NL2Lean: Translating Natural Language into Lean 4 through Multi-Aspect Reinforcement Learning

Yue Fang[1,2*], **Shaohan Huang**[3†], **Xin Yu**[3], **Haizhen Huang**[3], **Zihan Zhang**[3]
Weiwei Deng[3], **Furu Wei**[3], **Feng Sun**[3], **Qi Zhang**[3], **Zhi Jin**[1,2*]
[1]School of Computer Science, Peking University, Beijing, China
[2]Key Laboratory of High Confidence Software Technologies (PKU), MOE, China
[3]Microsoft
y.fang@stu.pku.edu.cn, shaohanh@microsoft.com, zhijin@pku.edu.cn

Abstract

Translating natural language into formal language such as Lean 4 has gained attention for its potential to automate formal proof development. Automated methods provide a scalable and cost-effective alternative to manual formalization, driving increasing interest in this task. However, existing LLMs mainly rely on instruction tuning and lack fine-grained structural and semantic alignment, making it difficult to generate syntactically and logically sound formal proofs. To address this, we propose a reinforcement learning framework ReLean that enables LLMs to generate high-quality Lean 4 statements from natural language. We first fine-tune a LLaMA3-8B model on NL–Lean 4 data to obtain a base translator with basic translation ability. Then, we design a multi-aspect dense reward mechanism covering four key dimensions: semantic alignment, term-level alignment, global-level alignment, and compile-checking. Separate reward models are trained via preference modeling, and their normalized outputs are combined to guide optimization via PPO. Finally, a curriculum learning strategy based on multi-dimensional difficulty allows the model to learn progressively from simple to complex cases. Experiments on NL-to-Lean 4 tasks show that our method consistently outperforms baseline models. Further analysis on reward model and curriculum learning confirms their effectiveness in enhancing model performance.

1 Introduction

Recent advances in large language models (LLMs) have shown strong performance in mathematical reasoning tasks (Guo et al., 2025; Yang et al., 2024), particularly through natural language-based informal reasoning. However, such informal reasoning is difficult to verify automatically and lacks the rigor required for formal mathematics. Meanwhile, with the growing complexity of mathematical domains and increasing demand for formal correctness, traditional peer review alone is no longer sufficient for ensuring the validity of proofs. To address these challenges, formal proof assistants such as Lean (Moura and Ullrich, 2021; De Moura et al., 2015), Isabelle (Paulson, 1994), and Coq (Barras et al., 1999) have been developed. These formal languages allow computers to verify proofs automatically (Avigad, 2024), providing a clear and rigorous standard for assessing correctness.

However, writing accurate formal language like Lean 4 remains a significant burden for domain experts. It requires deep familiarity with formal syntax and logical rules, as well as considerable manual effort to execute low-level, repetitive proof steps (Jiang et al., 2022). Moreover, mathematicians often face the additional challenge of navigating unfamiliar theorem libraries and strict type systems—an especially difficult task for those more accustomed to expressing reasoning in high-level, informal natural language. As a result, the gap between informal and formal representations has drawn growing attention to the challenge of translating natural language into formal proofs. To illustrate this challenge, we present the following example:

- **Natural Language Sentence:**
  What is the sum of the smallest and second-smallest positive integers a satisfying the congruence $27a \equiv 17 \pmod{40}$? Show that it is 62.
- **Lean 4 Statement:**
  `theorem numbertheory_modulo_min_sum :`
  $(S : Set\ \mathbb{N})\ \ (u\ v : \mathbb{N})$
  $(h_0 : \forall a : \mathbb{N},\ a \in S \leftrightarrow 0 < a \wedge 27 * a\%40 = 17)$
  $(h_1 : \text{IsLeast}\ S\ u))$
  $(h_2 : \text{IsLeast}(S \setminus \{u\})\ v)$
  $\Rightarrow\ u + v = 62\ :=$ `by sorry`

Recent efforts have leveraged large language models (LLMs) to translate natural language into

[*]Work done during internship at Microsoft.
[†]Corresponding author

formal Lean 4 statements. For instance, TheoremLlama (Wang et al., 2024) employs instruction tuning and transfer learning to build a long Chain-of-Thought (CoT) translator for generating accurate formal statements. FANS (Yao et al., 2025) adopts a similar CoT-based approach but focuses on converting math question-answer pairs into verifiable Lean 4 propositions to facilitate formal answer selection. HERALD (Gao et al., 2024) constructs a large NL–Lean 4 dataset with hierarchical annotations to support supervised fine-tuning. While these methods improve generation quality, they rely on static supervision and lack fine-grained semantic feedback. This often results in Lean 4 outputs that are logically incorrect or semantically misaligned with the input natural language.

To address the limitations of instruction-tuned LLMs in Lean 4 statement generation, we propose ReLean, a reinforcement learning framework that systematically improves the translation of natural language into formal Lean 4 statements. The framework consists of three components: initialization, reward modeling, and curriculum-based optimization. We begin by fine-tuning a LLaMA3-8B model on a dataset of natural language and Lean 4 statement pairs. This step yields a base generator with the ability to translate natural language into Lean 4, forming the foundation for subsequent reinforcement learning.

Next, we define four types of reward signals to evaluate different aspects of statement quality. (1) semantic alignment evaluates how well the generated statement preserves the intent of the natural language input, using reverse translation and embedding similarity; (2) term-level alignment compares the generated subterm sequence with the reference using the normalized length of their longest common subsequence (LCS); (3) global-level alignment computes edit distance between the generated and reference Lean 4 statement to assess structural resemblance; and (4) compile checking verifies whether the generated statement can be compiled successfully. Each signal is modeled by a separate reward model trained through preference learning. During reinforcement learning with Proximal Policy Optimization (PPO), the normalized outputs of all reward models are aggregated into a unified feedback signal.

Finally, to improve training stability and efficiency, we apply a curriculum learning strategy based on multi-dimensional difficulty. Difficulty scores are computed per reward dimension, and a joint ranking is used to schedule training examples. The model is progressively trained on increasingly difficult examples, thereby improving its generalization and robustness through a stable learning process.

This framework enables the generation of Lean 4 statement that is not only syntactically correct but also semantically aligned and logically sound. Experiments on multiple NL-to-Lean 4 tasks show consistent improvements over strong baselines, particularly in producing structurally accurate and formally verifiable statements.

In general, our contributions are as follows:

- We propose a reinforcement learning framework that directly optimizes LLMs for Lean 4 statements generation. Our framework introduces a multi-aspect dense reward mechanism to provide fine-grained feedback.
- We design a curriculum learning strategy that organizes samples based on multi-aspect difficulty, enabling the model to learn progressively from simple to complex examples.
- Extensive experiments on Lean Workbook and ProofNet demonstrate that our ReLean framework outperforms supervised baselines, validating the effectiveness of multi-aspect rewards and curriculum learning.

2 Related Work

2.1 Automatic Formalization

Many researchers have explored the task of transforming natural language (NL) into formal languages such as Mizar, Isabelle, and Lean (Wang et al., 2018; Wu et al., 2022; Wang et al., 2024; Gao et al., 2024). For example, Wang et al. (2018) demonstrate the potential of neural machine transformation models for converting informal mathematics into formal Mizar statements. Later works leveraged LLMs to perform few-shot or instruction-tuned for transformation. For instance, Wu et al. (2022) and Patel et al. (2023) explore LLM-based autoformalization of mathematical problems, using staged or sketch-based pipelines. Zhou et al. (2024) further improve consistency by validating LLM-generated quantitative reasoning through autoformalization. Agrawal et al. (2022) use Codex with adaptive prompts to transform undergraduate-level math into Lean, while Azerbayev et al. (2022) constructed a benchmark for autoformalizing undergraduate math, and proposed prompt-based and backtranslation-based techniques. In addition, Gao et al. (2024) propose the HERALD dataset, which provides hierarchical annotations for both statements and proofs, enabling better fine-tuning on Lean 4. For Lean 4 specifically, Wang et al. (2024) introduce TheoremLlama, which combines dataset bootstrapping, curriculum learning, and block training to support full proof synthesis with in-line NL comments. FANS (Yao et al., 2025) further extends this idea to formalize math QA pairs into verifiable Lean propositions using a CoT-based formal reasoning process, enabling precise answer selection. While these methods demonstrate significant progress, they often rely on static datasets and coarse-grained supervision, lacking dense feedback on semantic and structural alignment. To address these limitations, we propose a reinforcement learning framework with multi-aspect feedback to improve both the semantic fidelity and logical soundness of Lean 4 formalization.

2.2 Reinforcement Learning for LLMs

With the rapid development of large language models (LLMs), reinforcement learning (RL) has become a widely adopted post-training technique (Ziegler et al., 2019; Christiano et al., 2017). A representative paradigm is reinforcement learning from human feedback (RLHF) (Christiano et al., 2017), which combines a learned reward model with policy optimization algorithms such as Proximal Policy Optimization (PPO) (Schulman et al., 2017). However, RLHF often suffers from instability due to sparse rewards. To address this, recent studies have further enhanced RL effectiveness by introducing multi-reward frameworks (Dann et al., 2023), which evaluate outputs from complementary perspectives. For example, Ryu et al. (2024) balance quality dimensions in summarization, while Wang et al. (2025) integrate retrieval and generation feedback in RAG. In parallel, Curriculum Learning (CL) (Bengio et al., 2009; Graves et al., 2017) has shown promise in organizing training from easy to hard, improving RL generalization (Justesen et al., 2018; Wang et al., 2019; Li et al., 2020). Methods like Kimi k1.5 (Team et al., 2025) and LogicRL (Xie et al., 2025) adopt staged curricula to stabilize training and improve performance. In this work, we build on these insights by proposing a reinforcement learning framework that integrates multi-dimensional dense reward modeling with curriculum scheduling to improve the transformation of natural language into Lean 4.

3 Approach

In this section, we present our approach for translating natural language into formal Lean 4 statement using a reinforcement learning framework. The method incorporates multiple reward mechanisms that provide fine-grained feedback across four key aspects: semantic alignment, tactic-level alignment, compile-checking and global structural similarity. During reinforcement learning, we further apply curriculum learning by organizing training examples based on their multi-aspect difficulty, as illustrated in Figure 1.

3.1 Problem Definition

Given a natural language instruction $x = \{w_1, \ldots, w_{L_x}\} \in \mathbb{X}$ that describes a mathematical statement or proof goal, the task is to generate a corresponding Lean 4 statement $y = \{z_1, \ldots, z_{L_z}\} \in \mathbb{Y}$, where each y_t is a token in the Lean 4 language.

The objective is to learn a mapping function $f_\theta : \mathbb{X} \to \mathbb{Y}$ that produces correct Lean 4 statements from natural language inputs.

3.2 Lean 4 Tanslator Initialization

We initialize a Lean 4 translator by fine-tuning the LLaMA 3-8B model on the HERALD dataset (Gao et al.), which contains approximately 580k NL–Lean 4 pairs. This pretraining step provides the model with basic capabilities for NL-to-Lean 4 translation, serving as an initialization for subsequent reinforcement learning.

3.3 Feedback Signal Construction

To mitigate common issues when translating natural language into Lean 4 statement, we design **four** reward functions that serve as RL feedback. These reward functions guide the model toward generating accurate, well-structured, and verifiable Lean4 statement.

Semantic Alignment Reward. LLMs often generate Lean 4 statement that deviate from the original user intent. To address this issue, we introduce a reward that evaluates how well the generated statement $\hat{y}$ preserves the semantics of the input natural language sentence x.

Specifically, we reverse-map the generated Lean 4 output $\hat{y}$ into a natural language representation $Rev(\hat{y})$ via prompting as illustrated in Appendix 4, making its semantic content comparable to the original input x. Both $Rev(\hat{y})$ and x are then embedded into a shared semantic space using a pre-trained

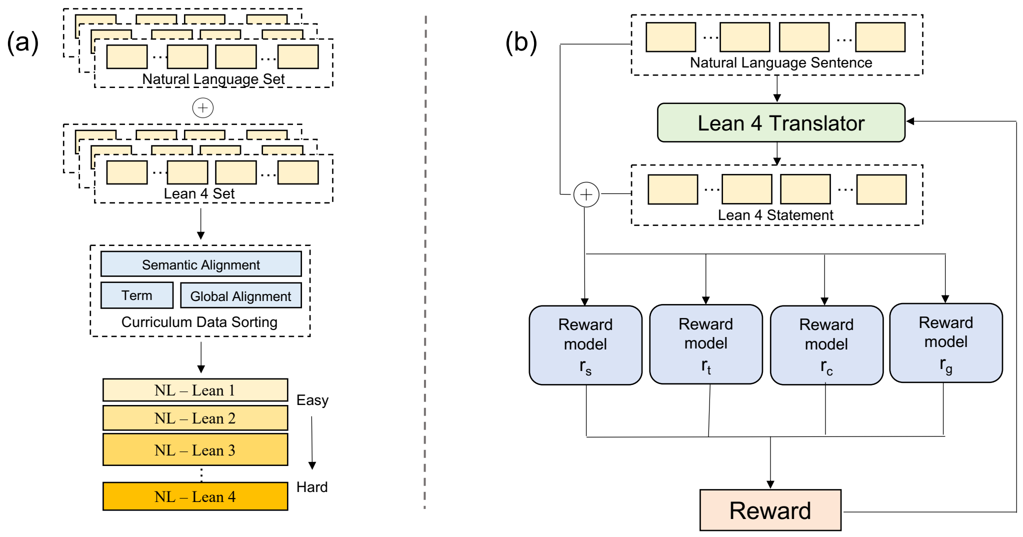


Figure 1: The overall framework of ReLean. (a) illustrates that the input data is ordered based on curriculum sorting. (b) shows that the multi-reward model evaluates Lean 4 outputs and provides feedback to the Lean 4 generator.

encoder, yielding vectors $v_{Rev(\hat{y})}$ and v_x. The semantic alignment reward is defined as the cosine similarity between these vectors:

$$m_{\text{s}} = \cos\big(v_{Rev(\hat{y})},\, v_x\big),$$

where cos denotes cosine similarity.

Term-Level Alignment Reward. We use *subterm* to refer to any syntactic component of a Lean expression, for example constants or intermediate goal formulas. In the NL-to-Lean 4 translation task, subterm sequences serve as a structural bridge between natural language and formal language. To assess whether the model captures the true trems, we propose *Term-Level Alignment Reward* that measures the structural alignment between the generated and reference subterm sequences.

We first use a prompt-based extraction mechanism as shown in Appendix 5 to extract the subterm sequence used in the generated statement, yielding $\hat{T} = [\hat{t}_1, \ldots, \hat{t}_r]$, and the reference subterm sequence $T = [t_1, \ldots, t_k]$, where $\hat{t}_i$ or t_i denotes the subterm used in the i-th step.

Then, we define the reward function based on the normalized length of the longest common subsequence (LCS) between the generated and reference subterm sequences:

$$m_{\text{t}} = \frac{\text{LCS}(\hat{T}, T)}{k},$$

where $\hat{T} = [\hat{t}_1, \ldots, \hat{t}_r]$ is the generated subterm sequence, $T = [t_1, \ldots, t_k]$ is the reference subterm sequence, and $\text{LCS}(\hat{T}, T)$ denotes the length of their longest common subsequence.

Complie-checking Reward. To ensure that the generated Lean 4 $\hat{y}$ is syntactically and type-theoretically valid, we adopt a REPL (Read-Eval-Print Loop) based framework to perform complie-time verification within the Lean 4 environment. The output is passed to the Lean 4 compiler, and a binary reward is assigned based on whether it compiles successfully:

$$m_{\text{c}} = \begin{cases} 1 & \text{if } \hat{y} \text{ passes checking,} \\ 0 & \text{otherwise.} \end{cases}$$

Global-Level Alignment Reward. To assess the overall structural similarity between the generated Lean 4 and the reference, we define a global reward based on the edit distance between the generated Lean 4 $\hat{y}$ and the ground truth y. This metric captures token-level discrepancies such as omissions and distortions.

The reward is defined as:

$$m_{\text{g}} = -\,\text{EditDist}(\hat{y},\, y),$$

where $\text{EditDist}(\cdot,\cdot)$ computes the unnormalized token-level edit distance between the two Lean 4 programs. A smaller distance corresponds to better global alignment, and the negative sign converts it into a reward signal.

3.4 Reward Model Training

Given a natural language input x, we obtain two generated Lean 4 candidates y_1 and y_2 from the initial generator, both corresponding to the same instruction. Each pair is evaluated using the **four** reward metrics introduced in Section 4.2: m_{s}, m_{t}, m_{c}, and m_{g}. For a metric m_*, we compare the two Lean 4 and label the one with the higher score as *chosen* and the other as *rejected*, thereby forming a preference pair. For example, under the semantic reward m_{s}, we record:

$$\{\,(\text{chosen} : [x, y_1],\ \text{rejected} : [x, y_2]) \\ \mid m_{\text{s}}(y_1) > m_{\text{s}}(y_2)\,\}$$

Reward Model Objective. We train a separate reward model r_ψ for each metric using the Bradley–Terry formulation. Given an input x with *chosen* Lean 4 statementy_c and *rejected* statement y_r, the preference likelihood is modeled as:

$$P_\psi(y_c \succ y_r \mid x) \;=\; \sigma\big(r_\psi(x, y_c) \;-\; r_\psi(x, y_r)\big),$$

where $\sigma(\cdot)$ denotes the logistic function. The objective minimizes the negative log-likelihood across the preference dataset D_p:

$$\Delta r_\psi := r_\psi(x, y_c) - r_\psi(x, y_r), \\ L = -\mathbb{E}_{(x,y_c,y_r)\in D_p}\big[\log \sigma(\Delta r_\psi)\big].$$

Model Architecture. Each reward model shares the same architecture: it is initialized from the generator's backbone and extended with a single linear head atop the final Transformer layer to produce a scalar score. We denote the reward models as: $r_{\text{s}\,\psi}$, $r_{\text{t}\,\psi}$, $r_{\text{c}\,\psi}$, and $r_{\text{g}\,\psi}$. Given an input pair $(x, \hat{y})$, we use the shorthand: $r_{\text{s}}(\hat{y})$, $r_{\text{t}}(\hat{y})$, $r_{\text{c}}(\hat{y})$, and $r_{\text{g}}(\hat{y})$.

Score Normalization. Because different reward models may produce outputs on different scales, we linearly normalize each reward to the $[0, 1]$ range before aggregation. The normalized scores are then combined with weights λ_i to produce the final scalar signal used in PPO fine-tuning.

3.5 Multi-Aspect Curriculum Learning

To improve the efficiency and stability of RL fine-tuning, we incorporate curriculum learning by organizing training examples from easier to more challenging cases. We define separate curricula based on our three reward metrics, capturing distinct dimensions of Lean 4 statement difficulty.

Semantic Alignment Curriculum. We reverse-map the initial Lean 4 output $\hat{y}_0$ into natural language and compute its semantic similarity with the input instruction x. Examples with lower similarity are considered more challenging, as the initial generator struggles to capture the intended meaning, and are scheduled later in the curriculum. High-similarity examples are introduced earlier to help the model first learn to preserve semantic alignment.

$$\text{Diff}_{\text{s}}(x, \hat{y}_0) = 1 - m_{\text{s}}(\hat{y}_0),$$

Term Curriculum. We measure difficulty based on the number of subterms in the reference statement. Examples with fewer subterms are introduced earlier, while those with longer subterm sequences are scheduled later in the curriculum.

$$\text{Diff}_{\text{t}}(y) = |\mathcal{T}(y)|,$$

where $\mathcal{T}(y)$ denotes the sequence of subterms in the reference Lean 4 statement y, $|\mathcal{T}(y)|$ is the number of subterms.

Global Alignment Curriculum. We assess difficulty based on the global similarity between the initial output $\hat{y}_0$ and the reference statement y, measured using the ROUGE score. Examples with higher ROUGE scores—indicating stronger structural alignment—are introduced earlier in training. Lower-scoring examples, which reflect greater divergence in global proof organization, are deferred to later stages.

$$\text{Diff}_{\text{g}}(\hat{y}_0, y) = 1 - \text{Rouge}(\hat{y}_0, y).$$

Curriculum Scheduling. We compute an overall difficulty score for each example by aggregating its difficulty under three criteria. Let $\mathcal{D} = \{(x^{(j)}, y^{(j)})\}_{j=1}^{N}$ denote the full dataset, and define the aggregated difficulty for each example as:

$$\text{Diff}_{\text{total}}^{(j)} = \frac{1}{3}\sum_{i=1}^{3} \text{Diff}_i(x^{(j)}, \hat{y}_0^{(j)}, y^{(j)}).$$

We then sort all examples in $\mathcal{D}$ in ascending order of their total difficulty:

$$\mathcal{D}_{\text{sorted}} = \text{Sort}_{\text{asc}}\left(\mathcal{D},\ \text{Diff}_{\text{total}}\right),$$

where Sort_{asc} denotes sorting the dataset in ascending order according to the total difficulty score.

3.6 Reinforcement Learning

To further improve the Lean 4 generator, we adopt a policy-gradient reinforcement learning framework. The NL-to-Lean 4 task is cast as a Markov decision process (MDP) defined by the 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$.

At decoding step t, the state s_t comprises the input instruction x and the partial output $\hat{y}_{<t}$, while the action a_t selects the next token $\hat{y}_t$. After generation is complete, the full program $\hat{y}$ is evaluated using a weighted sum of the four reward signals introduced in Section 4.2:

$$r_{\mathrm{RL}}(\hat{y}) = \lambda_{\mathrm{s}}\, m_{\mathrm{s}}(\hat{y}) + \lambda_{\mathrm{t}}\, m_{\mathrm{t}}(\hat{y}) + \lambda_{\mathrm{g}}\, m_{\mathrm{g}}(\hat{y}) + \lambda_{\mathrm{c}}\, m_{\mathrm{c}}(\hat{y}).$$

Here, m_{s}, m_{t}, m_{g}, and m_{c} denote the semantic alignment, tactic-level alignment, global alignment, and compile-checking rewards, respectively. The weights λ_{*} are hyperparameters.

We optimize G_θ with *proximal policy optimization* (PPO), maximizing expected reward while limiting divergence from the initialization G_{θ_0}:

$$\mathcal{L}_{\mathrm{PPO}} = \mathbb{E}\big[r_{\mathrm{RL}}(\hat{y})\big] - \eta\, \mathrm{KL}\big(G_\theta \parallel G_{\theta_0}\big), \qquad (1)$$

where η is a KL-penalty coefficient that stabilizes training by discouraging overly large policy shifts.

4 Experiments

In this section, we conduct experiments on Lean Workbook dataset (Ying et al., 2024a) and ProofNet dataset (Azerbayev et al., 2022) to evaluate our methods.

4.1 Experiment Settings

We first introduce our empirical settings, including datasets, evaluation measures, baselines and implementation details.

Datasets. We evaluate the translation ability of ReLean using two public datasets: Lean Workbook and ProofNet. Lean Workbook is a large-scale Lean 4 dataset automatically constructed from natural language math problems. We sample 500 natural language–formal language (NL–FL) pairs from the dataset to form an independent test set for performance evaluation. ProofNet is a standard benchmark dataset for automated theorem proving at the undergraduate level, containing Lean-formatted problems across areas such as real analysis, complex analysis, linear algebra, abstract algebra, and topology. While the original dataset is written in Lean 3, we adopt a converted version in Lean 4.9.0 provided by Xin et al. (2024) to ensure compatibility with model training and inference.

Evaluation Measures. Following the validation approach of Ying et al. (2024a); Gao et al., we first perform a compiler check on the generated Lean 4 code to ensure its syntactic correctness. Next, we apply InternLM2Math-Plus-7B to back-translate the formal statements into natural language. Finally, we use the DeepSeek Chat v2.5 model to compare the back-translated results with the original informal statements, assessing whether the mathematical semantics and intended meaning have been accurately preserved. The automatic accuracy is computed as the percentage of generated Lean 4 statements that can be both compiled and judged semantically correct.

For human evaluation, we randomly selected 50 NL-Lean 4 pairs from the test set of Lean Workbook and ProofNet. Four annotators (all students familiar with the syntax and semantics of Lean 4) were asked to compare the outputs of our model with those of baseline models. The annotators were blind to the source of each Lean 4 formal statement and did not know which ones were generated by our model or by the baselines. To complement the automatic evaluation and ensure fairness, we introduce a human evaluation phase. For formal statements that pass the automatic checks, human annotators judge each statement as either: (1) Correct: The formal statement clearly matches the meaning of the original natural language sentence. (2) Incorrect: The formal statement contains major errors, does not reflect the original meaning. The final human evaluation accuracy is calculated as the percentage of samples labeled as "Correct" out of the total number of evaluated samples.

Baselines. We select Six models with mathematical reasoning capabilities as our baselines, including LLaMA 3-8B-Instruct[1], InternLM2-Math (Ying et al., 2024b), GPT-4o[2], DeepSeek-v1 (Liu et al., 2024), TheoremLlama and Herald Translator. These models are general or math-oriented language models with varying levels of mathematical reasoning ability. Specifically, TheoremLlama is a fine-tuned model for Lean4 proof writing. Herald Translator fine-tuned on the Herald dataset, serves as a specialized model for translating natural language into Lean 4.

Implementation Details. Our experiments are conducted on 8 NVIDIA GeForce RTX 4090 GPUs (24GB VRAM each). We implement our framework using PyTorch (Imambi et al., 2021) and Huggingface Transformers (Wolf et al., 2019), with LLaMA-Factory as the base for model customiza-

[1]https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

[2]https://openai.com/index/hello-gpt-4o/

Model	Lean Workbook	ProofNet
Llama3-instruct	$14.51 \pm 0.72\%$	$7.26 \pm 0.34\%$
InternLM2-Math	$16.32 \pm 1.65\%$	$9.87 \pm 0.98\%$
GPT-4o	$19.78 \pm 0.65\%$	$11.85 \pm 1.24\%$
DeepSeek-v1	$19.95 \pm 0.85\%$	$12.84 \pm 0.62\%$
TheoremLlama	$20.86 \pm 1.33\%$	$13.88 \pm 0.94\%$
Herald	$22.36 \pm 0.65\%$	$14.67 \pm 0.71\%$
ReLean (ours)	$\mathbf{26.58 \pm 0.52\%}$	$\mathbf{19.20 \pm 0.31\%}$

Table 1: Accuracy of ReLean and baseline models on Lean Workbook and ProofNet.

Model	Accuracy (%)	
	Lean Workbook	ProofNet
Llama3-instruct	13.2	7.6
InternLM2-Math	17.4	10.2
GPT-4o	18.8	12.6
DeepSeek-v1	22.4	16.8
TheoremLlama	22.6	14.2
Herald	22.4	14.8
ReLean (ours)	**27.6**	**19.0**

Table 2: Human evaluation results on Lean Workbook and ProofNet.

Reward	Lean Workbook		ProofNet	
	Com. (%)	Acc. (%)	Com. (%)	Acc. (%)
m_s	17.3	16.8	17.1	15.3
m_t	20.1	18.5	15.6	14.8
m_c	19.3	15.6	18.9	14.3
m_g	24.9	24.6	17.8	17.5
ReLean (all)	**28.3**	**26.6**	**23.1**	**19.2**
LLaMA3 (Finetuned)	15.9	14.7	11.2	9.8

Table 3: Performance of different reward signals for translation. **Com.** indicates the percentage of generated Lean 4 can be compiled. **Acc.** denotes the proportion of statements that are both compiled judged semantically faithful.

tion. We select LLaMA 3-8B as the base Lean 4 translator and fine-tune it using PPO for 80,000 steps with a batch size of 32 and a learning rate of 1.41e-5. Each reward model is fine-tuned on LLaMA 3-8B with a linear value head for 5 epochs using the Adam optimizer (Kingma, 2014), a learning rate of 5e-5, and a batch size of 16. The KL penalty coefficient is set to $\eta = 0.05$. The final reward signal is computed as a weighted sum of four components, with weights $\lambda_1 = 0.2$, $\lambda_2 = 0.25$, $\lambda_3 = 0.35$, and $\lambda_4 = 0.2$.

4.2 Experimental Results

In this section, we demonstrate our experiment results on Lean Workbook and ProofNet datasets.

4.2.1 Automatic Evaluation

The evaluation results on Lean Workbook and ProofNet datasets are shown in Table 1. All models are queried using few-shot prompting. It can be observed that large models such as GPT-4o and Llama3-Instruct still exhibit suboptimal performance on formal language translation tasks, which is consistent with findings from previous studies (Wang et al., 2024; Yao et al., 2025).

Our ReLean model performs the best. ReLean achieves 26.58% on Lean Workbook, which outperforms the Herald model, i.e., 22.36%. From the results on ProofNet, we can see that our model also obtains the best performance. For example, our ReLean achieves 19.2%, which again outperforms Herald, i.e., 14.67%. In conclusion, our ReLean model has the ability to generate more accurate Lean 4 statements than baselines.

4.2.2 Human Evaluation

The results of human evaluation are shown in Table 2. We adopt the percentage of correct outputs as the evaluation metric to assess the semantic accuracy of generated Lean 4 formal statements. From the results, it can be observed that annotators consider our model to produce the highest proportion of correct Lean 4 code among all compared methods. For instance, on the Lean Workbook dataset, our model achieves an accuracy of 27.6%, significantly outperforming Herald at 22.4%, further demonstrating the advantages of our approach in semantic preservation and compilability.

4.3 Analysis

4.3.1 Ablation Study

As shown in Table 3, each single-reward variant outperforms the LLaMA3 (Finetuned) baseline, confirming that even isolated signals provide effective supervision for Lean 4 code generation. Among these, the global-level alignment reward (m_g) achieves the highest accuracy among all single-signal settings, reaching 24.6% on Lean Workbook and 17.5% on ProofNet, which highlights its effectiveness. The semantic alignment reward (m_s) also shows consistent improvements over the baseline, particularly in terms of semantic alignment, indicating its strength in preserving the user's intent. In contrast, although the compile-checking reward (m_c) achieves relatively high compilation rates (19.3% and 18.9%), it contributes only limited gains to semantic accuracy (15.6% and 14.3%), suggesting that syntactic validity alone is not sufficient to guarantee meaningful formalization. When integrating all four reward

Reward Model	Lean Workbook	ProofNet	Average
r_s	74.6	72.8	74.6
r_t	89.4	86.5	87.95
r_c	79.8	80.1	79.95
r_g	82.6	84.3	83.45

Table 4: The accuracy(%) of reward models.

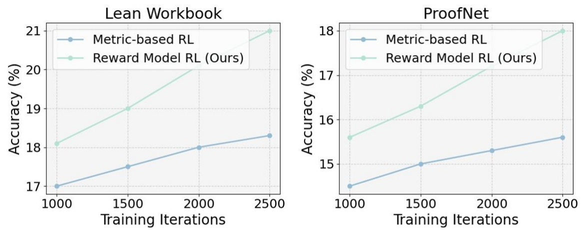


Figure 2: RL performance under reward models and metric-based reward on two datasets. Each curve represents the accuracy trend over training iterations when using a specific reward model.

signals, the full ReLean model achieves the highest overall performance, with a compile rate of 28.3% and accuracy of 26.6% on Lean Workbook, and 23.1% and 19.2% on ProofNet, respectively. These results demonstrate that combining multiple complementary reward signals is crucial for generating Lean 4 statements that are both formally correct and semantically faithful.

4.3.2 Accuracy of Reward Model

Table 4 reports the pairwise classification accuracy of each reward model on the Lean 4 preference test set. All reward models achieve strong performance, with classification accuracies exceeding 70% across both Lean Workbook and ProofNet datasets. These results validate the quality of the trained reward models and their suitability for guiding downstream reinforcement learning via PPO.

4.3.3 Impact of Reward Feedback

As shown in Figure 2, both on Lean Workbook and ProofNet datasets, using trained reward models as feedback leads to more stable and steadily improving accuracy across training iterations. In contrast, training directly with metric scores results in relatively slower improvement and less consistency. This indicates that reward models, learned from human or preference-based supervision, capture more reliable optimization signals, which ultimately contribute to better learning efficiency and semantic fidelity during Lean 4 statements generation.

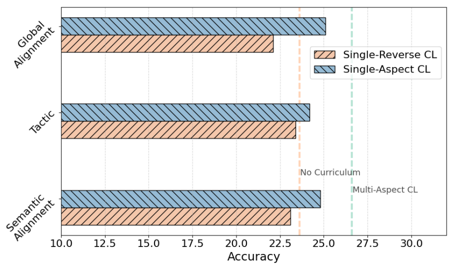


Figure 3: Comparison of Curriculum Learning Strategies.

4.3.4 Impact of Curriculum Learning

To evaluate the effectiveness of each curriculum component, we conduct controlled experiments in which only a single curriculum dimension is explicitly applied, while the remaining dimensions follow a default shuffled schedule (i.e., without curriculum). As shown in Figure 2, we consider two experimental configurations: (1) enabling only one curriculum dimension while disabling all others (Single-Aspect CL), and (2) reversing the training order of a specific curriculum dimension, i.e., training from hard to easy (Single-Reverse CL).

The results reveal three findings. First, compared to the full multi-aspect curriculum setting, training with a single curriculum dimension consistently underperforms in accuracy, indicating that multi-aspect curriculum learning is effective for model training. Second, reversing the training order in any single dimension leads to noticeable performance degradation, highlighting the importance of the easy-to-hard progression in curriculum design. Finally, all curriculum-based configurations outperform the baseline without curriculum learning, further validating the effectiveness of curriculum learning.

5 Conclusion

In this work, we present an RL-based framework, ReLean, for translating natural languages into Lean 4. The results of both metric-based and human evaluations demonstrate that our approach significantly improves transformation capabilities across two datasets. Our approach facilitates the automatic transformation of human-readable mathematical statements into machine-checkable formal proofs, contributing to more scalable and accessible formal verification.

Limitations

The limitation of our work lies in the evaluation process, which still relies on human annotation to assess the correctness and semantic alignment of generated Lean 4 statements. Although manual evaluation ensures high-quality assessment, it is time-consuming, expensive, and difficult to scal, especially when dealing with large datasets. In future work, we plan to leverage automated evaluation metrics to enable reliable and fine-grained assessment of formal proof generation, reducing reliance on human effort.

Acknowledgment

The author gratefully acknowledges the support of the National Nature Science Foundation of China under Grant Nos. 62192731, 62192732, and 62192730, which enabled a dedicated human resource for this research.

References

Ayush Agrawal, Siddhartha Gadgil, Navin Goyal, Ashvni Narayanan, and Anand Tadipatri. 2022. Towards a mathematics formalisation assistant using large language models. *arXiv preprint arXiv:2211.07524*.

Jeremy Avigad. 2024. Mathematics and the formal turn. *Bulletin of the American Mathematical Society*, 61(2):225–240.

Zhangir Azerbayev, Bartosz Piotrowski, and Jeremy Avigad. 2022. Proofnet: A benchmark for autoformalizing and formally proving undergraduate-level mathematics problems. In *Second MATH-AI Workshop*.

Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, et al. 1999. The coq proof assistant reference manual. *INRIA, version*, 6(11):17–21.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Christoph Dann, Yishay Mansour, and Mehryar Mohri. 2023. Reinforcement learning can be more efficient with multiple rewards. In *International Conference on Machine Learning*, pages 6948–6967. PMLR.

Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. 2015. The lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 378–388. Springer.

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

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. 2024. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. 2017. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. Pmlr.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. 2021. Pytorch. *Programming with TensorFlow: solution for edge computing applications*, pages 87–104.

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

Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. 2018. Illuminating generalization in deep reinforcement learning through procedural level generation. In *NeurIPS Workshop on Deep Reinforcement Learning*.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. 2020. Towards practical multi-object manipulation using relational reinforcement learning. In *2020 ieee international conference on robotics and automation (icra)*, pages 4051–4058. IEEE.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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 28*, pages 625–635. Springer.

Nilay Patel, Rahul Saha, and Jeffrey Flanigan. 2023. A new approach towards autoformalization. *arXiv preprint arXiv:2310.07957*.

Lawrence C Paulson. 1994. *Isabelle: A generic theorem prover*. Springer.

Sangwon Ryu, Heejin Do, Yunsu Kim, Gary Lee, and Jungseul Ok. 2024. Multi-dimensional optimization for text summarization via reinforcement learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5858–5871.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. 2018. First experiments with neural translation of informal to formal mathematics. In *Intelligent Computer Mathematics: 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings 11*, pages 255–270. Springer.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. 2024. Theoremllama: Transforming general-purpose llms into lean4 experts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11953–11974.

Yujing Wang, Hainan Zhang, Liang Pang, Binghui Guo, Hongwei Zheng, and Zhiming Zheng. 2025. Maferw: Query rewriting with multi-aspect feedbacks for retrieval-augmented large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25434–25442.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Jiarui Yao, Ruida Wang, and Tong Zhang. 2025. Fans–formal answer selection for natural language math reasoning using lean4. *arXiv preprint arXiv:2503.03238*.

Huaiyuan Ying, Zijian Wu, Yihan Geng, JIayu Wang, Dahua Lin, and Kai Chen. 2024a. Lean workbook: A large-scale lean problem set formalized from natural language math problems. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, et al. 2024b. Internlm-math: Open math large language models toward verifiable reasoning. *CoRR*.

Jin Peng Zhou, Charles E Staats, Wenda Li, Christian Szegedy, Kilian Q Weinberger, and Yuhuai Wu. 2024. Don't trust: Verify–grounding llm quantitative reasoning with autoformalization. In *The Twelfth International Conference on Learning Representations*.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

A Prompts input to Large Language Models

In this section, we present the prompts designed to guide large language models.

Lean 4 Statement-to-NL Prompt

You are given a Lean 4 theorem statement written in formal syntax.
Please translate only the statement into a clear and concise natural language sentence that preserves its mathematical meaning.
Use plain mathematical English that captures the core logic and relationships described by the formal statement.
Avoid referring to Lean syntax or keywords.
Your explanation should be understandable to someone familiar with basic mathematics and logic, but not with Lean.
Do not include any information about how to prove the statement.

Figure 4: Prompt used to generate natural language translations of Lean 4 theorem statements.

Subterm Extraction Prompt

You are given a Lean 4 theorem statement written in formal syntax.
Please extract all subterms that appear in the statement, including nested and repeated subterms, while preserving their original syntax.
Each subterm should be recorded exactly as written (e.g., x + y, f (g x), ∀ x : ℕ, x > 0 → P x).
Return the result as a list of strings, where each entry corresponds to a single subterm.
Do not include the theorem statement itself or any explanation and only extract the list of subterms in their original textual form.

Figure 5: Prompt used to extract subterm sequences from Lean 4 statement.

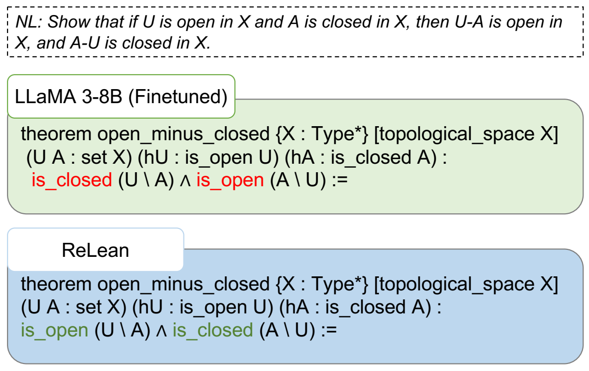
We show two representative prompts that are used to extract structured feedback signals from Lean 4 data. The prompts in Figure 4 instructs the model to translate a formal Lean 4 theorem statement into a natural language sentence. This facilitates the construction of the semantic alignment reward, which measures whether the generated formal statement preserves the intended mathematical meaning. The prompts in Figure 5 asks the model to extract the complete sequence of subterms from a Lean 4 statement, including their arguments and execution order. This serves as the basis for the term alignment reward.

B Case Study

To illustrate the effectiveness of our approach, we present two case studies comparing ReLean with a finetuned LLaMA-3-8B model.

As shown in Figure 6, the task involves formalizing a topological separation property. ReLean correctly assigns the predicates `is_open` and `is_closed` to the appropriate set expressions, while LLaMA-3-8B reverses the polarity, resulting in a semantically incorrect statement.

In Figure 7, the goal is to express the connectedness of an infinite set in the cofinite topology. The baseline fails to identify the correct set and



NL: Show that if U is open in X and A is closed in X, then U-A is open in X, and A-U is closed in X.

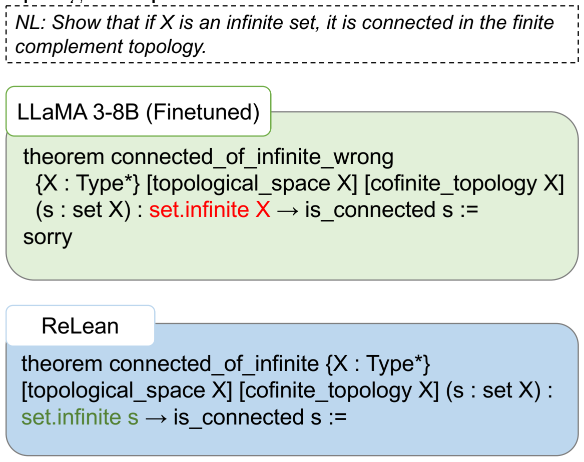
LLaMA 3-8B (Finetuned)

```
theorem open_minus_closed {X : Type*} [topological_space X]
 (U A : set X) (hU : is_open U) (hA : is_closed A) :
  is_closed (U \ A) ∧ is_open (A \ U) :=
```

ReLean

```
theorem open_minus_closed {X : Type*} [topological_space X]
(U A : set X) (hU : is_open U) (hA : is_closed A) :
is_open (U \ A) ∧ is_closed (A \ U) :=
```

Figure 6: Comparison of formalization results for a topological separation task.



NL: Show that if X is an infinite set, it is connected in the finite complement topology.

LLaMA 3-8B (Finetuned)

```
theorem connected_of_infinite_wrong
  {X : Type*} [topological_space X] [cofinite_topology X]
  (s : set X) : set.infinite X → is_connected s :=
sorry
```

ReLean

```
theorem connected_of_infinite {X : Type*}
[topological_space X] [cofinite_topology X] (s : set X) :
set.infinite s → is_connected s :=
```

Figure 7: Comparison of formalization results for a connectedness property in topology.

leaves the statement incomplete, whereas ReLean generates a valid and faithful Lean 4 theorem.

These examples demonstrate ReLean's advantage in capturing both the logical structure and semantic intent of mathematical statements.