CriticLean: Critic-Guided Reinforcement Learning for Mathematical Formalization

Anonymous ACL submission

Abstract

Translating natural language mathematical statements into formal, executable code is a fundamental challenge in automated theorem proving. While prior work has focused on generation and compilation success, little attention has been paid to the *critic phase*—the evaluation of whether generated formalizations truly capture the semantic intent of the original problem. In this paper, we introduce **CriticLean**, a novel critic-guided reinforcement learning framework that elevates the role of the critic from a passive validator to an active learning component. Specifically, first, we propose the **CriticLeanGPT**, trained via supervised fine-tuning and reinforcement learning, to rigorously assess the semantic fidelity of Lean 4 formalizations. Then, we introduce **CriticLean-Bench**, a benchmark designed to measure models' ability to distinguish semantically correct from incorrect formalizations, and demonstrate that our trained CriticLeanGPT models can significantly outperform strong open- and closed-source baselines. Building on the CriticLean framework, we construct **FineLeanCorpus**, a dataset comprising over 509K problems that exhibits rich domain diversity, broad difficulty coverage, and high correctness based on human evaluation. Overall, our findings highlight that optimizing the critic phase is essential for producing reliable formalizations and we hope our CriticLean will provide valuable insights for future advances in formal mathematical reasoning.

1 Introduction

The formalization of mathematical statements (Yang et al., 2024) is a critical task in modern mathematical computation, particularly in the context of theorem provers like Lean 4 (Leanprover Community, 2023). The translation of natural language mathematical problems into formal, executable code remains a significant challenge, as it requires not only syntactical accuracy but also a deep understanding of the problem's semantics (Scholze, 2022; Tao, 2023). Existing approaches have shown progress, but they often face limitations in accuracy, especially in the context of complex, high-level problems that involve sophisticated mathematical reasoning (Zheng et al., 2021; Azerbayev et al., 2023; Welleck et al., 2021, 2022; Lewkowycz et al., 2022). In contrast to existing works, we argue that the *critic* phase—the step where the semantic correctness of generated formalizations is evaluated—is not only underexplored but also fundamentally essential to the success of mathematical autoformalization.

Therefore, in this paper, we systematically investigate and optimize the critic component and introduce **CriticLean**, a comprehensive framework that places the critic model at the center of the formalization pipeline. Unlike prior work that primarily focuses on generation quality or compiler validity, CriticLean introduces the reinforcement learning-based **CriticLeanGPT** models explicitly trained to evaluate whether the Lean 4 output truly reflects the intent of the original mathematical statement., and we present a full methodology for training, and evaluating CriticLeanGPT model. Specifically, as shown in Figure 1, for each natural language statement, we apply the autoformalization iteratively based on the feedback from the Lean compiler and the CriticLeanGPT models, which is trained to critically assess whether a generated formalization accurately represents the semantics of the original mathematical statement. In each iteration, the feedback provides valuable signals that drive an iterative refinement process, further improving the quality of the final Lean code output.

Additionally, we present **CriticLeanBench**, a benchmark designed to evaluate the performance of CriticLeanGPT models, which contain 500 natural language and Lean 4 language statement pairs (i.e., 250 correct and 250 incorrect pairs).

Through extensive experiments, we demonstrate that our trained CriticLeanGPT models outperform the SOTA open-source models (Bai et al., 2023; Grattafiori et al., 2024; Guo et al., 2025a) and many closed-source API models (Google, 2023; OpenAI, 2023; Guo et al., 2025b) greatly.

Furthermore, building upon our critic-centric CriticLean pipeline, we propose the high-quality open-source Lean 4 statement dataset **FineLean-Corpus**, comprising 509,356 fully verified entries. When compared to the previous related datasets (e.g., LeanWorkBook (Ying et al., 2025)), FineLeanCorpus is distinguished by its diversity in mathematical domains, difficulty distribution, and strict semantic validation via critical feedback loops. Notably, its difficulty distribution and targeted domain enrichment create a more structurally balanced training environment, mitigating overfitting and transforming sparse topics into well-supported sub-domains. Furthermore, to foster research into the upper echelons of mathematical reasoning, we have curated the specialized subset called FineLeanCorpus-Diamond, comprising over 36,000 high-difficulty problems.

2 Related Works

2.1 Autoformalization

Autoformalization (Szegedy, 2020; Wu et al., 2022; Yu et al., 2025c) refers to the process by which AI systems parse natural language (NL) contents and translate them into machine-verifiable formal representations, such as those in theorem provers like Lean4 (Moura and Ullrich, 2021) or Isabelle (Nipkow et al., 2002). Recent advances leverage large language models (LLMs) (Zhang et al., 2024a) to tackle this problem through two primary paradigms: (1) In-context learning (Wei et al., 2022), where models utilize annotated examples (Wu et al., 2022; Liu et al., 2023; Lu et al., 2024) to generate formalizations without explicit fine-tuning, (2) Supervised fine-tuning (e.g., (Lin et al., 2025; Xin et al., 2024a; Yu et al., 2025b)), which adapts general-purpose LLMs into domain-specific autoformalization experts. To assess correctness, prior works (Lin et al., 2025; Xin et al., 2024a; Yu et al., 2025b) employ LLM-as-judge (Zheng et al., 2023) to verify semantic alignment between formal and informal statements. We advance this by training the first open-sourced, domain-specific light LLM on top of Qwen (Team, 2025a) family for critiquing Lean4 statement alignment via reinforcement learning (Shao et al., 2024b), enhancing both critique capability and formalization robustness.

2.2 RL for LLM Reasoning

The exploration of complex reasoning capabilities in Large Language Models (LLMs) has achieved significant advancements (Liu et al., 2024b, 2025a; Wang et al., 2025b; Huang et al., 2025; Liu et al., 2024a; He et al., 2025a), with Reinforcement Learning (RL) establishing itself as a critical paradigm for transcending the constraints inherent to Supervised Fine-Tuning (SFT) (Yu et al., 2025a; Yue et al., 2025b; Wang et al., 2025c; Liu et al., 2025b, 2024a). Notable methodologies, including GRPO (Shao et al., 2024b; Guo et al., 2025a), DAPO (Yu et al., 2025a) have demonstrated substantial gains in mathematical reasoning and intricate problem-solving domains. However, the underlying mechanisms by which RL enhances reasoning capabilities remain insufficiently characterized. Empirical analyses increasingly indicate that RL primarily functions to activate, refine, or optimize the sampling of latent reasoning competencies rather than generating entirely novel cognitive frameworks de novo. For example, Yue et al. (2025a) interrogate whether current reinforcement learning frameworks incorporating verifiable rewards (RLVR) genuinely expand the frontier of reasoning performance or merely enhance the efficiency of retrieving pre-existing solutions.

3 CriticLeanGPT

3.1 CriticLeanBench

3.1.1 Overview of CriticLeanBench

CriticLeanBench aims to evaluate the critical reasoning of LLMs in key aspects such as translating natural language mathematical statements into formally verified theorem declarations in Lean 4, including critique and correction. By integrating these core dimensions, CriticLeanBench can comprehensively measure the performance of models in Formalization tasks. In this section, we will elaborate on the construction principles and processes of CriticLeanBench. More details about CriticLeanBench see Appendix C. the construction process is shown in Figure 2.

CriticLeanBench is constructed following the following principles: (1) It covers various types of errors, aiming to thoroughly evaluate the comprehensive capabilities of LLMs in capturing the semantic intent of formalized statements, using

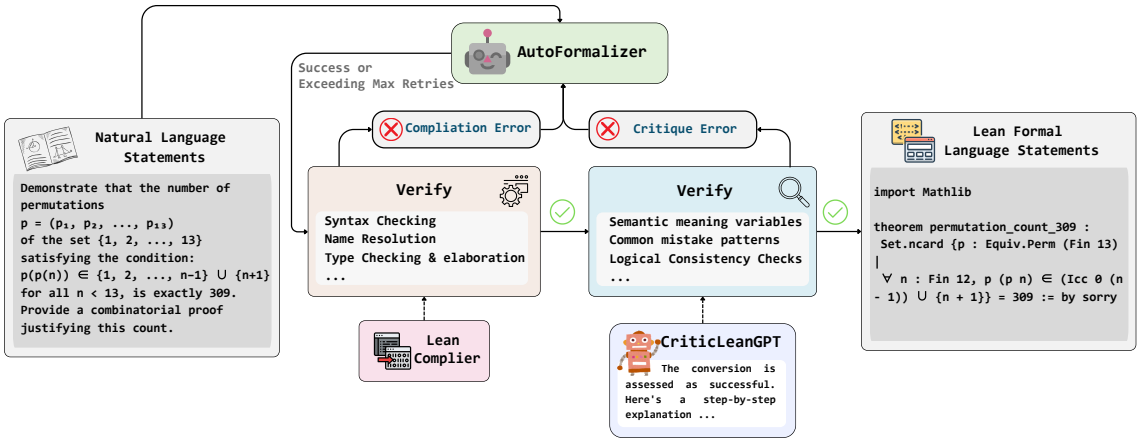


Figure 1: Illustration of CriticLean framework based on Critic-Guided Reinforcement Learning for Mathematics Autoformalization.

Template L for systematic assessment; (2) It incorporates diverse data sources to enhance the diversity and representativeness of evaluation data; and (3) It ensures the reliability and validity of the evaluation benchmark through a combination of expert review and automated verification. See Appendix B for detailed sample selection criteria and comprehensive error type coverage.

3.1.2 Automatically verified

This section outlines the methodology employed to apply the Automatic Validation Filter , guided by predefined criteria for compiler and model-based verification.

Data Collection We selected Math Statements from data sources (Perez et al., 2025; Mahdavi et al., 2025a; Yu et al., 2025a; He et al., 2025c; LI et al., 2024; Online Math Contest; AI Research Group of TAL Education Group), such as Omni-MATH (Gao et al., 2024), AIME (Zheng et al., 2021), U-MATH (Chernyshev et al., 2024), DEMI-MathAnalysis (Demidovich, 1964), HARDMath (Fan et al.), OlympiadBench (He et al., 2024), and BlueMO (Zhang et al., 2024b), along with their corresponding Lean 4 Statements from public dataset (Yu et al., 2025b).

Compiler Verified We submitted the Lean 4 statements from the above dataset to the Lean 4 compiler. If the compilation succeeded, the results were passed to the DeepSeek R1 (Guo et al., 2025a) model for further processing.

- **Compile false:** For statements that failed to compile, we randomly sampled 50 entries and retained the compiler feedback messages, which were included as part of our CriticLean-Bench benchmark.

LLM Verified For the data that has passed compilation, we utilize a template L to process each sample's Math Statement and its corresponding Lean 4 Code Statement through the DeepSeek R1 large language model. The model is tasked with determining whether the Lean 4 Code Statement is consistent with the Math Statement, producing for each sample a tag indicating consistency or inconsistency along with a reasoning statement. The results are then submitted to human reviewers for further validation.

3.1.3 Human Validation Filter

This section outlines the methodology employed to apply the Human Validation Filter, guided by predefined criteria(detailed in Appendix H).

We categorize the data into two groups based on whether the **Lean-compiled** autoformalization output semantically aligns with the refined statement. Therefore, a Lean-compiled autoformalization falls into the following:

- **Human Check right:** If the autoformalized statement *both* compiles *and* accurately captures the mathematical meaning, logic, and intended behavior of the original problem, it satisfies the semantic consistency criteria.
- **Human Check false:** If the autoformalized statement compiles but *fails* to accurately represent the original mathematical problem's semantics. It violates one or more of the semantic consistency criteria, despite being syntactically valid Lean code. This often happens

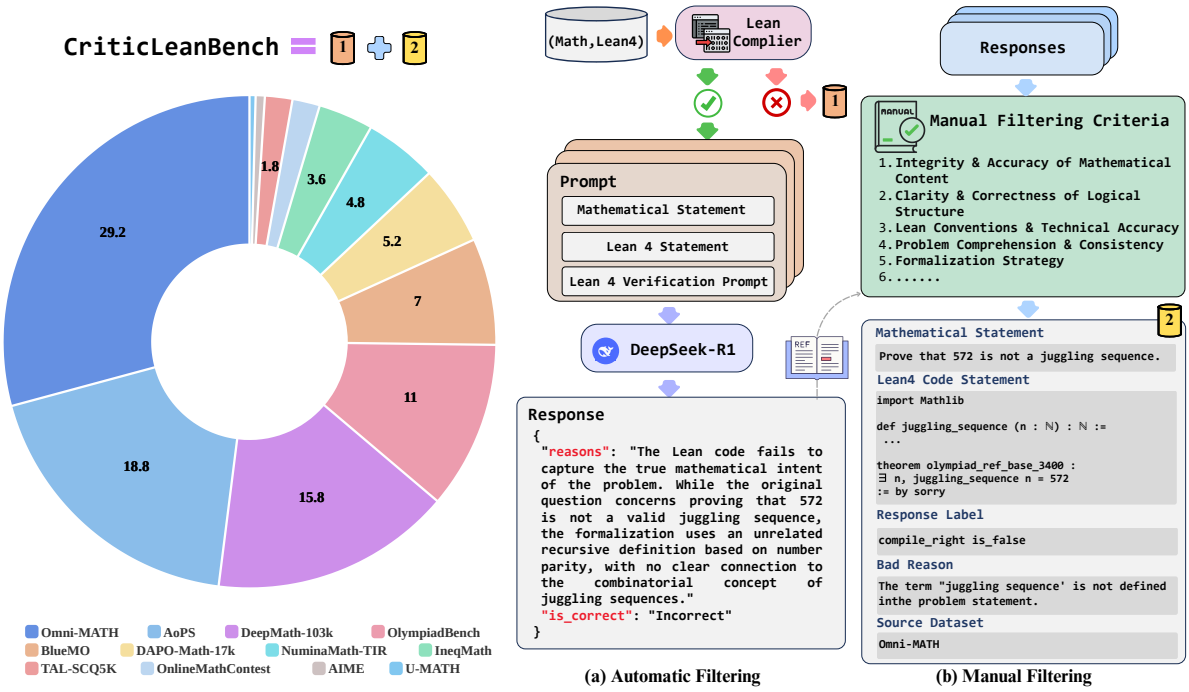


Figure 2: An overview for the CriticLeanBench construction.

when there's a mismatch between the code's logic and the intended mathematical meaning.

3.1.4 Data Statistics

Statistics	Number
#Problems	500
Correct Pairs	250
- human check	250
Incorrect Pairs	250
- human/compiler check	200/50
Question Tokens Length	
- max/min/avg	1,583/495/700.94

Table 1: Dataset statistics of CriticLeanBench.

Table 1 shows CriticLeanBench contains 500 problems (250 correct/incorrect Q/GT pairs). Questions range from 495–1,583 tokens (Qwen2.5 tokenizer), averaging 700.94 tokens. This length makes it a complex benchmark requiring models to process lengthy inputs compared to standard datasets.

4 CriticLeanInstruct

To enhance the CriticLeanGPT model's efficacy in critically evaluating the transformation of mathematical statements into Lean code, we **construct** a comprehensive training dataset comprising 48,000 samples. This dataset integrates three complementary components: (1) human-annotated seed data, (2) augmented samples from formalized mathematics corpora, and (3) domain-specific code and math data. More details about seed data and data augmentation see Appendix D.

To significantly broaden the CriticLeanGPT model's knowledge coverage, we also integrates three times additional code and math (Hugging Face, 2025) datasets, enabling it to more comprehensively understand mathematical concepts and Lean code structures.

Specifically, the Seed Data, with a 1:3 mix of math and code, is termed **CriticLeanInstruct(16K)**. When combined with data augmentation while retaining the same ratio, it forms the full **CriticLeanInstruct** dataset.

4.1 Training Paradigm

Supervised fine-tuning (SFT). These models are instruction-tuned versions based on their respective pre-trained Qwen2.5 checkpoints, with a focus on improving their ability to interpret and formalize complex mathematical statements expressed in natural language. The CriticLeanInstruct 4 dataset includes Critic data consisting of mathematical statements converted into formally verified theorem declarations in Lean 4, along with three times as much code and mathematics data for SFT

(Supervised Fine-Tuning). We used the LLaMA-Factory (Zheng et al., 2024) framework to facilitate the fine-tuning process and optimize model performance.

Reinforcement Learning Optimization (RL). The recent success of R1-style methods has demonstrated the effectiveness of online RL using discrete, rule-based rewards (Shao et al., 2024a). In our pipeline, Qwen2.5 series and Qwen3 (Team, 2025a) series are further refined using reinforcement learning signals derived from both format validation of the generated critic data and consistency checking between model predictions and expert-labeled ground truth (GT) labels . Specifically, the RL training data consists of 4,000 Seed Data D.1, where each example transforms a mathematical statement into a corresponding formal proof in Lean 4. Based on this dataset, we apply a rule-based RL approach to optimize the model's capability in judgment reasoning. More specifically, we mainly utilize the GRPO (Shao et al., 2024b) algorithm within the VeRL (Sheng et al., 2024) reinforcement learning framework, whose optimization objective is:

$$J_{\text{online}}(\pi_\theta; \mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D},\{y_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(y|x)} \left[\frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)} A_i, \text{clip}\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}, 1-\epsilon, 1+\epsilon \right) A_i \right) - \beta D_{\text{KL}}(\pi_\theta || \pi_{\text{ref}}) \right] \quad (1)$$

where G is group size, and A_i is advantage. The reward function is designed as follows[1]:

$$r_{\text{accuracy}} = \begin{cases} 1, & \text{if judgement} = \text{label} \\ 0, & \text{if judgement} \neq \text{label} \end{cases} \quad (2)$$

$$r_{\text{format}} = \begin{cases} 1, & \text{if format is right} \\ 0, & \text{if format is wrong} \end{cases} \quad (3)$$

$$r_{\text{final}} = \min(r_{\text{accuracy}}, r_{\text{format}}) \quad (4)$$

5 Experiments

5.1 Experimental Setup

Baseline Models. For the closed-sourced API models, we select the following models: Claude35_Sonnet2 (Anthropic, 2024), Doubao-1.5-pro-32k (Guo et al., 2025b), Gemini 2.5 Pro (Team et al., 2023), GPT-4o-2024-11-20 (OpenAI, 2023). We select a group of the most advanced open-source LLMs to serve as critic models for evaluation, which includes various reasoning models (DeepSeek-R1 (Guo et al., 2025a), QwQ-32B (Team, 2025b), Qwen3-8B (Team, 2025a), Qwen3-14B (Team, 2025a), Qwen3-32B (Team, 2025a)), DeepSeek-Prover models (Xin et al., 2024b) (DeepSeek-Prover-V1.5-RL, DeepSeek-Prover-V1.5-SFT), Llama-3.3-70B-Instruct (Grattafiori et al., 2024) and several Qwen models (Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), Qwen2.5-Coder-32B-Instruct (Hui et al., 2024), Qwen2.5-7B-Instruct (Team, 2024), Qwen2.5-14B-Instruct (Team, 2024) and Qwen2.5-32B-Instruct (Team, 2024)).

CriticLeanGPT Models. We further evaluate three variants from the Qwen2.5 series and Qwen3 series. These are instruction-based models fine-tuned specifically on either the CriticLeanInstruct 4 dataset or the RL-based clean critic Seed Data D.1. All open-source models are inferred using the vLLM (Kwon et al., 2023) framework with default inference parameters.

5.2 Main Results

5.2.1 Evaluation of Critic Capability

Model	ACC	TPR	FPR	TNR	FNR
SOTA LLMs					
Gemini-2.5-Pro	89.2	95.6	4.4	82.8	17.2
QwQ-32B	86.4	93.6	6.4	79.2	20.8
Qwen3-32B	85.6	**96.0**	**4.0**	75.2	24.8
Qwen3-235B-A22B	84.8	90.4	9.6	79.2	20.8
DeepSeek-R1	84.0	90.8	9.2	77.2	22.8
Qwen3-14B	83.6	92.4	7.6	74.8	25.2
Qwen3-8B	79.8	94.4	5.6	65.2	34.8
Doubao-1.5-pro-32k	78.4	95.2	4.8	61.6	38.4
Claude35-Sonnet	74.2	97.2	2.8	51.2	48.8
Qwen2.5-32B-Instruct	73.0	91.6	8.4	54.4	45.6
Qwen2.5-Coder-32B-Instruct	71.6	91.6	8.4	51.6	48.4
Llama-3.3-70B-Instruct	68.2	95.2	4.8	41.2	58.8
GPT-4o-2024-11-20	67.8	95.6	4.4	40.0	60.0
Qwen2.5-14B-Instruct	66.6	80.4	19.6	52.8	47.2
Qwen2.5-Coder-7B-Instruct	65.4	88.4	11.6	42.4	57.6
Qwen2.5-7B-Instruct	60.8	89.6	10.4	32.0	68.0
DeepSeek-Prover-V1.5-SFT	52.4	78.8	21.2	26.0	74.0
DeepSeek-Prover-V1.5-RL	50.0	76.4	23.6	23.6	76.4
CriticLeanGPT (Ours)					
Qwen3-8B-RL	79.8	90.0	10.0	72.0	28.0
Qwen3-14B-RL	84.8	91.6	8.4	78.0	22.0
Qwen3-32B-RL	**87.0**	88.4	11.6	85.6	14.4
Qwen2.5-7B-Instruct-RL	68.6	85.6	14.4	51.6	48.4
Qwen2.5-14B-Instruct-RL	69.4	85.2	14.8	53.6	46.4
Qwen2.5-32B-Instruct-RL	72.0	60.4	39.6	**83.6**	**16.4**
Qwen2.5-7B-Instruct-SFT	69.8	94.4	5.6	45.2	54.8
Qwen2.5-14B-Instruct-SFT	70.6	83.6	16.4	57.6	42.4
Qwen2.5-32B-Instruct-SFT	76.2	85.2	14.8	67.2	32.8
Qwen2.5-7B-Instruct-SFT-RL	68.2	90.4	9.6	46.0	54.0
Qwen2.5-14B-Instruct-SFT-RL	74.6	81.6	18.4	67.6	32.4
Qwen2.5-32B-Instruct-SFT-RL	78.6	88.0	12.0	69.2	30.8

Table 2: **Performance on CriticLeanBench.** The best, the second-best and the third-best scores for each indicator are shown in box, **bold** and underlined, respectively.

As indicated in Table 2, the experimental results clearly demonstrate the effectiveness of the CriticLeanGPT models trained on our CriticLeanInstruct, in converting natural language mathemati-

[1]We do not include a length penalty in rewards to encourage longer thinking.

cal statements into Lean 4 formal theorem declarations. Within the CriticLeanBench benchmark, our CriticLeanGPT models trained via supervised fine-tuning (SFT) and reinforcement learning (RL), along with their enhanced variants—outperform a range of closed-source API models, open-source models, and baseline models, highlighting distinct advantages. These outcomes yield several key insights: (1) Reasoning models excel in critical tasks, with Gemini 2.5 Pro, QwQ-32B, Qwen3-32B, and DeepSeek-R1 all attaining scores above 80. When compared to baseline models, our Qwen3-32B-RL model, optimized through RL, achieves a strong accuracy level, underscoring the efficacy of both our training methodology and dataset. (3) Our innovative mixed SFT strategy substantially boosts the performance of the Qwen2.5 family, with notable improvements observed across the 7B, 14B, and 32B models. (4) Additionally, SFT and RL significantly strengthen the models' capacity to identify erroneous samples, as evidenced by higher true negative rates (TNR) and lower false negative rates (FNR)—a critical enhancement for accurate detection of incorrect formalizations, which is indispensable for effective critical tasks.

5.3 Ablation Study

Model	ACC	TPR	FPR	TNR	FNR
7B Size Models					
Qwen2.5-7B-Instruct	60.8	89.6	10.4	32.0	68.0
Qwen2.5-7B-Instruct-SFT(Critic Only)	64.0	70.8	29.2	57.2	42.8
Qwen2.5-7B-Instruct-SFT	69.8	94.4	5.6	45.2	54.8
14B Size Models					
Qwen2.5-14B-Instruct	66.6	80.4	19.6	52.8	47.2
Qwen2.5-14B-Instruct-SFT(Critic Only)	67.4	80.8	19.2	54.0	46.0
Qwen2.5-14B-Instruct-SFT	70.6	83.6	16.4	57.6	42.4
32B Size Models					
Qwen2.5-32B-Instruct	73.0	91.6	8.4	54.4	45.6
Qwen2.5-32B-Instruct-SFT(Critic Only)	71.0	72.0	28.0	70.0	30.0
Qwen2.5-32B-Instruct-SFT	76.2	85.2	14.8	67.2	32.8

Table 3: **Comparison of model performance under different training strategies:** base model, SFT on Critic data only, and SFT on combined Critic, code, and math data. The best score for each indicator is shown in $\boxed{box}$.

5.3.1 Effect of Reasoning Data

We conduct an ablation study on CriticLeanBench to evaluate the impact of different training strategies. As shown in Table 3, incorporating code and math reasoning data significantly improves performance across all model sizes compared to using Seed Data D.1. Specifically, using the CriticLeanInstruct 4, which samples Critic and non-Critic data at a ratio of 1:3, leads to substantial gains, demonstrating that integrating diverse reasoning tasks enhances the critical reasoning capabilities of the model. This suggests that multi-task learning with math and code data improves the critique abilities of mathematical formalization.

5.3.2 Effect of SFT Dataset Size

Figure 5 highlights a notable parameter-dependent relationship between the size of the SFT dataset and key reasoning performance. Across all model scales, performance improvements are observed through SFT, albeit with varying degrees of fluctuation depending on the training set size. Smaller models (e.g., 7B) exhibit more pronounced gains as the dataset expands, whereas larger models (e.g., 32B) demonstrate a less consistent trend, with marginal improvements at lower data volumes but substantial gains at higher data volumes. These findings align with prior studies (Muennighoff et al., 2025; Zhou et al., 2023), underscoring the interplay between model capacity, data scale, and performance optimization in SFT scenarios. The results emphasize the need for tailored strategies to balance data efficiency and model generalization, particularly for large-scale architectures.

5.4 Analysis

5.4.1 Scaling Analysis

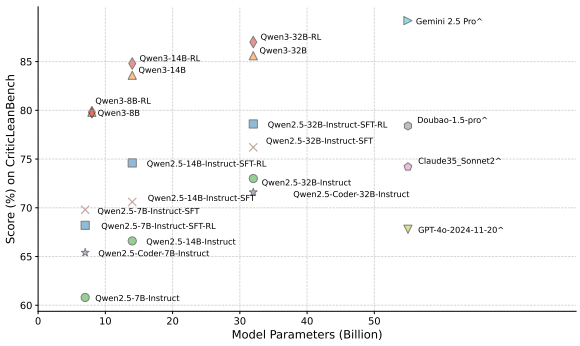
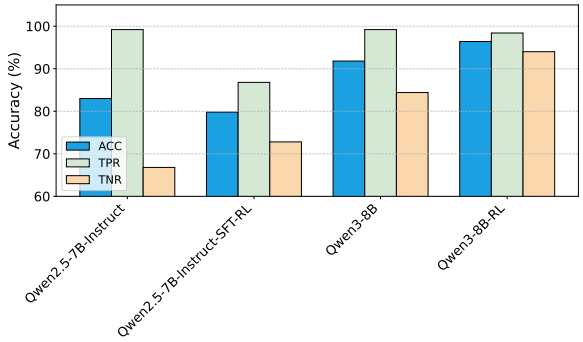
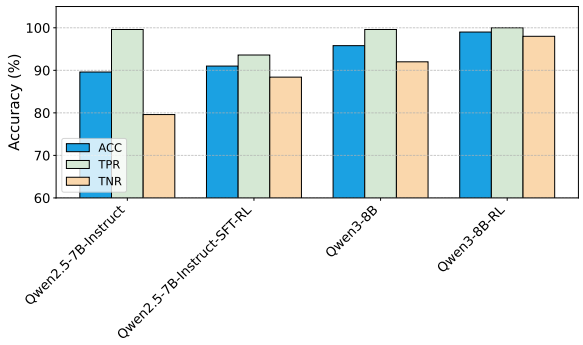


Figure 3: Scaling Analysis of LLMs on CriticLeanBench. $\boxed{\hat{}}$ denoted closed-source LLMs.

We evaluate the performance of Qwen series models on CriticLeanBench across different model scales, including Qwen2.5-Coder, Qwen2.5-Instruct, Qwen2.5-Instruct-SFT, Qwen2.5-Instruct-SFT-RL, Qwen3, and Qwen3-RL. The results in Figure 3 show that the performance improves consistently as the model size increases, demonstrating a clear scaling law of LLMs on CriticLeanBench.



(a) k = 8



(b) k = 32

Figure 4: Performance on CriticLeanBench using **Pass@k** metrics, where **k = 8** (top) and **k = 32** (bottom).

5.4.2 Effect of Pass@k

The Pass@k metric identifies high-quality responses from large language models, demonstrating their potential for improvement through post-training techniques like RLHF (Li et al., 2023) and GRPO (Shao et al., 2024b). This study evaluates Qwen2.5-7B-Instruct, Qwen2.5-7B-Instruct-SFT-RL, Qwen3-8B, and Qwen3-8B-RL on CriticLeanBench using Pass@8 and Pass@32, measuring Accuracy (ACC), True Positive Rate (TPR), and True Negative Rate (TNR).

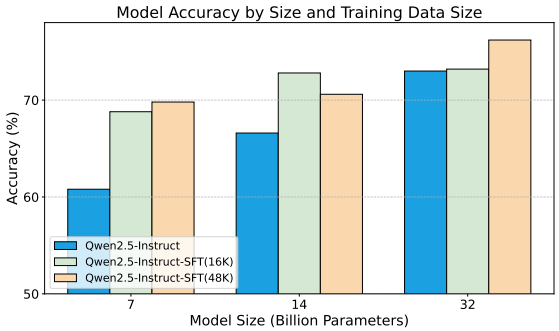


Figure 5: **Comparison of model performance under different amounts of SFT data:** base model, CriticLeanInstruct(16K), and CriticLeanInstruct.

As shown in Figure 4, models with Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) achieve superior overall performance. At Pass@32, Qwen2.5-7B-Instruct-SFT-RL outperforms Qwen2.5-7B-Instruct in accuracy, markedly enhancing overall correctness. For True Negative Rate (TNR), models without SFT-RL more readily misclassify errors as correct, while SFT-RL models mitigate this: Qwen2.5-7B-Instruct-SFT-RL shows a higher TNR than its base counterpart at Pass@32, lowering such risks. A similar trend appears in Qwen3-8B. All models perform better as k increases, suggesting SFT-RL-optimized models more effectively select high-quality responses with more candidates, highlighting their strengthened ability to identify superior outputs with additional candidates.

6 FineLeanCorpus

Building on this pipeline, we develop **FineLeanCorpus**, an open-source dataset of 509,358 verified pairs of mathematical statements and their Lean 4 Code. As shown in Table 6, compared to prior datasets, it offers greater diversity in mathematical domains, difficulty levels, and quality, validated through iterative critical feedback. For more details, see Appendix E.

6.1 Analysis on CriticLean Pipeline

As shown in Table 7, our autoformalization pipeline significantly improves accuracy. We selected 50 problems from the Omni-MATH and applied the following three formalization strategies, with the correctness of all outputs confirmed by manual human inspection.

Our baseline model, Kimina-Autoformalizer-7B, is used in each strategy: (1) a **Single Pass** baseline (38.0% accuracy); (2) a **Compiler Feedback** loop, where the model regenerates formalizations until they successfully compile (54.0% accuracy); (3) our **CriticLean Pipeline**, which extends this process, regenerating formalizations until they both compile successfully and are validated by our integrated **CriticLeanGPT Model** (84.0% accuracy).

While compiler feedback resolves syntactical errors, it fails to detect logical flaws. Our pipeline addresses this gap. The integration of critic model, which performs deeper semantic and logical validation, is directly responsible for the accuracy increase from 54.0% to 84.0%. By filtering out plausible but incorrect formalizations, our method

# Attempt	1	5	10	50	100	200
Count / Ratio	63 / 12.6%	137 / 27.4%	170 / 34.0%	229 / 45.8%	245 / 49.0%	264 / 52.8%

Table 4: **Effectiveness of the Multi-Attempt Strategy on Formalization Yield**. The table shows the cumulative number of successfully formalized problems retained by the critic model as the attempt limit increases. Statistics are from a 500-problem sample.

Dataset	Accuracy	Dataset	Accuracy
NuminaMath-TIR (LI et al., 2024)	78%	IneqMath (Jiayi et al., 2025)	96%
BlueMO (Zhang et al., 2024b)	86%	DeepMath-103k (He et al., 2025c)	84%
DAPO-Math-17k (Yu et al., 2025a)	69%	OnlineMathContest	88%
AOPs (Mahdavi et al., 2025b)	73%	TAL-SCQ5K (Math-eval, 2023)	75%
Omni-MATH (Gao et al., 2024)	84%	DeepTheorem (Zhang et al., 2025b)	100%
DeepScaleR (Luo et al., 2025)	100%		

Table 5: Human evaluation results of different sources.

Dataset	Source	Theorems	Level	Detailed Critic Process	Difficulty Profile	Topic Diversity
Lean-Workbook (Ying et al., 2025)	Synthetic	140K	Undergraduate	Opaque	Avg: 3.70 Top-tier (≥6): 7.81%	Highly Skewed
FineLeanCorpus (ours)	Synthetic	**509K**	Diverse	**Transparent**	Avg: 3.59 Top-tier (≥6): **17.16%**	**Balanced & Diverse**

Table 6: Comparison of dataset statistics. FineLeanCorpus offers a transparent critic process, a higher proportion of top-tier problems, and a more balanced and diverse topic distribution compared to the highly skewed Lean-Workbook.

Model	ACC
Kimina-Autoformalizer-7B	38.0
Kimina-Autoformalizer-7B (Compiler)	54.0
Kimina-Autoformalizer-7B (CriticLean)	$\boxed{84.0}$

Table 7: **Human evaluation for autoformalization performance:** The best score is highlighted in $\boxed{box}$.

provides a more robust path toward reliable autoformalization. These verified results present strong empirical evidence for the efficacy of our approach.

Table 4 shows our pipeline achieved a 52.8% success rate across 500 problems, where success required passing both syntactic validation and our critic model's semantic check. The value of our multi-attempt strategy is evident: while only 12.6% of samples passed on the first try, the pipeline successfully recovered an additional 40.2% that would be discarded by single-pass systems. Conversely, the 47.2% failure rate within the 200-attempt limit highlights a fundamental bottleneck: the pipeline's performance is ultimately constrained by the base auto-formalization model's ability to produce a candidate our critic can approve.

Moreover, as shown in Table 5, we also provide the human evaluation results of different sources. We observe that the accuracy varies across different sources, a discrepancy we attribute primarily to the differing domain and difficulty distributions of the respective datasets. [2]

7 Conclusion

This paper presents CriticLean, a comprehensive framework that positions the critic as a central component in the autoformalization of mathematical statements. Through the development of CriticLeanGPT and the construction of CriticLean-Bench, we demonstrate that explicitly modeling and training the critic yields significant improvements in formalization quality. Our pipeline not only refines the translation process through semantic validation, but also enables the construction of FineLeanCorpus, which is validated by both compiler and critic.

[2] Our human evaluation standard is particularly stringent. To calibrate our criteria, we inspected a random sample of 50 entries from the `Lean-Workbook`, which yielded an accuracy of 84%.

References

AI Research Group of TAL Education Group. K-12 handwritten mathematical expressions dataset (hme100k). https://ai.100tal.com/dataset. Accessed: 2025-04-05.

Anthropic. 2024. Claude 3.5 sonnet model card addendum. Accessed: 2024-09-21.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

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

B.P. Demidovich. 1964. *Problems in Mathematical Analysis. Edited by B. Demidovich. Translated From the Russian by G. Yankovsky*. Russian Monographs and Texts on Advanced Mathematics and Physics. Mir Publishers.

J Fan, S Martinson, EY Wang, K Hausknecht, J Brenner, D Liu, N Peng, C Wang, and MP Brenner. Hardmath: A benchmark dataset for challenging problems in applied mathematics. arxiv 2024. *arXiv preprint arXiv:2410.09988*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, and 1 others. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Google. 2023. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, and 1 others. 2025b. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *Preprint*, arXiv:2402.14008.

Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Zhongyuan Peng, Zhaoxiang Zhang, Zhicheng Zheng, Wenbo Su, and Bo Zheng. 2025a. Can large language models detect errors in long chain-of-thought reasoning? *Preprint*, arXiv:2502.19361.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025b. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, and 1 others. 2025c. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

Hui Huang, Yancheng He, Hongli Zhou, Rui Zhang, Wei Liu, Weixun Wang, Wenbo Su, Bo Zheng, and Jiaheng Liu. 2025. Think-j: Learning to think for generative llm-as-a-judge. *ArXiv*, abs/2505.14268.

Hugging Face. 2025. Open r1: A fully open reproduction of deepseek-r1.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Sheng Jiayi, Lyu Luna, Jin Jikai, Xia Tony, Gu Alex, Zou James, and Lu Pan. 2025. Solving inequality proofs with large language models. *arXiv preprint arXiv:2506.07927*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Leanprover Community. 2023. A read-eval-print-loop for Lean 4. https://github.com/leanprover-community/repl.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint*.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath tir. [https://huggingface.co/AI-MO/NuminaMath-TIR](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Zihao Li, Zhuoran Yang, and Mengdi Wang. 2023. Reinforcement learning with human feedback: Learning dynamic choices via pessimism. *arXiv preprint arXiv:2305.18438*.

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

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. *arXiv preprint arXiv:2402.14809*.

Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, and 1 others. 2023. Fimo: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*.

Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, Ke Jin, and 1 others. 2024a. M2rc-eval: Massively multilingual repository-level code completion evaluation. *arXiv preprint arXiv:2410.21157*.

Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, ZhiqiBai, Jie Liu, Ge Zhang, JiakaiWang, Yanan Wu, Congnan Liu, Jiamang Wang, Lin Qu, Wenbo Su, and Bo Zheng. 2024b. DDK: Distilling domain knowledge for efficient large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025a. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025b. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, Zhicheng Yang, Jing Tang, and Zhijiang Guo. 2024. Process-driven autoformalization in lean 4. *Preprint*, arXiv:2406.01940.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-S Notion Blog.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. 2025a. Leveraging online olympiad-level math problems for llms training and contamination-resistant evaluation. *arXiv preprint arXiv:2501.14275*.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. 2025b. Leveraging online olympiad-level math problems for llms training and contamination-resistant evaluation. *Preprint*, arXiv:2501.14275.

Math-eval. 2023. TAL-SCQ5K. https://github.com/math-eval/TAL-SCQ5K.

Leonardo de Moura and Sebastian Ullrich. 2021. 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*, page 625–635, Berlin, Heidelberg. Springer-Verlag.

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

Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, Berlin, Heidelberg.

Online Math Contest. Online math contest. https://onlinemathcontest.com/. Accessed: 2025-04-05.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5).

Miguel Angel Peñaloza Perez, Bruno Lopez Orozco, Jesus Tadeo Cruz Soto, Michelle Bruno Hernandez, Miguel Angel Alvarado Gonzalez, and Sandra Malagon. 2025. Ai4math: A native spanish benchmark for university-level mathematical reasoning in large language models. *arXiv preprint arXiv:2505.18978*.

Peter Scholze. 2022. Liquid tensor experiment. *Experimental Mathematics*, 31(2):349–354.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024a. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024b. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Christian Szegedy. 2020. A promising path towards autoformalization and general artificial intelligence. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings 13*, pages 3–20. Springer.

Terence Tao. 2023. The polynomial freiman-ruzsa conjecture. `https://github.com/teorth/pf`.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Qwen Team. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qwen Team. 2025b. Qwq-32b: Embracing the power of reinforcement learning.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, and 21 others. 2025a. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, Zichen Liu, Haizhou Zhao, Dakai An, Lunxi Cao, Qi Cao, Wanxi Deng, Feilei Du, Yiliang Gu, Jiahe Li, and 22 others. 2025b. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. 2025c. Reinforcement learning for reasoning in large language models with one training example. *Preprint*, arXiv:2504.20571.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. 2021. Naturalproofs: Mathematical theorem proving in natural language. *arXiv preprint*.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. 2022. Naturalprover: Grounded mathematical proof generation with language models. *arXiv preprint*.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Preprint*, arXiv:2205.12615.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024a. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2024b. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search.

Tianyi Xiong, Xiyao Wang, Dong Guo, Qinghao Ye, Haoqi Fan, Quanquan Gu, Heng Huang, and Chunyuan Li. 2025. Llava-critic: Learning to evaluate multimodal models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 13618–13628.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2024. Formal mathematical reasoning: A new frontier in ai. *Preprint*, arXiv:2412.16075.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Zheng Yuan, Dahua Lin, and Kai Chen. 2025. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Preprint*, arXiv:2406.03847.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, and 1 others. 2025a. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. 2025b. Formalmath: Benchmarking formal mathematical reasoning of large language models. *Preprint*, arXiv:2505.02735.

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

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025a. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? *arXiv preprint arXiv:2504.13837*.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, and 1 others. 2025b. VAPO: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*.

Alexander Zhang, Marcus Dong, Jiaheng Liu, Wei Zhang, Yejie Wang, Jian Yang, Ge Zhang, Tianyu Liu, Zhongyuan Peng, Yingshui Tan, and 1 others. 2025a. Codecriticbench: A holistic code critique benchmark for large language models. *arXiv preprint arXiv:2502.16614*.

Ge Zhang, Scott Qu, Jiaheng Liu, Chenchen Zhang, Chenghua Lin, Chou Leuang Yu, Danny Pan, Esther Cheng, Jie Liu, Qunshu Lin, and 1 others. 2024a. Map-neo: Highly capable and transparent bilingual large language model series. *arXiv preprint arXiv:2405.19327*.

Yifan Zhang, Yifan Luo, and Yizhou Chen. 2024b. Bluemo: A comprehensive collection of challenging mathematical olympiad problems from the little blue book series.

Ziyin Zhang, Jiahao Xu, Zhiwei He, Tian Liang, Qiuzhi Liu, Yansi Li, Linfeng Song, Zhenwen Liang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025b. Deeptheorem: Advancing llm reasoning for theorem proving through natural language and reinforcement learning. *Preprint*, arXiv:2505.23754.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, and 1 others. 2023. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021.

A Limitations

The current framework requires careful tuning of hyperparameters for SFT and RL training processes to maintain optimal performance. Additionally, CriticLeanBench's evaluation scope remains limited by its relatively small test dataset (500 samples) despite its rigorous design. Potential risks include the pipeline's performance being fundamentally constrained by the base auto-formalization model's ability to generate candidates, resulting in a high error rate for complex problems even after 200 attempts.

B Sample Selection Criteria and Error Type Coverage

We select representative samples that mirror the characteristics of the original autoformalization statements while maximizing diversity within each subset and capturing the full spectrum of observed error types in the negative samples. This is achieved by balancing several factors:

Stratified Representation: The selected samples should maintain a similar distribution across different strata of the original dataset. These strata are defined by:

- Problem Complexity: This encompasses factors like the number of variables, quantifiers, logical connectives, and the depth of nested mathematical structures.
- Mathematical Branches: The samples should represent the various mathematical domains present in the original data.
- Statement Well-Formedness: This refers to the degree to which the original mathematical statements adhere to standard mathematical notation and conventions. This stratification ensures that the subsets reflect the variability in the quality of the original problem statements.

Comprehensive Error Coverage: The negative samples are specifically chosen to exemplify the full range of typical errors observed in the autoformalization process. This range includes fundamental semantic and logical issues, such as Premise Translation Errors (e.g., incorrect domains or missing conditions), Mathematical Representation Errors (e.g., faulty expressions or definitions), and Goal Translation Errors. The set also covers issues such as Incorrect Assumptions, logical flaws like Operator & Parenthesis Errors (e.g., misplaced quantifiers), and high-level structural problems like Incomplete Formalization, where crucial context is omitted(detailed in figure9).

C Details of CriticLeanBench

C.1 Comparison to Other Benchmarks

Benchmark	Critic	Lean	Test
CriticBench	✓	✗	✓
CodeCriticBench	✓	✗	✓
LLaVA-Critic	✓	✗	✗
CriticLeanBench (ours)	✓	✓	✓

Table 8: Dataset statistics and comparison of various code benchmark datasets.

In Table 8 , CriticLeanBench has the following features: (1) We focus on a modest data size of 500 samples, acknowledging the expensive manual annotation costs (ranging from tens to hundreds of dollars per instance) and prioritizing efficiency without compromising evaluation rigor; (2) We integrate both Critic and Lean functionalities, distinguishing ourselves from benchmarks like CriticBench (Lin et al., 2024) and CodeCriticBench (Zhang et al., 2025a) that lack Lean capabilities, and LLaVA-Critic (Xiong et al., 2025) that omits both Lean capabilities and test evaluation components; (3) We incorporate a test component focused on translating mathematical statements into formally verified theorem declarations in Lean 4, offering a fine-grained evaluation framework to assess the correctness of model transformations. This aspect of evaluative completeness remains unmatched by existing datasets.

D Details of CriticLeanInstruct

D.1 Seed Data

The seed data comprises 4,000 samples evenly split into 2,000 correct and 2,000 incorrect instances. For both correct and incorrect samples, human experts provided critical feedback K. Additionally, the incorrect samples include compiler error messages generated by the Lean 4 Compiler as supplementary feedback. Then, we adopt the Gemini2.5-Pro (Google, 2023) to extend the critical feedback to detailed Chain-of-Thought (CoT) explanations.

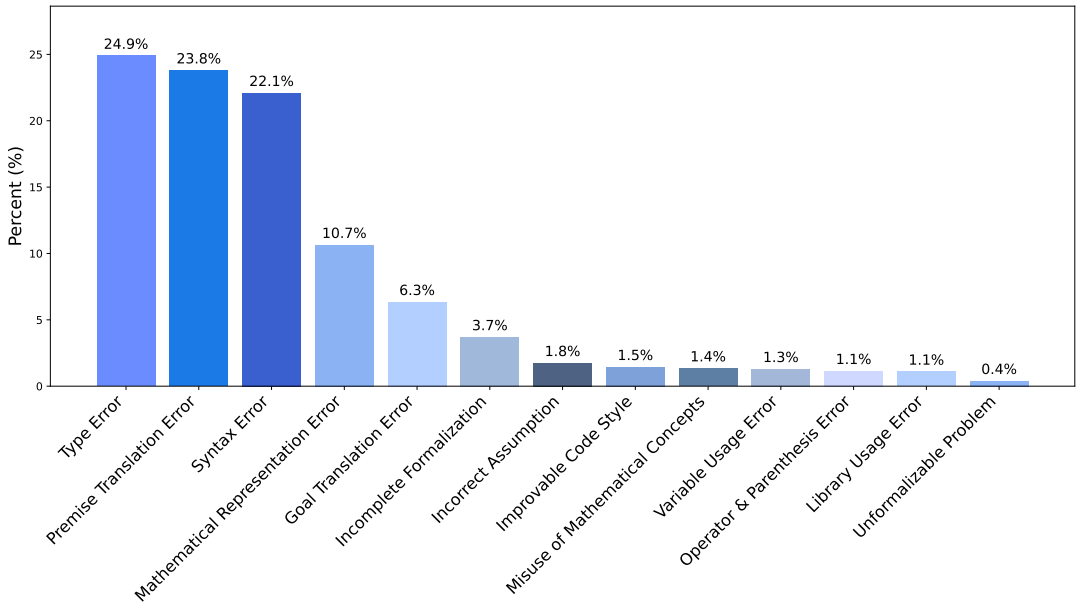


Figure 6: Distribution of Different Error Types.

To understand the primary failure modes, we analyzed the error distribution across the 2,000 incorrect samples in our seed data as illustrated in Figure 6. The analysis reveals two primary challenges:(1) Syntactic Barriers: The most frequent obstacles are syntactic, such as Type Error and Syntax Errors. These issues typically prevent the code from compiling, indicating a fundamental difficulty in mastering the formal language.(2) Semantic Gaps: Beyond syntax, these errors stem from a failure to interpret the natural language scenario and accurately model its key information. This is evident in high rates of errors when translating a problem's core logical components—including its premises, goal, and mathematical representation. This semantic challenge is particularly acute in application-style "word problems," where difficulty comprehending complex contexts leads to a fundamentally flawed formalization.

Drawing from the error taxonomy (detailed in Appendix I) and building upon our initial annotation standards (detailed in Appendix H), we established a fine-grained checklist, which is provided in full in Appendix J. This checklist formalizes the observed error paradigms, providing the foundational framework for methodically constructing the negative samples used throughout our study.

D.2 Data Augmentation

D.2.1 Correct Samples

We selected **5560** correct mathematical statements and Lean code pairs from the FormalMATH (Yu et al., 2025b) dataset and used the Gemini-2.5-Pro model to generate Critical Chain-of-Thought for initial assessment. To ensure data quality, we kept these samples where the model's judgment is "correct."

D.2.2 Incorrect Samples

We provide two strategies to obtain the incorrect samples as follows:

- Based on OmniMath, we adopt the Kimina-Autoformalizer-7B (Wang et al., 2025a) to generate the Lean 4 code statements, and we kept 2,000 Lean code snippets that failed to compile due to syntactic or logical issues during automated formalization processes. For each detected error, a detailed Chain-of-Thought (CoT) explanation was generated to elucidate the error's cause, enabling the model to recognize common compilation error patterns and thereby enhance its understanding of Lean 4 code syntax.

- Based on correct mathematical statements and Lean 4 code from the FormalMATH (Yu et al., 2025b) dataset, we implemented a three-step collaborative process to generate negative samples, aiming to enhance the CriticLeanGPT model's ability to identify subtle errors and logical flaws. First, a checklist of various potential issues, refined by human experts, was established as shown in Appendix J. Second, the Gemini 2.5 Pro model was invoked to randomly select error types from this checklist and modify correct Lean code accordingly, generating incorrect samples(detailed in Appendix M). Then, we adopt the Gemini model to generate the critical Chain-of-Thought explaination.

E FineLeanCorpus

To construct the FineLeanCorpus, we began by aggregating a vast and diverse collection of natural language mathematical problems. Sourcing from a wide array of materials, including high school olympiad datasets (e.g., AoPS, BlueMO), standard high school curricula (e.g., TAL-SCQ5), and undergraduate-level challenges (e.g., Omni-MATH), we ensured an extensive initial distribution in both the mathematical domain and difficulty. The first step in our process was to standardize this heterogeneous collection into a uniform, proof-based format, making each problem compatible with the Lean 4 theorem prover and ready for the subsequent formalization pipeline.

Dataset	**Difficulty Level**	**Size**
AOPs (Mahdavi et al., 2025b)	High School Olympiad	350714
DeepMath-103k (He et al., 2025b)	Diverse	45853
NuminaMath-TIR (LI et al., 2024)	High School	45152
DeepTheorem (Zhang et al., 2025b)	High School Olympiad	31409
DeepScaleR (Luo et al., 2025)	High School Olympiad	22360
DAPO-Math-17k (Yu et al., 2025a)	High School	8868
Omni-MATH (Gao et al., 2024)	Undergraduate	1181
IneqMath (Jiayi et al., 2025)	High School Olympiad	1180
BlueMO (Zhang et al., 2024b)	High School Olympiad	1099
TAL-SCQ5K (Math-eval, 2023)	High School	393
OnlineMathContest	High School	156
Multi-Source Math Competition	High School Olympiad	993

Table 9: Overview of different sources for FineLeanCorpus.

These Standardized problems were then subjected to a rigorous, gated auto-formalization process powered by our CriticLean framework (Figure 1).The Kimina-Autoformalizer-7B model first generates a candidate formal statement. This statement must pass a syntactic check via the Lean 4 compiler; failure leads to regeneration. A successful compilation is followed by a semantic correctness check from our CriticLeanGPT model, with rejection also triggering a new attempt. This regenerative approach is critical, maximizing the yield from our source corpus by iteratively seeking a valid formalization. Finally, to further enhance precision, we apply a final filtering stage using another, higher-performance CriticLeanGPT model. Manual validation indicates that this step is expected to eliminate 74.7% of the remaining incorrect formalizations. The resulting corpus is characterized by its expressive range: natural language statements vary from a concise 9 tokens to a complex 2,984 tokens (avg. 86.1), while their

corresponding Lean formalizations span from 7 to 2112 tokens (avg. 94.1), reflecting the deep spectrum of complexity successfully captured by our pipeline.

Statistics	Number
#Problems	509356
Length	
Statement	
maximum length	2984 tokens
minimum length	9 tokens
avg length	86.1 tokens
Lean Result(success)	
maximum length	2112 tokens
minimum length	7 tokens
avg length	94.1 tokens

Table 10: Dataset statistics of FineLeanCorpus.

To further analyze the differences between our FineLeanCorpus dataset and the Lean-Workbook, we employed the templates from Appendix O and Appendix N to assess the difficulty levels and classify the mathematical domains of the datasets using Doubao-1.5-pro (Guo et al., 2025b). A comparative analysis of our proposed FineLeanCorpus against Lean-Workbook (Figure 7, Figure 8, Table 9 and Table 10) reveals two fundamental advancements: (1) Scale and Coverage: Our corpus provides a significant quantitative expansion in both problem difficulty and domain coverage. This expansion is evident not only across nearly the entire difficulty spectrum—offering a much richer data pool for training foundational skills—but also in the substantial augmentation of high-volume domains like Intermediate Algebra and Elementary Number Theory. (2) FineLeanCorpus exhibits a more diverse and structurally balanced profile, achieved by collecting natural language statements from numerous sources. From a difficulty perspective, it features a multimodal distribution with substantial problem counts at several distinct complexity points, in stark contrast to the unimodal distribution of Lean-Workbook. This characteristic is crucial for mitigating model overfitting to a narrow complexity band. From a domain perspective, it substantially reinforces previously underrepresented areas. For instance, categories such as Analytic Geometry and Integral Calculus are significantly expanded, while niche topics like Algorithms and Graph Theory are also robustly augmented. This targeted enrichment transforms sparsely sampled topics into well-supported, learnable sub-domains, yielding a more comprehensive dataset designed to foster holistic reasoning capabilities. More details regarding our FineLeanCorpus dataset, including its fine-grained mathematical domain distribution (see Appendix F), are provided for further analysis.

Furthermore, to push the boundaries of current models and foster research into the upper echelons of mathematical reasoning, We have curated a specialized training subset from FineLeanCorpus, which we designate as Diamond. This subset comprises 87,354 problems with a difficulty rating of 6 or higher. The purpose of this high-difficulty training set is to create a demanding training environment that fosters the development of the sophisticated, multi-step reasoning required to tackle the most formidable mathematical problems.A detailed breakdown of the mathematical domain distribution within this subset is provided in table 12.

F Fine-Grained Domain Distribution of FineLeanCorpus

The following table illustrates the fine-grained domain distribution of the FineLeanCorpus by presenting its mathematical topics with relatively high frequency for illustrative purposes. To enhance readability and highlight the hierarchical structure, entries are sorted alphabetically by Main Category, then Sub-Category, and Topic.

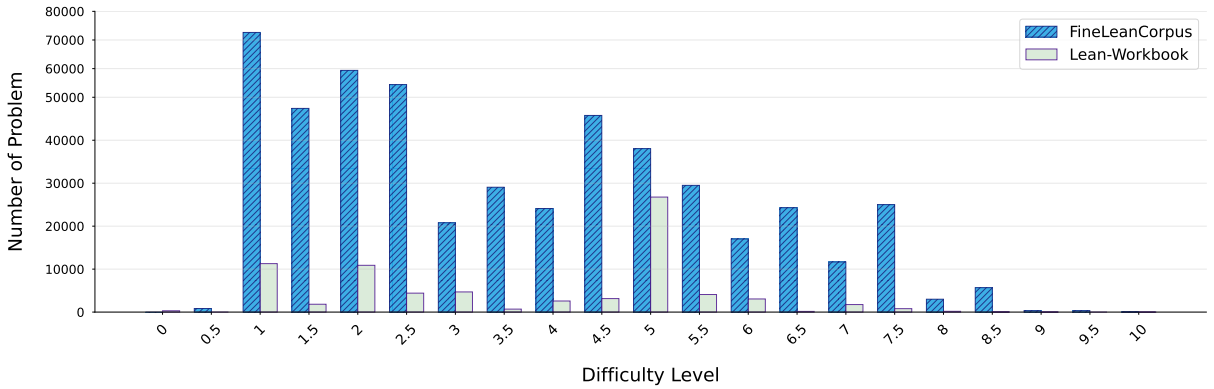


Figure 7: Comparison of dataset statistics. FineLeanCorpus offers a transparent critic process, a higher proportion of top-tier problems, and a more balanced and diverse topic distribution compared to the highly skewed Lean-Workbook.

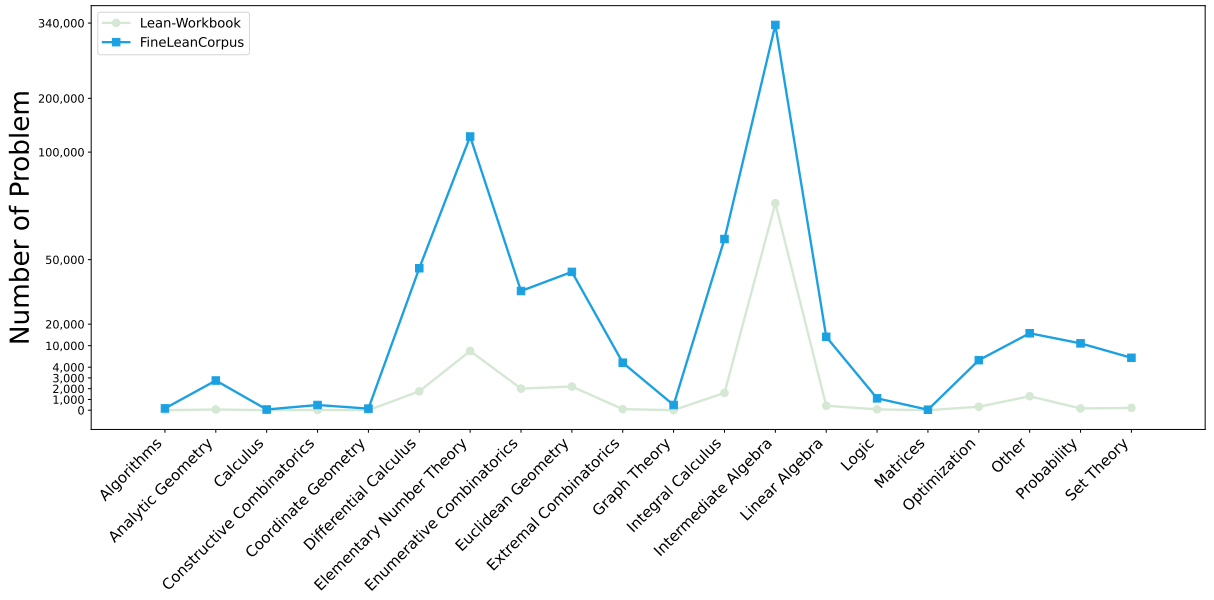


Figure 8: Math Domain Distributions: FineLeanCorpus vs. Lean-Workbook.

Table 11: Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
Algebra	Intermediate Algebra	Diophantine Equations	26
		Equations	64
		Exponential Functions	18
		Functional Equations	53978
		Inequalities	119092
		Other	104546
		Polynomials	58722
		Sequences	37
		Sequences and Series	47
		Trigonometric Identities	15
	Linear Algebra	Determinants	16
		Matrices	9130
		Other	170

(Continued) Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
		Vector Spaces	4815
	Other	Group Theory	13
		Other	29
Applied Mathematics	Algorithms	Greedy Algorithms	96
		Other	71
	Optimization	Linear Programming	1024
		Other	3629
	Other	Other	630
	Probability	Conditional Probability	1832
		Expected Value	2792
		Other	6459
Calculus	Calculus	Other	65
	Differential Calculus	Applications of Derivatives	15903
		Continuity	32
		Derivatives	20025
		Differential Equations	29
		Limits	902
		Other	9066
		Series Expansion	16
		Series Expansions	19
		Taylor Series	12
	Integral Calculus	Applications of Integrals	51
		Definite Integrals	26361
		Other	33180
	Other	Limits	1087
		Limits of Multivariable Functions	35
		Limits of Sequences	106
		Other	8950
	Series and Sequences	Other	18
Combinatorics	Constructive Combinatorics	Invariants	427
		Other	56
	Enumerative Combinatorics	Binomial Coefficients	19
		Combinations	12359
		Inclusion-Exclusion	20
		Other	18327
		Permutations	4658
		Pigeonhole Principle	35
	Extremal Combinatorics	Other	860
		Pigeonhole Principle	3557
	Graph Theory	Other	422
		Trees	32
	Other	Other	23

(Continued) Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
Discrete Mathematics	Graph Theory	Other	28
	Logic	Other	31
		Propositional Logic	1072
	Other	Other	4678
	Set Theory	Cardinality	2755
		Other	2119
Geometry	Analytic Geometry	Conic Sections	2440
		Coordinate Geometry	45
		Other	274
	Coordinate Geometry	Other	85
		Transformations	12
	Euclidean Geometry	Circles	3923
		Conic Sections	188
		Coordinate Geometry	14562
		Inequalities	21
		Other	9596
		Transformations	1105
		Triangles	14898
		Trigonometry	14
	Other	Other	45
Graph Theory	Other	Other	20
Number Theory	Elementary Number Theory	Diophantine Equations	36121
		Diophantine Equations.	14
		Divisibility	33467
		Divisibility.	18
		Inequalities	44
		Modular Arithmetic	19811
		Other	22918
		Prime Numbers	16563
Other	Other	Other	52
Set Theory	Other	Other	11
Trigonometry	Other	Other	12
	Trigonometric Identities	Other	15

G Fine-Grained Domain Distribution of FineLeanCorpus-Diamond

The following table illustrates the fine-grained domain distribution of the Diamond dataset, our high-difficulty subset, by presenting its mathematical topics with relatively high frequency for illustrative purposes. To enhance readability and highlight the hierarchical structure, entries are sorted alphabetically by Main Category, then Sub-Category, and Topic.

Table 12: Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
Algebra	Intermediate Algebra	Functional Equations	16751
		Functional Equations.	5
		Inequalities	23963
		Inequalities.	4
		Other	9104
		Polynomials	7793
		Sequences	11
		Sequences and Series	10
	Linear Algebra	Determinants	14
		Matrices	2619
		Other	54
		Vector Spaces	768
	Other	Group Theory	5
		Other	16
Applied Mathematics	Algorithms	Greedy Algorithms	7
		Other	11
	Optimization	Linear Programming	130
		Other	257
	Other	Other	60
	Probability	Conditional Probability	112
		Expected Value	229
		Other	247
Calculus	Calculus	Other	18
	Differential Calculus	Applications of Derivatives	3662
		Asymptotic Analysis	9
		Continuity	7
		Derivatives	1868
		Differential Equations	9
		Limits	69
		Other	1202
		Series	4
		Series Expansions	7
	Integral Calculus	Applications of Integrals	19
		Asymptotic Analysis	6
		Definite Integrals	5467
		Other	7830
		Series	5
		Series Convergence	4
	Optimization	Other	6
	Other	Limits	45
		Other	895
	Series and Sequences	Other	7
Combinatorics	Constructive Combinatorics	Invariants	196

(Continued) Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
		Other	34
	Enumerative Combinatorics	Binomial Coefficients	15
		Combinations	1227
		Other	3130
		Permutations	533
		Pigeonhole Principle	13
	Extremal Combinatorics	Other	566
		Pigeonhole Principle	1428
	Graph Theory	Other	221
		Trees	9
	Other	Other	11
Complex Analysis	Other	Other	5
Discrete Mathematics	Graph Theory	Other	19
	Logic	Other	13
		Propositional Logic	89
	Other	Other	1465
	Set Theory	Cardinality	850
		Other	436
Geometry	Analytic Geometry	Conic Sections	206
		Coordinate Geometry	8
		Other	30
	Coordinate Geometry	Other	24
		Transformations	6
	Euclidean Geometry	Circles	448
		Conic Sections	28
		Coordinate Geometry	1712
		Inequalities	19
		Other	1471
		Transformations	179
		Triangles	2825
		Trigonometry	6
	Other	Other	23
Graph Theory	Other	Other	14
Number Theory	Elementary Number Theory	Diophantine Equations	8899
		Diophantine Equations.	5
		Divisibility	7684
		Divisibility.	12
		Inequalities	18
		Modular Arithmetic	5068
		Modular Arithmetic.	5
		Other	4851
		Prime Numbers	5515

(Continued) Distribution of Problems by Mathematical Topic

Main Category	Sub-Category	Topic	Count
Other	Other	Other	17
Physics	Mechanics	Rotational Dynamics	5
Set Theory	Cardinality	Other	6
	Other	Other	4
Trigonometry	Trigonometric Identities	Other	7

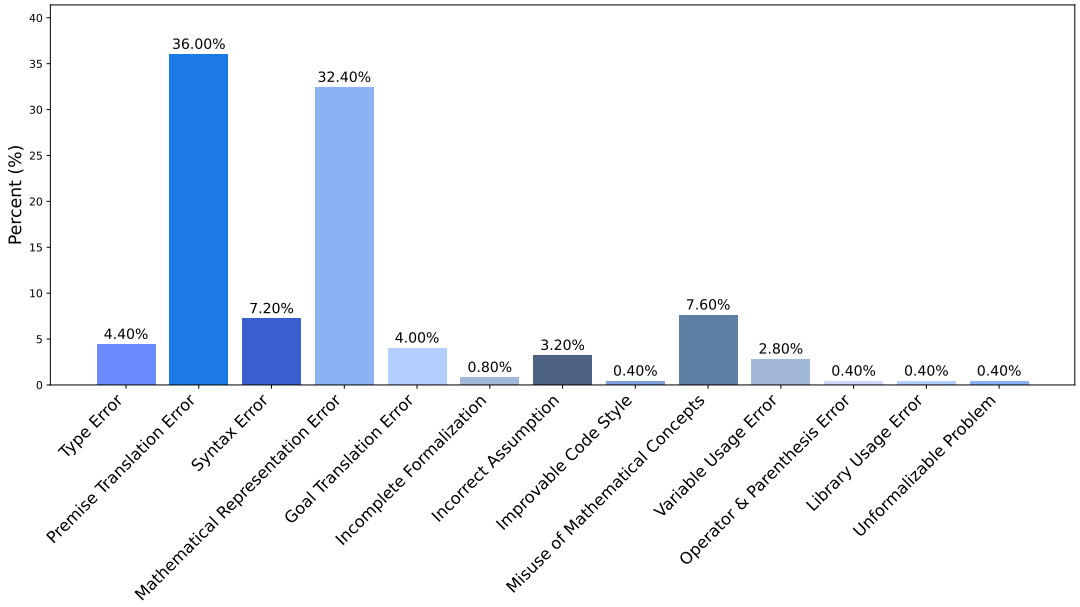


Figure 9: Distribution of Different Error Types of CriticLeanBench.

H Formalization Quality Assessment Criteria

Formalization Quality Assessment Criteria

I. Integrity & Accuracy of Mathematical Content

- **Conditions & Hypotheses:** Are all explicit premises, variable domains (e.g., $\mathbb{N}$, $\mathbb{R}$, `Fin k`), index ranges (e.g., a_0 vs. a_1), properties of specific objects (e.g., geometric shapes, algebraic structures), and implicit context (e.g., non-zero divisors) accurately translated? Are mathematical meanings preserved (e.g., $\neq 0$ vs. > 0)?
- **Goals & Conclusions:** Are all goals/conclusions translated (including multiple parts/cases)? Is the goal type accurate (e.g., specific value, extremum, existence/uniqueness)? For extrema, is attainability addressed? Is the mathematical meaning accurately translated?

II. Clarity & Correctness of Logical Structure

- **Propositional Structure:** Are logical connectives ($\leftrightarrow$, $\rightarrow$, $\wedge$, $\vee$, $\neg$) and quantifiers ($\forall$, $\exists$, $\exists!$) used correctly? Are quantifier order, scope, and nesting accurate (e.g., dependencies like

$\forall \epsilon > 0, \exists \delta > 0, \dots$)?

- **Relation of Conditions to Conclusions:** How do multiple premises combine (e.g., $(A \wedge B) \to C$ vs. $A \to (B \to C)$)? Are constraints within the correct scope?
- **Reasoning Path:** Does the formalization reflect the original logic and key steps without introducing flaws or altering proof difficulty?

III. Lean Conventions & Technical Accuracy

- **Syntax & Declarations:** Is the syntax (e.g., parentheses, keywords, type declarations) correct? Are `theorem`, `example`, `lemma` used appropriately?
- **Type System:** Do operations, parameters, and return values satisfy type constraints? Are numerals used in the correct type? Are mathematical concepts correctly mapped to Lean counterparts?
- **Definitions & Library Usage:** Are custom definitions clear? Are imports correct and non-redundant? Are standard symbols and operations used correctly (e.g., `Complex.abs`, `Nat.Prime`)?
- **Code Style & Readability:** Are names clear and consistent? Are there sufficient comments for complex parts? Is there any redundancy?

IV. Problem Comprehension & Overall Consistency

- **Grasping the Core:** Does the formalization capture the core mathematical idea?
- **Internal Self-Consistency:** Are there any logical contradictions between the translated parts?
- **Suitability for Formalization:** Is the problem suitable for formalization? Are assumption-s/interpretations documented?

V. Formalization Strategy & Choices

- **Abstraction Level:** Is the abstraction level appropriate, avoiding unnecessary generalization or over-specification?
- **Alternative Evaluation:** Were alternatives considered? Was decomposition/modularization used for complex problems?

VI. Provability & Proof Assistance

- **Proof Complexity:** Does the formalization maintain a similar proof complexity? Were lemmas added to simplify the proof?
- **Automation Potential:** Is the structure amenable to automation tools?

I Error Taxonomy

Error Taxonomy

1. Semantic and Logical Errors

1.1 Premise Translation Error

- **Description:** This error occurs when formalizing the given conditions, constraints, or assumptions from the original problem, resulting in a discrepancy between the logical premises in the Lean code and the problem's description.
- **Examples:** Failing to constrain variables to be positive, integers, or coprime as required; not ensuring that denominators in mathematical expressions are non-zero; omitting geometric constraints such as "A, B, C form a triangle" in geometry problems; not explicitly specifying the exact range of a variable. For instance, relationships between angles and sides must be precisely defined in Lean, otherwise the resulting correspondence may not be unique.

Generated code

1.2 Mathematical Representation Error

- **Description:** This error involves an inaccurate representation of the form and meaning of mathematical entities such as variables and expressions from the original problem. This leads to the formalized mathematical proposition in the code being inconsistent with the original problem, thereby undermining formal semantic correspondence.
- **Examples:** Formalizing a cubic polynomial as a quadratic one; mistranslating "all eigenvalues are 1" as "the determinant is 1"; simplifying a complex algebraic relationship into an incorrect equation; using the conclusion as a premise; mismatch of mathematical entities. An incorrect expression structure can change the intrinsic structure and semantics of the original mathematical expression, even if the computed result might happen to be the same. For example, incorrectly formalizing a finite nested expression (e.g., $2002 + 21(2001 + \dots)$) as the sum of an infinite series.

1.3 Goal Translation Error

- **Description:** The final goal or conclusion achieved by the code does not match what the problem asks for, failing to complete the specified task.
- **Examples:** The problem asks for a specific numerical value, but the code only proves its existence; the problem asks to calculate the radius of convergence, but the code incorrectly solves for the sum of the series; the final answer has a numerical calculation error or a formal writing error (e.g., writing a fraction n/m as m/n). The final goal might also be translated incompletely, with omissions.

1.4 Variable Usage Error

- **Description:** Improper use of a variable's type, scope, name, or index.

- **Examples:** Using natural numbers ($\mathbb{N}$) for a variable that requires real numbers ($\mathbb{R}$); off-by-one errors in summation or sequence indices; confusing or redefining variable names.

1.5 Misuse of Mathematical Concepts

- **Description:** Incorrectly using a mathematical formalism in Lean to represent a different mathematical concept.
- **Examples:** Translating "calculate the residue" as "find the limit"; formalizing "locally uniform convergence" as "pointwise convergence"; treating a problem of counting unordered combinations (e.g., non-congruent triangles) as one of counting ordered tuples.

1.6 Incorrect Assumption

- **Description:** Adding conditions that are not present in the original problem, which oversimplifies the problem or leads to an incorrect conclusion.
- **Example:** Introducing an unfounded assumption, such as a specific numerical value.

2. Lean Syntax and Technical Errors

- **Description:** These are technical issues at the code level that prevent the code from compiling or cause unexpected runtime behavior.

2.1 Syntax Error

- **Description:** The code does not conform to the basic syntax rules of Lean 4.
- **Examples:** A `theorem` statement is missing its name; the `by sorry` block to skip a proof is absent; incorrect keywords or symbols are used.

Generated code

2.2 Type Error

- **Description:** Performing incompatible operations on variables of different data types, including type mismatches and type casting errors.
- **Examples:** Performing division on a natural number (`Nat`) and expecting a fractional result, but the outcome is floored to 0; failing to cast integers or natural numbers to real numbers before performing real-valued operations.

2.3 Operator & Parenthesis Error

- **Description:** The calculation order of an expression does not match the intended logic due to misunderstandings of operator precedence, improper placement of quantifiers, or incorrect use of parentheses.
- **Example:** $\tan^2(\frac{\pi}{9})$ is incorrectly parsed as $(\frac{\tan \pi}{9})^2$.

2.4 Library Usage Error

- **Description:** Improper use of functions or definitions from `mathlib`.
- **Examples:** Using `.ncard` on an incorrect type of set; using a deprecated function name like `Complex.abs`.

3. Translation Completeness and Other Meta-Errors

- **Description:** These errors reflect that the formalization fails to cover all requirements of the problem, or that the problem itself is difficult to formalize.

3.1 Unformalizable Problem

- **Description:** The original problem description is vague, ambiguous, relies on diagrams, or involves real-world scenarios that are difficult to express in formal logic.
- **Examples:** The problem depends on a geometric figure that is not explicitly defined; a physical context or narrative scenario cannot be modeled precisely; the problem statement itself contains mathematical errors.

Generated code

3.2 Incomplete Formalization

- **Description:** The code only formalizes part of the problem, omitting other requirements.
- **Examples:** Ignoring a "prove or disprove" requirement and assuming the statement is true by default; omitting multi-step derivation requirements from the problem.

3.3 Improvable Code Style

- **Description:** The code may be logically correct but can be improved in terms of clarity, robustness, or adherence to conventions.
- **Examples:** Adding parentheses could enhance logical clarity; variable names are not intuitive; better use could be made of Lean's syntactic features.

J Complete Checklist for Lean4 Mathematical Formalization

Complete Checklist for Lean4 Mathematical Formalization

Conditions & Hypotheses:

1. Completeness of Preconditions: Are all explicitly stated preconditions in the problem translated without omission?
2. Accuracy of Variable Domains: Are the domains of variables (e.g., $\mathbb{N}, \mathbb{N}^+, \mathbb{R}$, `Fin k`, `Set.Icc a b`) accurately translated?
3. Accuracy of Indexing: Do the starting points and ranges of sequence/function indices (e.g., a_0 vs a_1, `Finset.range n` vs `Finset.Icc 1 n`) align with the original intent?
4. Clarity of Object Properties: Are the properties of specific objects (e.g., geometric figures like trapezoids, incircles; algebraic structures like groups, rings) clearly expressed?

5. Inclusion of Implicit Conditions: Are common implicit contextual conditions in mathematics (e.g., non-zero divisors, non-negative radicands, non-degenerate geometric objects, definedness of functions/sequences at application points, default to real numbers if unspecified) appropriately added?

6. Accuracy of Conditional Semantics: Is the mathematical meaning of conditions (e.g., "not equal to 0" ($\neq 0$) vs "greater than 0" (> 0), direction of inequality signs ($>$ vs $\geq$)) accurately translated?

Goals & Conclusions:

1. Completeness of Goals/Conclusions: Are all goals/conclusions that need to be proven or solved translated? (Pay special attention to multi-part conclusions and multiple solution scenarios).

2. Precision of Goal Type: Is the type of goal to be solved precise (e.g., specific value, extremum, existence/uniqueness, universal property, equivalence relation, implication)?

3. Attainability in Extremum Problems: For extremum problems, is "attainability" explicitly stated (i.e., demonstrating not just an inequality, but also that equality can be achieved)?

4. Accuracy of Goal Semantics: Is the mathematical meaning of the goals accurately translated?

Combination of Preconditions
Logical Structure:

1. Accuracy of Logical Connectives: Does the use of logical connectives ($\leftrightarrow$ (iff), $\rightarrow$ (if...then...), $\wedge$ (and), $\vee$ (or), $\neg$ (not)) accurately reflect the logical relationships of the original proposition?

2. Appropriateness of Quantifiers: Is the use of quantifiers ($\forall$ (for all), $\exists$ (exists), $\exists!$ (exists uniquely)) appropriate?

3. Correctness of Quantifier Scope and Nesting: Do the order, scope, and nesting of quantifiers correctly express the dependencies between variables (e.g., in $\forall \epsilon > 0, \exists \delta > 0, \ldots, \delta$ depends on ϵ)?

4. Combination of Preconditions: How do multiple preconditions combine to affect the conclusion (e.g., differentiate $(A \wedge B) \rightarrow C$ from $A \rightarrow (B \rightarrow C)$)?

5. Fidelity to Original Logic: Does the formalization faithfully represent the inherent logic and key steps of the original mathematical problem?

Lean Technical Accuracy:

1. Correctness of Basic Syntax: Is the basic Lean syntax (parenthesis matching, keywords like `theorem`, `def`, `variable`, `let`, `by`) entirely correct?

2. Adherence to Type Constraints: Do all operations, function parameters, and return values satisfy Lean's type constraints?

3. Correct Mapping of Mathematical Concepts: Are mathematical concepts correctly mapped to their Lean counterparts?

4. Clarity of Custom Definitions: Are all custom functions, predicates, and notations used clearly defined?

5. Correctness of Imports: Are necessary definitions and lemmas correctly imported from Mathlib?

Overall Consistency:

1. Capturing Core Mathematical Ideas: Does the formalization truly capture the core mathematical ideas and goals of the original problem?

2. Absence of Logical Contradictions: Are there any logical contradictions between the translated conditions, definitions, and goals?

3. Appropriateness for Formalization: Is the problem itself suitable for precise, unambiguous mathematical formalization?

4. Documentation of Assumptions: Are any assumptions or interpretative choices made during the formalization process documented?

K Prompt:Critical Feedback to CoT

Prompt:Critical Feedback to CoT

Instruction: You will be provided with a mathematical text and its Lean4 code representation. Your task is to evaluate whether the Lean4 code accurately and semantically represents the mathematical text. You will be given a boolean indicating conversion success and potentially failure information. Based on this, use a step-by-step Chain of Thought (COT) to generate a detailed explanation for why the conversion is considered successful or failed, focusing on the semantic equivalence and formal correctness of the Lean4 code relative to the mathematical meaning. The Lean4 code must preserve the intended meaning in the mathematical text and use correct Lean4 syntax and structure.
Input: You will receive the following four values:

1. **Mathematical Text**: A string containing mathematical content.

2. **Lean4Code**: A string representing the code equivalent of the mathematical text.

3. **Conversion Success**: A boolean value (True or False) indicating whether the mathematical text was successfully converted to the Lean4 code representation.

4. **Reason**: A string representing the code equivalent of the mathematical text.

5. If Conversion Success is True, this field will typically be empty. You must generate the detailed justification for the success.

6. If Conversion Success is False, this field will contain specific information pinpointing why

the conversion failed. You must elaborate on this identified failure, incorporating the analysis modules described below.

Your Role: You are an AI language assistant. Your role is to analyze the provided information and, using a step-by-step Chain of Thought (COT) approach, generate the explanation for the conversion's success (if indicated as successful) or elaborate on the identified failure information (if indicated as failed). **Guidelines:**

1. **Understand the Content:** Carefully read the mathematical text, the code representation, the Conversion Success value, and the Reason input (if Conversion Success is False).

2. **Generating the Explanation for Success (if Conversion Success is True):** If the conversion is successful, provide a detailed, step-by-step explanation using COT to justify why it is successful. Your justification should implicitly cover:

   (a) **Mathematical Text Analysis:** Briefly identify the core mathematical components (definitions, variables, operations, relations, constraints, statements) present in the text.

   (b) **Lean4 Code Analysis:** Briefly describe the structure and components of the provided Lean4 code, outlining how it represents the mathematical elements.

   (c) **Comparative Analysis:** Systematically compare each mathematical component identified in the text analysis with its corresponding part described in the Lean4 code analysis. Explain step-by-step how the Lean4 syntax and constructs accurately translate the mathematical concept and semantics.

   (d) **Confirmation of Correctness:** Explicitly confirm the absence of errors by verifying that:

   - Definitions: Types (variables, functions, sets, etc.) are correctly defined in Lean4, match the mathematical context, accurately reflect the intended meaning, and have a valid interpretation within Lean4. (Absence of Errors in Definition)
   - Constraints: All constraints from the mathematical text are present (no omissions), accurately represented (no inconsistencies or logical flaws), and without unnecessary additions (no redundancy). (Absence of Constraint Errors)
   - Syntax: The Lean4 code uses correct syntax for logical expressions (quantifiers, connectives), mathematical expressions ($\sum$, $\int$, $\{\}$, operations), and overall Lean4 language structure (theorem declarations, keywords like by). (Absence of Syntax Errors)
   - Proof Targets/Statements: If applicable, the proof target or statement in Lean4 is complete, unambiguous, and consistent with the mathematical claim. (Absence of Proof Target Errors)
   - Confirm that the Lean4 code fully and accurately captures the mathematical meaning without loss or distortion, preserving the intended meaning. Highlight the key aspects demonstrating a correct and complete translation.

3. **Elaborating on Failure (if Conversion Success is False):** If the conversion is unsuccessful, use a step-by-step COT approach structured around the following analysis modules to elaborate on the specific failure identified in the input Reason field:

(a) **Mathematical Text Analysis:** Briefly identify the core mathematical components (definitions, variables, operations, relations, constraints, statements) present in the text. This establishes the intended meaning.

(b) **Lean4 Code Analysis:** Briefly describe the structure and components of the provided Lean4 code, outlining how it *attempts* to represent the mathematical elements identified in the text analysis step. This describes the code *as presented*, before focusing on the error.

(c) **Comparative Analysis:** Compare the intended meaning and structure derived from the mathematical text analysis with the actual Lean4 code structure described in the previous step. **Crucially, use the input `Reason` to pinpoint and focus the comparison on the specific segment(s) where the Lean4 code fails to accurately represent the mathematical text**, clearly demonstrating the mismatch or divergence indicated by the failure reason.

(d) **Identification of Omission/Error:** Based on the discrepancy identified in the comparative analysis (which was guided by the input `Reason`), clearly articulate the specific error. Categorize this error by referencing the relevant error type and explain why the issue constitutes this type of error. Use the following categories and examples as a guide:

- Errors in Definition: e.g., Incorrect or mismatched type definitions (Type Mismatch), failure to reflect the precise mathematical context (Context Mismatch), definitions being mathematically meaningless or ill-formed in Lean4 (Ill-formed Definition).
- Constraint Errors: e.g., Omission of necessary constraints (Constraint Omission), Redundancy of constraints (Constraint Redundancy), Inconsistency with mathematical text (Constraint Inconsistency), Logical Errors within the translated constraints (Logical Flaw in Constraint).
- Syntax Errors: e.g., Errors in Logical Expression Syntax (quantifiers $\forall$, $\exists$; connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$), Mathematical Expression Syntax (notation for sums $\sum$, integrals $\int$, sets $\{\}$, specific operations), or Lean4 Language Structure Syntax (keywords like `theorem`, `def`, `variable`, `assume`, `show`, `by`; overall declaration structure). (Invalid Lean Syntax, Logical Syntax Error, Mathematical Notation Error)
- Proof Target Errors: e.g., Complete Omission of the target statement (Target Omission), Partial Omission or ambiguity in the target (Target Incomplete/Ambiguous), Inconsistency between the Lean4 goal and the mathematical claim (Target Mismatch).

(e) **Explanation of Impact:** Detail the consequences of this specific error. Explain how it alters the logical meaning compared to the original mathematical text, introduces ambiguity, makes the statement fundamentally different, or renders the Lean4 code syntactically invalid or semantically incorrect relative to the intended mathematical statement.

4. **Crucially:** Your final explanation should present this analysis directly. Do not explicitly state "The provided reason indicates..." or similar phrases referring back to the input Reason field in your output. Simply explain the error based on the structure above.

- **Focus on Explanation:** Your primary goal is to generate a clear and comprehensive explanation using the COT method, ensuring each step in your reasoning is explicit.
- **No Evaluation of Failure Information:** If the conversion failed, do not question or evaluate the validity of the failure information given in the Reason input. Your task is solely to elaborate on it based on that information, following the specified structure and error categorization.

Output Format: The output must be in English and presented strictly as follows:

```
Reason: [Your step-by-step Chain of Thought explanation here, either
justifying success or elaborating on the identified failure following
the specified modules and error types for failures]
```

This section will contain a single paragraph with your detailed Chain of Thought explanation.
Notes:

- Clarity and Conciseness: Ensure your explanation is clear, logical, and easy to understand. While using COT, ensure each step is articulated concisely.
- Professional Tone: Maintain an objective and professional tone throughout your response.
- No Additional Information: Do not introduce any external information or perform the conversion yourself. Focus solely on generating the explanation based on the provided inputs, using a step-by-step approach guided by the structure and error categories provided.

L Prompt:Formal Verification Expert

Prompt:Formal Verification Expert

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 —

M Prompt:Lean Flaw Injection

Prompt:Lean Flaw Injection

You are an exceptional Lean 4 code formalization engineer. Your current task is to meticulously introduce errors into correct Lean 4 code, following a specific checklist of error types.
I will provide you with:

1. A problem pair consisting of:
   (a) A mathematical definition or statement.
   (b) Its corresponding correct Lean 4 code formalization.
2. An error checklist and Lean 4 theorem statement: A list of potential error types or modification strategies, along with a Lean 4 theorem statement that formalizes the problem. Note that the proof is intentionally omitted (e.g., using `sorry`).

Your process should be as follows:

1. From the provided checklist, select exactly 2 error types or modification strategies. Each chosen item must be directly and plausibly applicable to the structure, logic, or types within the provided mathematical statement and its Lean 4 code.
2. Based on your selection(s), intentionally modify the Lean 4 code to make it incorrect or subtly deviate from the original mathematical intent. The modification should be contextually relevant to the provided mathematical statement and its original Lean 4 formalization. Aim for errors that someone might plausibly make when formalizing *this specific* concept, rather than completely arbitrary changes.
3. When applying multiple selected error types, aim to incorporate all of them naturally into the code modification. If this proves too complex or makes the resulting error contrived, you may focus on a primary subset of the selected types, but clearly explain your rationale and how the chosen modifications relate to your selections.
4. Important: Do not add any comments directly within the mathematical description or the Lean 4 code itself to explain your changes. All explanations should be in the "Detailed Explanation of Modifications" section.

You must then return your response as a JSON object with the following structure and content:
“‘json
Output Format
You must then return your response as a JSON object with the following structure and content:

```
{
  "modified_lean_code": "[Your intentionally flawed Lean 4 code here.
  Please ensure that you do not add any comments directly within the
  mathematical description or the Lean 4 code itself to explain your
  changes.]",
  "explanation": "Here, you will:\n1. Clearly state which 2 error
  types or modification strategies you selected from the checklist.
  \n2. Explain precisely what changes you made to the original
  correct code to introduce these errors/deviations.\n3. Describe
  how these changes reflect the selected strateg(ies). Crucially,
  point out the specific \"error points\" you introduced by comparing
  the modified code to the (unstated) original correct version, and
  explain *why* this type of error is plausible or relevant given
  the specific mathematical content and structure of the original
  code."
}
```

checklist:
{selected_checklist_items_placeholder}
"Mathematical statement": "{refined_statement_placeholder}"
"Lean4 code": "{autoformalization_placeholder}"

N Prompt: Mathematical Problem Classification Task

Prompt: Mathematical Problem Classification Task

I am working with natural language statements of advanced mathematics problems, which are intended to be later formalized in Lean. Before formalization, I need to categorize each problem into its appropriate mathematical domain.
#OBJECTIVE#

1. Summarize the math problem in one or two sentences, highlighting the key mathematical concepts or structures involved.
2. Classify the problem into one or more mathematical domains, using a hierarchical classification chain. For example:
   `Algebra -> Intermediate Algebra -> Inequalities`.

The classification should be based on the mathematical content of the *natural language* problem, even if no formal notation is used yet.

#DOMAIN TAXONOMY#

The domain classification should follow a standard taxonomy:

<math domains>

Algebra -> Intermediate Algebra -> Inequalities

Algebra -> Intermediate Algebra -> Polynomials

Algebra -> Intermediate Algebra -> Functional Equations

Algebra -> Linear Algebra -> Vector Spaces

Algebra -> Linear Algebra -> Matrices

Geometry -> Euclidean Geometry -> Triangles

Geometry -> Euclidean Geometry -> Circles

Geometry -> Euclidean Geometry -> Coordinate Geometry

Geometry -> Euclidean Geometry -> Transformations

Geometry -> Analytic Geometry -> Conic Sections

Number Theory -> Elementary Number Theory -> Divisibility

Number Theory -> Elementary Number Theory -> Modular Arithmetic

Number Theory -> Elementary Number Theory -> Diophantine Equations

Number Theory -> Elementary Number Theory -> Prime Numbers

Combinatorics -> Enumerative Combinatorics -> Permutations

Combinatorics -> Enumerative Combinatorics -> Combinations

Combinatorics -> Extremal Combinatorics -> Pigeonhole Principle

Combinatorics -> Constructive Combinatorics -> Invariants

Combinatorics -> Graph Theory -> Trees

Calculus -> Differential Calculus -> Derivatives

Calculus -> Integral Calculus -> Definite Integrals

Discrete Mathematics -> Logic -> Propositional Logic

Discrete Mathematics -> Set Theory -> Cardinality

Applied Mathematics -> Probability -> Expected Value

Applied Mathematics -> Probability -> Conditional Probability

Applied Mathematics -> Optimization -> Linear Programming

Applied Mathematics -> Algorithms -> Greedy Algorithms

Algebra -> Intermediate Algebra -> Other

Geometry -> Euclidean Geometry -> Other

Number Theory -> Elementary Number Theory -> Other

Combinatorics -> Enumerative Combinatorics -> Other

Calculus -> Integral Calculus -> Other

Discrete Mathematics -> Other -> Other

</math domains>

#RESPONSE FORMAT# ##Summarization [A brief summary of the problem, describing the mathematical concepts it involves.]

##Math domains [A hierarchical classification of the mathematical domains involved in the problem.]

#INSTRUCTIONS#

- You may include up to **three** domain classification chains, separated by semicolons.
- The format must be: `Major Domain -> Subdomain -> Specific Topic`.
- If a concept doesn't fit exactly, use `Other` as the last node only. For example: `Algebra -> Intermediate Algebra -> Other`.
- Avoid using vague or overlapping categories.
- End each report with the line `=== report over ===`.

#INPUT# Below is the original natural language math problem statement:
`<statement>{statement}</statement>`
#OUTPUT FORMAT# Please return your response in **JSON** format. For example:

```
{
  "Summary": "This problem involves minimizing a symmetric function
  over real variables under absolute value constraints.",
  "Domains": [
    "Algebra -> Intermediate Algebra -> Inequalities",
    "Calculus -> Differential Calculus -> Applications of Derivatives"
  ]
}
```

O Prompt:Difficulty Level Assessment

Prompt:Difficulty Level Assessment

You are an exceptional Lean 4 code formalization engineer. Your current task is to meticulously introduce errors into correct Lean 4 code, following a specific checklist of error types.
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

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

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 $\angle BAC = 60°$, $\angle CBA \leq 90°$, $BC = 1$, and $AC \geq AB$. 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 $\angle BC_1C + \angle CA_1A + \angle AB_1B = 180°$. 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 $n \geq 3$, there

exists a balanced set consisting of n points. Determine all integers $n \geq 3$ 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 $x \neq 0$. (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 $AB \neq AC$. 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.

$$\begin{array}{ccccccc} & & & 4 & & & \\ & & 2 & & 6 & & \\ & 5 & & 7 & & 1 & \\ 8 & & 3 & & 10 & & 9 \end{array}$$

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.

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 structural
  elements or comparison to benchmark problems."
}
```