Your detailed CoT analysis:
1. Math Assertion Analysis: [
– Variables and types: n is an integer with n ≥ 4; S = {1,2,...,n}; A and B are subsets of S. Cardinalities |A| and |B| are natural numbers. The quantity log_2 n is the base–2 logarithm of n (a real number).
– Hypotheses: n ≥ 4; A ⊆ S; B ⊆ S; for every pair (a,b) with a ∈ A and b ∈ B, ab + 1 is a perfect square (i.e. ∃ integer m with m^2 = ab + 1).
– Conclusion: min{|A|, |B|} ≤ log_2 n.
– Logical structure: universal quantification over the given data (once n, A, B satisfying hypotheses are fixed), then a numerical inequality comparing the minimum of two natural numbers to a real–valued logarithm.
]

2. Lean Statement Analysis (Proof Ignored): [
– Signature: theorem log_n (n : ℕ) (hn : 4 ≤ n) (A B : Finset ℕ) ... : ...
– n : ℕ with hypothesis hn : 4 ≤ n matches the n ≥ 4 assumption.
– A, B are Finset ℕ (finite sets of natural numbers).
– hA : A ⊆ Finset.Icc 1 n and hB : B ⊆ Finset.Icc 1 n assert A and B are subsets of the integer interval [1,n], so they are subsets of S = {1,...,n}.
– hAB : ∀ x ∈ A, ∀ y ∈ B, ∃ m, m^2 = x ∗ y + 1 exactly encodes "for every a ∈ A and b ∈ B, ab + 1 is a perfect square." The existential m will be inferred as a natural number (m : ℕ) because x∗y+1 : ℕ.
– Conclusion: min (A.card) (B.card) ≤ Real.logb 2 n. Here A.card and B.card are ℕ; min returns a ℕ. Real.logb 2 n is the base–2 logarithm of n (as a real). Coercions from ℕ to ℝ are available in mathlib, so the inequality is intended between the coerced left–hand side (as a real) and Real.logb 2 (n : ℝ).
]

3. Comparative Verification: [
– Quantifiers and variables: The Lean statement explicitly quantifies n, A, B and matches the mathematical variables and domains (n ∈ ℕ, A,B finite subsets of {1..n}).
– Domain restriction: The math text says A,B ⊆ S with S = {1..n}; Lean uses A ⊆ Icc 1 n and B ⊆ Icc 1 n, which is equivalent and correct.
– Perfect–square condition: The Lean hAB matches the natural–number perfect–square predicate (∃ m, m^2 = x∗y + 1).
– Conclusion form: The mathematical conclusion min{|A|,|B|} ≤ log_2 n is represented as min (A.card) (B.card) ≤

Real.logb 2 n. This preserves the intended inequality comparing the minimum cardinality to the base–2 logarithm of n.
– Typing/coercion remarks: In Lean the left–hand side is a ℕ and the right–hand side is a Real. Mathlib provides canonical coercions from ℕ to ℝ and numeral coercions, so the statement as written will be interpreted as comparing the real cast of the natural number min (A.card) (B.card) to Real.logb 2 (n : ℝ). This is a standard way in Lean to express such mixed–type inequalities; no semantic drift occurs here.
– No extra or missing constraints: The Lean statement does not introduce extra hypotheses beyond those in the math problem and does not omit any required hypothesis. The optional explicit naming of S is omitted but is unnecessary since A,B ⊆ Icc 1 n expresses the same content.
]

4. Conclusion: [
The Lean theorem statement is a faithful and exact formalization of the mathematical problem. The variables, hypotheses, the perfect–square condition, the restriction of A and B to {1,..,n}, and the conclusion min(|A|,|B|) ≤ log_2 n are all present and correctly represented. The minor technical point that the inequality compares a natural number to a real is handled by standard coercions in mathlib and does not change the meaning.
]

5. Accuracy Confirmation: [
All elements match: n ≥ 4, A,B ⊆ {1..n}, ∀ a ∈ A, ∀ b ∈ B, ∃ m with m^2 = ab+1, and the asserted inequality min(A.card,B.card) ≤ log_2 n are all present and correctly encoded. The use of Real.logb 2 n corresponds to log_2 n. The finite–set type Finset ℕ and card produce the expected natural–number cardinalities. The only annotation–worthy issue is the implicit coercion from ℕ to ℝ when comparing the natural–number minimum to Real.logb 2 n; this is standard in Lean/mathlib and does not constitute a semantic mismatch.]

HSA output (full raw). `is_assistant_correct = Incorrect`

```
Extraction (informal statement):
{
 "variables": ["n (integer)", "S = {1,2,...,n}", "A (subset
     of S)", "B (subset of S)"],
 "atomic_constraints": [
  "n is an integer",
  "n ≥ 4",
  "S = {1,2,...,n}",
  "A ⊆ S",
  "B ⊆ S",
  "for all a ∈ A and b ∈ B, ab + 1 is a perfect square
      (i.e., ∃k ∈ Z, k^2 = ab+1)"
 ],
 "quantified_constraints": [
  "∀ a (a ∈ A), ∀ b (b ∈ B), ∃ k (integer) such that
      k^2 = a*b + 1"
 ],
 "structural_assumptions": [
  "S is finite set {1,..,n}",
  "A and B are subsets of S (hence finite, elements in
      1..n)"
 ],
 "target_statement": {
  "claim": "min(|A|, |B|) ≤ log_2 n",
  "note": "log_2 denotes base-2 logarithm; comparison is
      between cardinalities (nonnegative integers) and a
      logarithmic quantity"
 },
 "implicit_or_derived_requirements": [
  "|A| and |B| are natural numbers (cardinalities)",
  "log_2 n is a real number (the statement compares an
      integer to a real)"
 ],
 "ambiguities": [
  "The problem does not state explicitly whether log_2 n
      is to be rounded (floor/ceiling) -- interpreted as
      real-valued log base 2.",
  "The term 'integer n ≥ 4' -- in formal work usually
      replaced by n ∈ ℕ, n ≥ 4 (nonnegative integer);
      implicit that n ≥ 1 so S is nonempty."
 ]
}

Extraction (Lean formalization):
{
 "variables": ["n : ℕ", "A : Finset ℕ", "B : Finset ℕ"],
 "atomic_constraints": [
  "hn : 4 ≤ n",
  "hA : A ⊆ Finset.Icc 1 n",
  "hB : B ⊆ Finset.Icc 1 n",
  "hAB : ∀ x ∈ A, ∀ y ∈ B, ∃ m, m^2 = x * y + 1"
 ],
 "quantified_constraints": [
  "∀ x, (x ∈ A) → ∀ y, (y ∈ B) → ∃ m, m^2 = x*y + 1"
 ],
 "structural_assumptions": [
  "A and B are finite sets of natural numbers with all
      elements in the inclusive interval [1,n]"
 ],
 "target_statement": {
  "claim": "min (A.card) (B.card) ≤ Real.logb 2 n",
  "note": "Real.logb 2 n is the base-2 logarithm of n (a
      real). Left-hand side is a Nat (min of two
      cardinals)."
 },
 "implicit_or_derived_requirements": [
  "m in the existential is intended as a natural number
      (type inferred from context of nat arithmetic)",
  "coercions between Nat and Real are not explicitly
      provided in the statement"
 ],
 "ambiguities": [
  "Use of Finset.Icc 1 n: assumes an available finset
      interval constructor; depending on mathlib version
      this exists, but not explicit in the problem text.",
  "Type mismatch potentially between Nat and Real in the
      inequality min(A.card) ≤ Real.logb 2 n (no
      explicit cast)."
 ]
}

1. Math Assertion Analysis:
– Variables and domains: n (integer, n ≥ 4); S = {1,..,n}; A,B
      subsets of S (finite subsets of positive integers ≤ n).
– Core constraint: ∀ a ∈ A, ∀ b ∈ B, ab + 1 is a perfect square
      (there exists integer k with k^2 = ab+1).
– Target: inequality comparing min(|A|,|B|) (a nonnegative
      integer) with log_2 n (a logarithmic real); concluding
      min(|A|,|B|) ≤ log_2 n.
– Implicit: cardinalities are natural numbers; log_2 is base–2
      real logarithm; no rounding is specified.

2. Lean Statement Analysis (proof ignored):
– Variables/types: n : ℕ, A,B : Finset ℕ.
– Preconditions: hn : 4 ≤ n (n is a natural number at least 4); hA
      : A ⊆ Finset.Icc 1 n; hB : B ⊆ Finset.Icc 1 n.
– Pairwise condition: hAB : ∀ x ∈ A, ∀ y ∈ B, ∃ m, m^2 = x ∗ y + 1.
      (m's type is not written but inferred -- in this context m :
      ℕ is natural.)
– Conclusion: min (A.card) (B.card) ≤ Real.logb 2 n.
– Type/typing observations: A.card and B.card are natural
      numbers (Nat). Real.logb 2 n is a real number (ℝ). The
      inequality mixes Nat (left) and Real (right) without an
      explicit cast (e.g., (A.card : ℝ) ≤ Real.logb 2 n). As
      written, this is a type mismatch in Lean: ≤ expects
      operands of the same ordered type. Unless there is an
      implicit coercion instance or overloaded ≤ between Nat
      and Real available in scope (which is not standard), this
      statement will not type–check as is.
– Minor library/detail notes: Finset.Icc 1 n is used to express
      the subset–of–{1..n} condition; depending on mathlib4
      naming this may exist (Finset.Icc) or require a different
      constructor, but semantically it matches S = {1..n}.

3. Comparative Verification:
– Domain of n: informal says integer n ≥ 4; Lean uses n : ℕ with
      4 ≤ n. Semantic match: correct and conventional
      (nonnegative integer).
– S and membership: informal S = {1,..,n}; Lean enforces A,B ⊆
      Finset.Icc 1 n. Matches the intended domain of elements.
– A,B are subsets: Lean's A,B : Finset ℕ plus subset constraints
      exactly formalize that A,B ⊆ S.
– Pairwise–square condition: informal ∀ a∈A, b∈B, ab+1 is a
      perfect square (integer square). Lean's hAB uses ∃ m with
      m^2 = x∗y+1; m is inferred natural -- this matches the
      informal condition.
```

– Target inequality: both state min{|A|,|B|} ≤ log_2 n. Lean uses min (A.card) (B.card) and Real.logb 2 n. Semantically this matches the informal target, except for the following critical formalization issue:
  – Type mismatch: the left–hand side is Nat, the right–hand side is Real. The informal statement compares cardinalities (naturals) to the real–valued log_2 n; in a formal Lean statement an explicit coercion of the Nat to ℝ (or use of a real–valued cardinality) is required. The Lean code omits such coercion, so the statement as written is not well–typed in Lean.
– Additional potential differences/ambiguities:
  – The informal problem does not require A,B nonempty. Lean allows empty Finsets; vacuously the pairwise–square condition with empty A or B is trivially satisfied and the inequality min=0 ≤ log_2 n holds. This is consistent with the informal statement (which also allows empty subsets unless otherwise stated).
  – The Lean formalization does not define S as a named object; instead it encodes membership via the subset relation to Finset.Icc 1 n –– *semantically equivalent.*

4. Conclusion:
– The Lean statement captures the mathematical intent and constraints (domains of n, A, B, the pairwise perfect–square condition, and the target inequality) except for a crucial formal–typing detail: the inequality mixes Nat and Real without an explicit coercion. In Lean, this is a concrete syntactic/type error which prevents the statement from being a correct, well–formed formalization as written.
– Therefore the Lean statement is not an exact and faithful formalization in the sense of being a valid Lean theorem statement ready to be proved without modification.

5. Accuracy Confirmation / List of discrepancies and their effects:
– Missing numeric coercion between Nat and Real in the inequality: min (A.card) (B.card) has type Nat, while Real.logb 2 n has type ℝ. Effect: the Lean statement, as written, will not type–check; it needs an explicit coercion (for example: (min A.card B.card : ℝ) ≤ Real.logb 2 (n : ℝ) or Real.logb 2 (n : ℝ) with casts) to be a valid formalization. This is a substantive issue (not just stylistic) because it invalidates the code as a correct Lean statement.
– Minor library/style points (not fatal to semantics but worth noting): use of Finset.Icc 1 n to denote {1,..,n} depends on mathlib availability; if Finset.Icc is not defined in the used library version one must instead use Finset.range or another constructor. This does not change semantics but may require small syntactic edits.

Final judgment: Incorrect –– *the formalization is semantically faithful in content, but as written it is not a correct Lean statement because of a type mismatch in the final inequality (Nat vs Real) and therefore cannot be accepted as an exact and faithful formalization without the small but essential fix of inserting an explicit coercion or otherwise matching types.*

F.4 Failed CriticLean ID 183

Example Case 183: CriticLean vs HSA (Full Raw Output)

ID: 183
Gold tag: compile_right_is_right
Source: DeepMath-103k
CriticLean prediction: Correct
HSA prediction: Incorrect

Informal statement (full).

Evaluate the limit: \[\lim_{n \to \infty} \left(\sqrt[3]{\frac{\sin n}{n} + n^3 – n^2} – \sqrt[3]{n^3 + n} \right) \]
Prove that the answer is –\dfrac{1}{3}.

Lean statement (full).

```
import Mathlib

open Topology Filter Real Complex TopologicalSpace Finset
open scoped BigOperators
theorem limit :
 Tendsto (λ n => ((Real.sin n / n + n^3 – n^2) ^ ((1 : ℝ) / 3)) –
 ((n^3 + n) ^ ((1 : ℝ) / 3))) atTop (nhds (–1 / 3)) :=
sorry
```

CriticLean output (full raw). is_assistant_correct = Correct

1. Math Assertion Analysis: The mathematical problem asks for the limit as n → ∞ of the sequence
f(n) = cbrt((sin n)/n + n^3 – n^2) – cbrt(n^3 + n)
and claims the limit equals –1/3. Relevant components: the variable n tending to infinity (conventionally an integer index), the two real–valued expressions under cube roots, the use of the real sine function, the cube–root operation (principal real cube root), and the final numeric limit –1/3.

2. Lean Statement Analysis (Proof Ignored): The Lean theorem named limit states
Tendsto (λ n => ((Real.sin n / n + n^3 – n^2) ^ ((1 : ℝ) / 3)) – ((n^3 + n) ^ ((1 : ℝ) / 3))) atTop (nhds (–1 / 3)).
Components present: a function of n given by (Real.sin n / n + n^3 – n^2)^(1/3) – (n^3 + n)^(1/3), the Tendsto predicate with filter atTop and target neighborhood nhds(–1/3). The code uses Real.sin, division by n, polynomial terms n^3, n^2, and uses raising to the real exponent (1 : ℝ) / 3 to model cube roots.

3. Comparative Verification: Structure and logical form: both state that the sequence/function n ↦ expression tends to –1/3 as n → ∞, so the top–level quantifier/limit claim matches exactly. Variables and functions: both use sin n, polynomial terms in n, subtraction of two cube–rooted quantities. The Lean statement uses rpow with exponent 1/3 to represent the cube root; for large positive real arguments rpow x (1/3) coincides with the principal real cube root, which is the intended interpretation in the math problem. Domain nuance: the mathematical problem most naturally reads n as an integer sequence (n ∈ ℕ), while the Lean statement leaves n implicit; in Lean this is compatible with atTop on ℕ or ℝ and the code as written can be interpreted with n : ℕ (coerced to ℝ where needed). Semantic constraints: the rpow notation in mathlib is defined for nonnegative bases for non–integer exponents; in this specific expression both radicands are positive for all sufficiently large n (indeed for all n ≥ 1 in the natural–number interpretation), so the use of (^) with exponent 1/3 is appropriate and does not change the intended limit. Typing and Lean usage: the expression uses Real.sin and real exponent (1 : ℝ) / 3, and Tendsto ... atTop (nhds ...) is the correct way to assert the limit equals –1/3. There are no extra hypotheses introduced nor missing top–level elements of the claim.

4. Conclusion: The Lean statement accurately formalizes the mathematical limit as presented. It expresses the same function of n, the same limiting process (n → ∞), and the same limit value –1/3. The only minor implicit choices are the type of the index n (Lean will infer an appropriate numeric domain such as ℕ with coercion to ℝ) and the representation of the cube root via real exponent 1/3; both are faithful to the usual mathematical meaning in this context and do not change the statement of the limit.

5. Accuracy Confirmation: All elements match: the sequence terms match symbol–for–symbol (sin n/n + n^3 – n^2 and n^3 + n), cube roots are represented by raising to (1/3), the limit filter atTop corresponds to n → ∞, and the target nhds(–1/3) corresponds to the asserted limit –1/3. Minor remarks (not affecting correctness): the code relies on implicit typing of n (which Lean will resolve compatibly) and on rpow being used for the cube root; since radicands are positive eventually, this is equivalent to the principal real cube root in the mathematical statement. Therefore the Lean statement is a correct and faithful formalization of the given problem.

HSA output (full raw). is_assistant_correct = Incorrect

1. Extraction (from Mathematical_Text)


```
{
 "variables": [
  {"name": "n", "role": "index variable", "domain": "ambiguous
      (typical: natural numbers), tends to +infty"}
 ],
 "atomic_constraints": [
  "expression1: sin(n)/n + n^3 - n^2 (must be real)",
  "expression2: n^3 + n (must be real)",
  "cube_roots of expression1 and expression2 (real cube
      root assumed)",
  "division by n (requires n != 0 at points where used)"
 ],
 "quantified_constraints": [
  {"quantifier": "limit/implicit universal over tail",
      "variable": "n", "range": "n -> +infty (index
      variable)"}
 ],
 "structural_assumptions": [
  "sin is the real sine function (so its argument is
      real)",
  "cube root is the real cube root (odd root defined for
      all real inputs) unless otherwise specified"
 ],
 "target_statement": {
  "form": "limit",
  "expression": "lim_{n->infty} (cuberoot(sin n / n + n^3
      - n^2) - cuberoot(n^3 + n))",
  "value": "-1/3"
 },
 "implicit_or_derived_requirements": [
  "For the expression to be meaningful for large n, one
      needs n to be (eventually) large positive so that
      the quantities under the cube roots are real and
      well-behaved (in practice true for large real or
      integer n).",
  "If n is an integer index (typical sequence), division
      by n excludes n=0, but the limit as n->infty only
      requires values for large n>0.",
  "No explicit domain for n was given in the informal
      statement (ambiguity: Z, N, or R)."
 ],
 "ambiguities": [
  "Domain/type of n: the informal statement does not
      state whether n ranges over natural numbers
      (sequence) or real numbers (real variable).",
  "Interpretation of 'cube root': an odd real root
      (defined for all reals) is standard, while some
      formal systems use real rpow which requires
      nonnegative base for arbitrary real exponents."
 ]
}
```


2. Extraction (from Lean4Code)


```
{
 "variables": [
  {"name": "n", "role": "function argument", "domain": "ℝ
      (inferred from Real.sin n and real arithmetic)"}
 ],
 "atomic_constraints": [
  "uses Real.sin n / n + n^3 - n^2",
  "uses ( ... ) ^ ((1 : ℝ) / 3) (Lean/Mathlib real rpow
      notation)",
  "uses (n^3 + n) ^ ((1 : ℝ) / 3)",
  "Tendsto (λ n => ... ) atTop (nhds (-1 / 3)) meaning n
      -> +infty in ℝ with filter atTop"
 ],
 "quantified_constraints": [
  {"quantifier": "Tendsto", "variable": "n", "range": "n ∈ ℝ,
      n -> +infty (atTop)"}
 ],
 "structural_assumptions": [
  "The Lean expression uses real exponentiation with
      non-integer exponent (1/3). In Mathlib this is rpow
      which semantically requires the base to be positive
      (or uses a principal branch); the code does not add
      an explicit hypothesis about eventual positivity.",
  "No explicit restriction n ≠ 0 is stated; the lambda
      is a total function on ℝ as written in the code
      context."
 ],
 "target_statement": {
  "form": "Tendsto",
  "expression": "(λ n => (Real.sin n / n + n^3 - n^2) ^
      ((1 : ℝ) / 3) - (n^3 + n) ^ ((1 : ℝ) / 3))",
  "limit": "nhds(-1/3)"
 },
 "implicit_or_derived_requirements": [
  "Because rpow with a non-integer exponent requires the
      base to be positive in Mathlib, the Lean statement
      tacitly requires the two inner bases to be positive
      for the values of n considered (at least eventually
      as n->+infty).",
  "The Lean code fixes the domain to reals (ℝ) rather
      than integers or naturals."
 ],
 "ambiguities": [
  "How Mathlib's rpow is interpreted at nonpositive bases
      in this code is not explicitly stated; the code
      uses the notation x ^ ((1 : ℝ)/3) which in Mathlib
      has domain subtleties."
 ]
}
```


3. Math Assertion Analysis

- Variables and quantifiers: The informal problem has a single index n and the statement lim_{n–>∞} of an expression. The natural interpretation in many textbooks is n ∈ N (sequence) but the text did not state this explicitly; n could also be a real variable tending to +∞ (function limit). This ambiguity materially affects typing in a formal system.
- Atomic formulae: two cube–root expressions built from polynomials in n and a sin(n)/n perturbation. The main analytic content is the asymptotic expansion as n–>∞ leading to a finite limit −1/3.
- Implicit assumptions: the cube root in informal calculus is the real cube root (odd root), defined for all real inputs; division by n assumes n ≠ 0 when evaluating finite values but the limit concerns large n so n=0 is irrelevant.

4. Lean Statement Analysis (proof ignored)

- The Lean code concretely interprets n as a real variable (ℝ) because Real.sin expects a real argument and the arithmetic uses real polynomial operations.
- The code uses the notation (x ^ ((1 : ℝ) / 3)) which refers to real exponentiation (rpow) with a non–integer exponent. In Mathlib, rpow has domain considerations: for general real exponents it typically requires the base to be nonnegative (and strictly positive for some definitions) to be well–defined. The Lean theorem as written does not add any explicit hypothesis ensuring positivity of the bases for all n, but the tendsto atTop concerns behavior for large n where those bases are indeed positive.
- The target limit nhds (−1/3) exactly matches the numerical target −1/3 from the informal statement.

5. Comparative Verification

Matches:

- Both statements express the same target numeric limit: −1/3.
- Both consider n –> +infty.
- The algebraic expressions inside the cube roots in Lean match the informal expressions: sin n / n + n^3 – n^2 and n^3 + n.

Mismatches / Issues of Formalization Exactness:

- Domain of the index n: The informal text is ambiguous (commonly intended as n ∈ N). The Lean statement fixes n ∈ ℝ. This is a nontrivial choice: an integer–sequence limit and a real–variable limit are closely related but are not syntactically identical statements in a formal system. Because the informal statement did not specify the domain, the formalization made a concrete choice. According to the problem's instructions, the formalization must be an exact faithful formalization; making an unstated choice without documenting it is a divergence.
- Use of rpow vs odd real cube root: The Lean code writes x ^ ((1 : ℝ) / 3) (rpow). In informal calculus, the cube root of x is the unique real cube root defined for all real x (including negative). Mathlib's rpow for non–integer exponents has domain subtleties (generally defined only for x>0, and extended definitions for x=0 or negative may not coincide with the real odd root). The Lean code does not assert positivity hypotheses on the bases, nor does it use an explicit real_cube_root function. Thus the formalization may fail to capture the informal interpretation that the cube root is defined for all reals.
- Division by n at n = 0: The Lean function is written as a total

function on ℝ; the expression Real.sin n / n is problematic at n = 0 in informal arithmetic. While the limit atTop ignores a finite set (n=0), a faithful formalization of a sequence often restricts the domain to n ∈ ℕ (n≥1) to avoid such singularities. The Lean statement does not explicitly restrict n ≠ 0, relying on eventual behavior; that is a modeling choice that should be documented.

6. Conclusion
The Lean theorem is not an exact and fully faithful formalization of the informal Mathematical_Text as written, because it fixes choices that were left ambiguous in the informal text without documenting or ensuring equivalence. Specifically:
– It fixes the domain of n to be ℝ rather than leaving n as an index over ℕ (or explicitly stating the choice). This is a semantic difference in formal systems.
– It uses real exponentiation rpow (x ^ (1/3)) which in Mathlib requires domain considerations (positivity of the base) and so does not directly formalize the informal notion of the real cube root for all real inputs.
– It omits explicit statements ensuring the bases of the rpow are in the domain where rpow is defined (positivity), although these are true for sufficiently large n; a faithful formalization would either restrict the domain (e.g., n≥N) or use a cube–root function that matches the informal meaning.

7. Accuracy Confirmation / Final Judgment
– Because of the domain/type choice for n and the exponentiation–domain subtlety, the Lean statement is Incorrect as a strict exact formalization of the informal problem. These mismatches are material: they change the syntactic and semantic conditions under which the expression is well–defined and thus change the formal statement being proved.

8. Suggested minimal fixes (not requested as part of the verdict, but explanatory):
– If the informal statement intended n ∈ ℕ, the Lean theorem should state Tendsto (λ (k : ℕ) => ...) atTop (nhds (−1/3)) (with appropriate coercions to ℝ inside the expression), or explicitly quantify n : ℝ and remark that the limit holds as a real–variable limit.
– Use a function representing the real cube root (e.g., Real.cbrt or (Real.sqrt? there is cbrt in Mathlib?) or define x ↦ sign x ∗ |x|^(1/3)) to match the odd cube root meaning, or add hypotheses that the bases are eventually positive.

Therefore: the formalization is Incorrect.

F.5 CriticLean ID 210

Example Case 210: CriticLean vs HSA (Full Raw Output)

ID: 210
Gold tag: compile_right_is_right
Source: DeepMath-103k
CriticLean prediction: Correct
HSA prediction: Incorrect

Informal statement (full).

Evaluate the limit \(\lim_{n\to\infty}\int_{−1}^1 \frac{nx^2}{1+n^2x^4} \, dx \) using the Lebesgue Dominated Convergence Theorem.
Prove that the answer is 0.

Lean statement (full).

```
import Mathlib

open Topology Filter Real Complex TopologicalSpace Finset
open scoped BigOperators


theorem Tendsto (f : ℕ → ℝ)
 (h₀ : ∀ n, f n = ∫ x in (−1)..1, n ∗ x ^ 2 / (1 + n ^ 2 ∗ x ^ 4)) :
 Tendsto f atTop (nhds 0) :=
sorry
```

CriticLean output (full raw). is_assistant_correct = Correct

1. Math Assertion Analysis: The mathematical problem asks to evaluate lim_{n→∞} ∫_{−1}^1 (n x^2)/(1 + n^2 x^4) dx and conclude the limit equals 0. Relevant components: a sequence parameter n ∈ ℕ tending to infinity; an integral over the fixed closed interval [−1,1]; the integrand is the real function x ↦ n x^2 / (1 + n^2 x^4); the claim is that the sequence of real numbers given by these integrals converges to 0.

2. Lean Statement Analysis (Proof Ignored): The Lean theorem is

theorem Tendsto (f : Nat –> Real) (h0 : forall n, f n = integral) : Tendsto f atTop (nhds 0) := by sorry

Extracted components: a sequence f : ℕ → ℝ; an assumption h_0 that for every n, f n equals the integral over (−1)..1 of n ∗ x^2 / (1 + n^2 ∗ x^4); the conclusion is Tendsto f atTop (nhds 0), i.e. f(n) → 0 as n → ∞. The interval (−1)..1 encodes the real interval [−1,1], and the integrand matches the mathematical integrand exactly.

3. Comparative Verification: Structure and quantifiers match: both quantify over n ∈ ℕ and claim convergence of the sequence of integrals to 0. The integration interval and the integrand are identical in formula. The Lean statement does not fix f explicitly but assumes equality f n = integral for all n, which is an equivalent way to define the sequence of integrals. Potential minor semantic points:
– The original problem mentions using the Lebesgue Dominated Convergence Theorem; the Lean statement does not mention the mode of integration or measurability/integrability hypotheses explicitly. However, writing ∫ x in (−1)..1, ... in Mathlib denotes an integral over the interval and implicitly requires the integral to be well–defined; this is standard and suffices to state the convergence claim. Whether this is the interval (classical) integral or the measure–theoretic/Lebesgue integral is an implementation detail of Mathlib; in either interpretation the stated numeric limit is the same for this integrand.
– The Lean statement does not separately assert integrability/measurability of the integrand for each n; instead it equates f n with the value of the integral, which in Lean presupposes the integral expression is meaningful. This is a conventional and acceptable way to state the problem in Lean.
– No extra assumptions or conclusions appear in the Lean statement; there are no missing quantifiers or changed domains.

4. Conclusion: The Lean theorem statement is an exact and faithful formalization of the mathematical problem: it encodes the same sequence (n ↦ integral over [−1,1] of n x^2/(1 + n^2 x^4)) and asserts its limit is 0.

5. Accuracy Confirmation: All key elements match: the parameter set (ℕ), the integrand formula, the integration interval [−1,1], and the limit statement Tendsto ... (nhds 0). The only minor notational/semantic caveats are (a) the statement does not spell out integrability/measurability hypotheses (these are implicit in using the integral notation) and (b) the original mention of Lebesgue DCT is a suggested method rather than part of the statement; neither affects the correctness of the formalization. Therefore the formalization is correct and faithful to the mathematical problem.

HSA output (full raw). is_assistant_correct = Incorrect

```
Your detailed CoT analysis:
1. Math Assertion Analysis:
– Extracted from Mathematical_Text (informal problem):
 {
  "variables": ["n (natural number)", "x (real number)"],
  "atomic_constraints": [
   "n → ∞ (limit as n tends to infinity)",
   "integration variable x ranges over interval [-1,1]",
   "integrand: n * x^2 / (1 + n^2 * x^4) is defined for
      all x in [-1,1] and all n ∈ ℕ",
   "integral is the Lebesgue integral (explicit from
      statement: 'using the Lebesgue Dominated
      Convergence Theorem')"
  ],
```

"quantified_constraints": [
"Implicit: for each n ∈ ℕ, the map x ↦ n x^2/(1+n^2 x^4) is (Lebesgue) measurable and integrable on [-1,1]"
],
"structural_assumptions": [
"Integration is over the real interval [-1,1] with respect to Lebesgue measure",
"Sequence of integrals a_n = ∫_{-1}^1 (n x^2)/(1+n^2 x^4) dx",
"Consider lim_{n→∞} a_n; claim: limit equals 0"
],
"target_statement": {
"type": "limit of sequence",
"claim": "lim_{n→∞} ∫_{-1}^1 n x^2/(1 + n^2 x^4) dx = 0"
},
"implicit_or_derived_requirements": [
"That 'integral' means Lebesgue integral (since the problem explicitly asks to use LDCT), and so the statement concerns Lebesgue integrals/measure theory context"
],
"ambiguities": [
"None significant in the informal statement: the use of 'Lebesgue Dominated Convergence Theorem' pins down Lebesgue integral"
]
}

2. Lean Statement Analysis (Proof Ignored):
– Extracted from Lean4Code (formal statement):

{
"declaration": "theorem Tendsto (f : ℕ → ℝ) (h₀ : ∀ n, f n = ∫ x in (-1)..1, n * x ^ 2 / (1 + n ^ 2 * x ^ 4)) : Tendsto f atTop (nhds 0)",
"variables": ["f : ℕ → ℝ", "n : ℕ (bound in quantifier ∀ n)", "x : ℝ (bound inside integral)"],
"atomic_constraints": [
"For all n, f n equals the integral ∫ x in (-1)..1 of (n * x^2) / (1 + n^2 * x^4)",
"Conclusion: Tendsto f atTop (nhds 0) -- i.e., f(n) → 0 as n → ∞"
],
"quantified_constraints": [
"Universal quantifier over n: ∀ n, f n = integral expression"
],
"structural_assumptions": [
"Integral is written as '∫ x in (-1)..1, ...' which in mathlib notation typically denotes an interval/set integral (the code does not explicitly state 'Lebesgue' or 'measure_theory' integral).",
"No extra hypotheses are stated (e.g., no explicit measurability or integrability assumptions are given in the theorem statement; these would typically be implicit if Lean accepts the integral expression)"
],
"target_statement": {
"type": "limit of sequence",
"claim": "The sequence f tends to 0 at infinity (in ℝ)."
},
"implicit_or_derived_requirements": [
"The expression '∫ x in (-1)..1, ...' must be well-typed in Lean: this forces some notion of integral to be applicable (interval integral or measure integral), but the theorem text does not explicitly fix that as Lebesgue measure",
"Because f : ℕ → ℝ and the RHS is a real number, the integral is intended to be a real-valued integral over the real interval"
],
"ambiguities": [
"Which integral is used in Lean? The notation '∫ x in a..b, f x' is ambiguous between the interval integral (interval_integral) and the measure-theoretic integral over the set Icc a b (measure_theory.integral). The code does not explicitly import measure_theory or state 'with respect to volume/Lebesgue measure'."
]
}

3. Comparative Verification:
– Points of agreement:
  – Both statements assert that the sequence of integrals a_n = ∫_{−1}^1 n x^2/(1 + n^2 x^4) tends to 0 as n → ∞.
  – Variables and the integrand expression match syntactically and semantically (n, x, x ∈ ℝ, n ∈ ℕ, integrand n ∗ x^2/(1 + n^2 x^4)).
  – The Lean formalization packages the sequence of integrals as a function f : ℕ → ℝ with the equality hypothesis ∀ n, f n = integral. This corresponds exactly to the informal object 'sequence of integrals'.
– Points of mismatch / concerns:
  – Explicit mention of Lebesgue integral: The informal problem explicitly requires using the Lebesgue Dominated Convergence Theorem and hence is phrased in terms of Lebesgue integrals on [−1,1]. The Lean statement does not explicitly state that the integral is the Lebesgue (measure−theoretic) integral. The notation used (∫ x in (−1)..1, ...) in Mathlib can denote an interval integral; without additional imports or type annotations the theorem as written does not make the Lebesgue measure explicit.
  – Measurability / integrability hypotheses: The informal problem relies on Lebesgue measurability/integrability, but the Lean statement contains no explicit side−conditions (e.g., 'measurable' or 'Integrable'). While these may be inferred/required by the integral notation to make the expression well−typed, they are not explicitly stated and so the formalization omits an explicit Lebesgue−measure context that was part of the original problem statement (which named LDCT).
  – The original exercise explicitly instructs to use Lebesgue Dominated Convergence Theorem; the Lean theorem is silent about the method or context (it only states the numeric limit). A faithful formalization of the problem as given should reflect the Lebesgue/measure−theoretic context or at least be clear that the integral is the Lebesgue integral over [−1,1].

4. Conclusion:
– The Lean statement matches the computational target (the numerical limit 0 for the sequence of integrals) and the integrand, domains and limit structure are represented correctly. However it fails to explicitly formalize the integral as a Lebesgue (measure−theoretic) integral and does not make explicit the measure−theoretic hypotheses (measurability/integrability) that are central to the original statement which explicitly references the Lebesgue Dominated Convergence Theorem.
– Because the mathematical text explicitly specified the Lebesgue setting, and the Lean statement does not unambiguously encode that setting (the integral notation used can represent a different integral in Mathlib), the formalization is not an exact and fully faithful formalization of the given problem as stated.

5. Accuracy Confirmation / Discrepancies:
– Discrepancies (how each affects correctness):
  1) Missing explicit Lebesgue/measure−theoretic specification: The original problem explicitly asks to use the Lebesgue Dominated Convergence Theorem; the Lean statement does not state the integral is the Lebesgue integral with respect to Lebesgue measure. In Mathlib, the notation ∫ x in a..b, ... is ambiguous without context and often used for the interval integral (not necessarily the measure−theoretic Lebesgue integral). This is a semantic mismatch because the original problem's context is measure theory; a faithful formalization should make that explicit.
  2) Missing explicit measurability/integrability hypotheses: The mathematical text implicitly relies on measurability/integrability to apply LDCT; the Lean theorem includes no explicit hypotheses about measurability/integrability. While Lean might infer or require such facts to typecheck, they are not visible in the statement; thus the formal statement omits conditions that materially change the interpretation (requirement: removing those would change admissible solutions). The omission weakens faithfulness to the specified Lebesgue context.

Overall judgment: The Lean theorem correctly encodes the numerical limit claim and the integrand, but it is not fully faithful to the written problem because it fails to unambiguously assert the Lebesgue/measure−theoretic integral context and the related measurability/integrability hypotheses. Therefore the

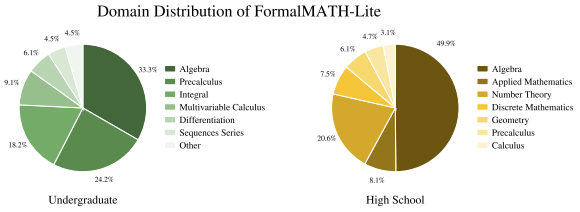


Figure 8: Problem distribution of FormalMATH Subset.

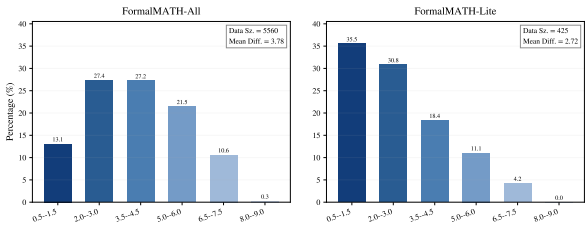


Figure 9: Difficulties distribution of FormalMATH Subset.

```
formalization is Incorrect as a strict, exact
formalization of the mathematical problem as given.
```

G Fail Case in NL-augmented CoT

Example: naive CoT outperforms NL-augmented CoT and vanilla strategies in formal theorem proving.

Naive CoT (Success):

```
theorem algebra_68653_cot : ¬ ∃ x y : ℤ, x^3 + y^4 = 7 := by
  -- Assume there are integers x and y such that x^3 + y^4 = 7.
  intro h
  -- Extract the integers x and y from the assumption.
  rcases h with ⟨x, y, hxy⟩
  -- Consider the equation modulo 13.
  have hxy_mod : (x^3 + y^4) % 13 = 7 % 13 := by rw [hxy]
  -- Calculate the possible values of x^3 modulo 13.
  have x3_mod : x^3 % 13 = 0 ∨ x^3 % 13 = 1 ∨ x^3 % 13 = 5 ∨
      x^3 % 13 = 8 ∨ x^3 % 13 = 12 := by
   have : x % 13 = 0 ∨ x % 13 = 1 ∨ x % 13 = 2 ∨ x % 13 = 3 ∨
      x % 13 = 4 ∨ x % 13 = 5 ∨ x % 13 = 6 ∨ x % 13 = 7 ∨ x
      % 13 = 8 ∨ x % 13 = 9 ∨ x % 13 = 10 ∨ x % 13 = 11 ∨ x
      % 13 = 12 := by omega
   rcases this with (h | h | h | h | h | h | h | h | h | h | h | h | h) <;>
      simp [h, pow_succ, Int.mul_emod]
  -- Calculate the possible values of y^4 modulo 13.
  have y4_mod : y^4 % 13 = 0 ∨ y^4 % 13 = 1 ∨ y^4 % 13 = 3 ∨
      y^4 % 13 = 9 := by
   have : y % 13 = 0 ∨ y % 13 = 1 ∨ y % 13 = 2 ∨ y % 13 = 3 ∨
      y % 13 = 4 ∨ y % 13 = 5 ∨ y % 13 = 6 ∨ y % 13 = 7 ∨ y
      % 13 = 8 ∨ y % 13 = 9 ∨ y % 13 = 10 ∨ y % 13 = 11 ∨ y
      % 13 = 12 := by omega
   rcases this with (h | h | h | h | h | h | h | h | h | h | h | h | h) <;>
      simp [h, pow_succ, Int.mul_emod]
  -- Check that x^3 + y^4 cannot be congruent to 7 modulo 13.
  rcases x3_mod with (h1 | h1 | h1 | h1 | h1) <;> rcases y4_mod
      with (h2 | h2 | h2 | h2) <;> simp [h1, h2, Int.add_emod] at
      hxy_mod <;> omega
```

NL-augmented CoT (Not Success):

```
theorem algebra_68653_nat : ¬ ∃ x y : ℤ, x^3 + y^4 = 7 := by
    /-
  We aim to show that there do not exist integers \( x \) and \( y \)
      such that \( x^3 + y^4 = 7 \). Suppose, for the sake of
      contradiction, that such integers exist. We would then have
      \( x^3 + y^4 = 7 \). We analyze the possible values of \( x^3
      \mod 7 \) and \( y^4 \mod 7 \) to find a contradiction.
  1. **Lemma:** For any integer \( x \), \( x^3 \mod 7 \) can only
      be 0, 1, or −1.
    – Proof: By examining all possible values of \( x \mod 7 \), we
      find that \( x^3 \mod 7 \) is either 0, 1, or −1.
  2. **Lemma:** For any integer \( y \), \( y^4 \mod 7 \) can only
      be 0, 1, or 2.
    – Proof: By examining all possible values of \( y \mod 7 \), we
      find that \( y^4 \mod 7 \) is either 0, 1, or 2.
  3. **Combining the Lemmas:** If \( x^3 + y^4 = 7 \), then \( x^3
      \mod 7 \) and \( y^4 \mod 7 \) must sum to 7 modulo 7.
      However, the possible values of \( x^3 \mod 7 \) are 0, 1, or
      −1, and the possible values of \( y^4 \mod 7 \) are 0, 1, or 2.
      None of these combinations can sum to 7 modulo 7,
      leading to a contradiction.
  Thus, we conclude that there do not exist integers \( x \) and \( y
      \) such that \( x^3 + y^4 = 7 \).
  -/
  -- Assume for contradiction that there exist integers x and y
      such that x^3 + y^4 = 7.
  intro h
  obtain ⟨x, y, hxy⟩ := h
  -- Analyze the possible values of x^3 modulo 7.
  have hx : x^3 % 7 = 0 ∨ x^3 % 7 = 1 ∨ x^3 % 7 = 6 := by
   have hx : x % 7 = 0 ∨ x % 7 = 1 ∨ x % 7 = 2 ∨ x % 7 = 3 ∨ x
      % 7 = 4 ∨ x % 7 = 5 ∨ x % 7 = 6 := by
    omega
   rcases hx with (h | h | h | h | h | h | h) <;>
    simp [h, pow_three, Int.mul_emod, Int.add_emod]
  -- Analyze the possible values of y^4 modulo 7.
  have hy : y^4 % 7 = 0 ∨ y^4 % 7 = 1 ∨ y^4 % 7 = 2 := by
   have hy : y % 7 = 0 ∨ y % 7 = 1 ∨ y % 7 = 2 ∨ y % 7 = 3 ∨ y
      % 7 = 4 ∨ y % 7 = 5 ∨ y % 7 = 6 := by
    omega
   rcases hy with (h | h | h | h | h | h | h) <;>
    simp [h, pow_four, Int.mul_emod, Int.add_emod]
  -- Combine the results to find a contradiction.
  have h7 : (x^3 + y^4) % 7 ≠ 7 := by
   rcases hx with (h | h | h) <;>
    rcases hy with (h' | h' | h') <;>
     simp [h, h', Int.add_emod, Int.mul_emod]
  -- Conclude the proof by contradiction.
  omega
```

Vanilla (Not Success):

```
theorem algebra_68653_vanilla : ¬ ∃ x y : ℤ, x^3 + y^4 = 7 :=
      by
  intro h; rcases h with ⟨x, y, hxy⟩; have hx : x^3 ≥ 0 := by
      positivity; have hy : y^4 ≥ 0 := by positivity
  linarith
```

H Smaller and Easier Subset for FormalMATH

We construct a smaller and easier subset as a development-oriented subset of FormalMATH-All (see Figure 9 and Figure 8). The goal is not only to reduce evaluation cost, but to retain the main sources of variation that matter during model iteration, including topic coverage, education level, and representative error patterns. Concretely, we select the Subset by following the following strategy: we preserve coverage over the major mathematical domains appearing in the full benchmark, while biasing the subset toward easier problems so that it is cheap enough for routine ablations, checkpoint selection, debugging, and case studies. The resulting split is therefore intentionally easier than the full benchmark, but still domain-diverse. This design choice is reflected in the difficulty statistics. Using GPT-5-mini difficulty estimates, FormalMATH subset has mean difficulty 2.72, compared with 3.78 for FormalMATH-All. Moreover, 66.4% of problems fall in the 0.5–3.0 difficulty

range, whereas the corresponding proportion in the full benchmark is only 40.5%. Conversely, only 15.3% of problems have difficulty at least 5.0, compared with 32.3% in the full dataset. Thus, the subset is not a uniformly random subset of the benchmark; it is a deliberately easier split designed for rapid development-time evaluation. At the same time, the split is not restricted to a single school level or topic family. The subset contains 425 problems in total, including 359 high-school-level problems and 66 undergraduate-level problems. The undergraduate portion is drawn from sources such as U-Math, HardMath, and DEMIMATH, ensuring that the development split still contains nontrivial undergraduate material rather than collapsing entirely to olympiad-style high-school problems. On the high-school portion, the primary domain distribution remains broad: Algebra accounts for 49.9%, Number Theory for 20.6%, Applied Mathematics for 8.1%, Discrete Mathematics for 7.5%, Geometry for 6.1%, Precalculus for 4.7%, and Calculus for 3.1%. This confirms that the FormalMATH subset is, on average, easier to use, while still covering the major top-level domains present in the benchmark. Therefore, the FormalMATH subset should be understood as a coverage-preserving but difficulty-reduced benchmark slice. The full benchmark remains the primary testbed for final reporting, while the subset serves as a fast and diagnostically useful proxy for development-time model comparison and qualitative analysis. This follows the broader benchmark-design principle of retaining meaningful variation while substantially reducing evaluation cost (Hardt, 2025).

I LLM Usage Disclosure.

In this work, we use large language models (LLMs) as components of the autoformalization pipeline and theorem-proving evaluation. We also use LLM for minor paper polishing.