Mathesis: Towards Formal Theorem Proving from Natural Languages

Xuejun Yu[1,*], Jianyuan Zhong[2,3,*], Zijin Feng[2,*], Pengyi Zhai[1], Roozbeh Yousefzadeh[2], Wei Chong Ng[1], Haoxiong Liu[2], Ziyi Shou[1], Jing Xiong[2], Yudong Zhou[1], Claudia Beth Ong[1], Austen Jeremy Sugiarto[1], Yaoxi Zhang[1], Wai Ming Tai[1], Huan Cao[1], Dongcai Lu[1], Jiacheng Sun[2], Qiang Xu[3], Xin Shen[1, †], and Zhenguo Li[2, †]

[1]Huawei Celia Team
[2]Huawei Noah's Ark Lab
[3]The Chinese University of Hong Kong

Abstract

Recent advances in large language models show strong promise for formal reasoning. However, most LLM-based theorem provers have long been constrained by the need for expert-written formal statements as inputs, limiting their applicability to real-world problems expressed in natural language. We tackle this gap with Mathesis, the first **end-to-end theorem proving** pipeline processing informal problem statements. It contributes Mathesis-Autoformalizer, the first autoformalizer using reinforcement learning to enhance the formalization ability of natural language problems, aided by our novel LeanScorer framework for nuanced formalization quality assessment. It also proposes a Mathesis-Prover, which generates formal proofs from the formalized statements. To evaluate the real-world applicability of end-to-end formal theorem proving, we introduce Gaokao-Formal, a benchmark of 488 complex problems from China's national college entrance exam. Our approach is carefully designed, with a thorough study of each component. Experiments demonstrate Mathesis's effectiveness, with the autoformalizer outperforming the best baseline by 22% in pass-rate on Gaokao-Formal. The full system surpasses other model combinations, achieving 64% accuracy on MiniF2F with pass@32 and a state-of-the-art 18% on Gaokao-Formal.

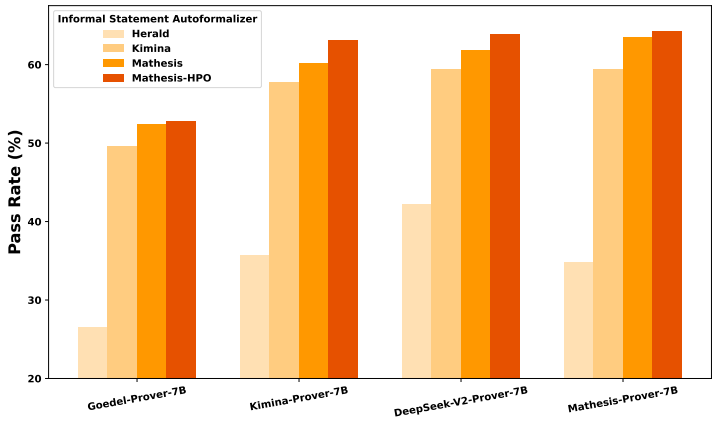

Figure 1: Performance of end-to-end theorem proving on the MiniF2F test set. Each configuration pairs an autoformalizer, which formalizes informal statements to formal ones, with a prover that generates formal proofs. Results reports pass@32 accuracy for all autoformalizer-prover combinations.

* Equal Contribution
† Correspondence to: Zhenguo Li <Li.Zhenguo@huawei.com> and Xin Shen <shenxin19@huawei.com>.

Preprint. Under review.

1 Introduction

The emergence of reasoning abilities in large language models (LLMs) has opened new frontiers in automated mathematics [1]. Recent automatic theorem provers (ATPs) leverage formal verification systems, such as Lean [2], Isabelle [3], and Coq [4], to enable formal reasoning. Formal reasoning starts with a clear formal problem statement, followed by the generation of mechanically verifiable proofs in formal languages. This approach ensures greater reliability and verifiability. Notable models in this field, including Deepseek-Prover-V2 [5], Kimina-Prover [6], and Goedel-Prover [7], have advanced the state-of-the-art on benchmarks: MiniF2F [8], proofNet [9], and putnamBench [10], where the input problem statements are already formalized by human experts.

However, real-world mathematical problems are typically written in natural language, which prohibits the direct use by ATPs. Traditionally, manual formalization ensures faithful translation of problems but requires significant effort and expertise before ATPs can solve them. In this paper, we study automatic **end-to-end theorem proving** from natural languages. The task begins with a given natural language (NL) problem statement and entails its automatic translation into formal language (FL), followed by the generation of a formal proof. A critical step in the end-to-end task is autoformalization–the process of automatically translating informal mathematics into formal language [11]–which can significantly impact the success of proving due to errors introduced during formalization. Figure 2 illustrates two common examples, highlighting how improper formalization can yield misleading proof successes or render problems unprovable.

Despite its importance, the task remains underdeveloped in terms of benchmarking, evaluation, and the availability of a powerful autoformalizer. Existing benchmarks such as MiniF2F are not designed to evaluate autoformalization. Some human-authored formal statements in existing benchmarks subtly deviate from the original NL intent by simplification, while others exclude problems that are hard to formalize, such as those involving geometry, combinatorics[8], and word problems[9]. A fine-grained method to assess autoformalization quality also remains lacking, hindering systematic assessment. Existing works often use MiniF2F, proofNet [9], and Numina [12], as benchmark datasets, relying primarily on binary compilation checks in Lean 4 and basic LLM judgments [7], which fail to capture nuanced errors. In terms of LLM-based autoformalization training, recent methods [13, 11, 14, 7] fine-tune LLMs on paired informal and formal statements (referred to as parallel statements [13, 14]) for higher quality, with Kimina-Autoformalizer [6] achieving state-of-the-art performance via expert iteration. However, these training approaches lack dynamic learning from direct feedback on both syntactic and semantic correctness.

In this paper, we present **Mathesis-Autoformalizer** (**M**ulti-domain **A**utoformalization **T**hrough **He**uristic-guided **S**yntact**i**c and **S**emantic Learning), the first autoformalization framework trained via online reinforcement learning (RL) combined with a novel **Hierarchical Preference Optimization** (**HPO**) mechanism. By incorporating Lean compilation and semantic verification into the RL reward function while learning prover preferences through HPO, Mathesis significantly enhances formalization quality and achieves state-of-the-art performance. To advance end-to-end theorem proving and establish a robust evaluation paradigm for autoformalization, we introduce **Gaokao-Formal**, a benchmark comprising 488 proof problems from China's National Higher Education Entrance Examination (Gaokao) spanning diverse mathematical domains, alongside **LeanScorer**, a novel evaluation framework for nuanced formalization assessment. Furthermore, we develop **Mathesis-Prover** to support end-to-end theorem proving. Our key contributions are as follows.

- We release Mathesis-Autoformalizer, pioneered through online reinforcement learning with rewards for both syntactic validity and semantic correctness, as well as a novel hierarchical preference optimization (HPO) process. It achieves a 22% improvement in pass rate on Gaokao-Formal and 5% on MiniF2F compared to the state-of-the-art baseline Kimina-Autoformalizer.
- We introduce Gaokao-Formal benchmark, a challenging set of 488 Gaokao problems with parallel English and human-verified formal statements, aimed at advancing end-to-end formal reasoning research.
- We propose LeanScorer, a novel evaluation method combining LLM-based analysis with Sugeno Fuzzy Integral for nuanced formalization assessment. It achieves 0.92 F1 score on Gaokao-Formal, outperforming the prior methods LLM-as-a-Judge by 7 percentage points and re-informalization by 27 points.

Informal Statement: Let S_n denote the sum of the first n terms of the sequence a_n. Given that $\frac{2S_n}{n} + n = 2a_n + 1$, prove that $\{a_n\}$ is an arithmetic sequence.

Two Formal Statement Cases:

```
theorem case_goal_as_assumption_error (a : ℕ → ℝ)
(ha : ∃ d, ∀ n, a (n + 1) = a n + d) ⇒ Erroneously includes the desired goal in assumptions
(h : ∀ n, 2 * (∑ i in Finset.range n, a i) / n + n = 2 * a n + 1) :
∃ d, ∀ n, a (n + 1) = a n + d := by sorry

theorem case_definition_error (a : ℕ → ℝ) (S : ℕ → ℝ)
(hS : ∀ (n : ℕ), n ≥ 1, S n = ∑ k in Finset.range n, a k) ⇒ ∑ k in Finset.Icc 1 n, a k
(h : ∀ (n : ℕ), n ≥ 1, 2 * S n / (n : ℝ) + (n : ℝ) = 2 * a n + 1) :
∃ (d : ℝ), ∀ (n : ℕ), n ≥ 1, a (n + 1) = a n + d := by sorry
```

Figure 2: Illustrative example of incorrect formalizations by Kimina-Autoformalizer. The first mistakenly includes the goal as an assumption, resulting in a circular yet technically provable formalization that is mathematically invalid. The second mistranslates the summation range, leading to an incorrect formal statement that is both unprovable and misaligned with the informal input.

- We study end-to-end theorem proving, a critical yet underexplored task for LLM-based formal theorem provers to solve practical mathematical problems directly from natural language. Extensive experiments shows the effectiveness of our end-to-end pipeline, achieving state-of-the-art performance 18% on Gaokao-Formal.

2 Related work

Formal Reasoning Recent advancements have produced powerful LLM-based Automated Theorem Provers for proof assistants like Lean 4, including DeepSeek-Prover-V2 [5], Kimina-Prover [6], and Goedel-Prover [7], and motivated many advanced algorithms for proof search or finding [15–19]. These systems demonstrate strong formal-to-formal (F2F) reasoning capabilities, where both input statements and proofs are in formal language. Correspondingly, prominent benchmarks such as MiniF2F [8] and ProofNet [9] focus on evaluating F2F proving capabilities using already well-formalized problem statements. However, this leaves a significant gap in assessing the critical informal-to-formal (I2F) formalization and the full end-to-end proving pipeline from the informal statement. Our work addresses this by proposing an end-to-end proving pipeline, introducing *Gaokao-Formal*, a diverse benchmark for evaluating I2F and end-to-end performance, and developing *LeanScorer* for nuanced I2F evaluation. While some other works [20–22], such as Lego-Prover [21], also target I2F, require an additional informal proof sketch as input. In contrast, our work focuses on fully automatic end-to-end proving, starting from informal statements to whole-proof generation [23], making direct comparisons inappropriate.

Autoformalization Autoformalization, the process of formalizing informal mathematics into formal language [24, 13], is essential for bridging the NL-FL gap. Prior work includes prompting pre-trained LLMs [24, 9] and fine-tuning models on static NL-FL pairs [13, 11, 14], with systems like Kimina-Autoformalizer [6] achieving notable success. However, these approaches often lack dynamic learning from direct feedback on syntactic and semantic correctness, and their evaluation has typically relied on binary compilation checks or basic LLM judgments [7, 14], which may not capture nuanced errors. To address these limitations, we introduce *Mathesis-Autoformalizer*, which, to our knowledge, is the first autoformalizer for Lean to leverage online reinforcement learning for improved accuracy and robustness. Concurrently, our *LeanScorer* framework provides a more fine-grained evaluation of autoformalization quality, moving beyond simple pass/fail metrics.

Reinforcement Learning Fine-Tuning (RLFT) Reinforcement learning has proven highly effective for enhancing LLM capabilities in complex reasoning [25, 26]. The autoformalization task is well-suited for RL, as syntactic validity (from a Lean verifier) and semantic equivalence (e.g., assessed by an LLM judge or *LeanScorer*) can serve as direct reward signals. Despite this clear potential, the application of outcome-based RL techniques to specifically optimize these syntactic and semantic properties in autoformalization models has been largely underexplored in the literature. Our *Mathesis-Autoformalizer* pioneers this direction by employing Group Relative Policy Optimization [27] with a carefully designed composite reward function (in Section 3.1). This approach allows the model to iteratively refine its ability to generate syntactically correct and semantically faithful formalizations, addressing a key gap in existing autoformalization methodologies.

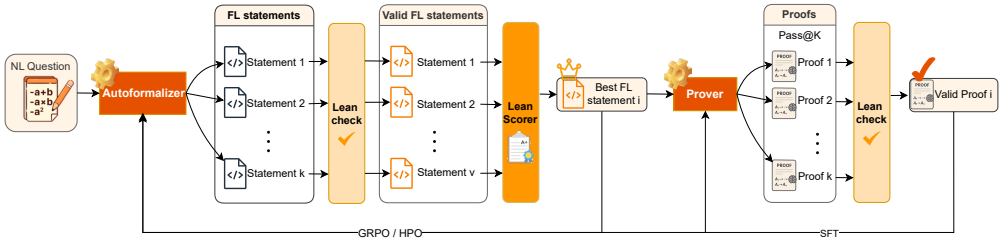

Figure 3: Overview of our end-to-end theorem proving pipeline.

3 End-to-End Theorem Proving

The core objective of this paper is to tackle the **end-to-end theorem proving** task (defined in Section 1) by enabling automated theorem proving directly from informal natural language inputs. To this end, as illustrated in Figure 3, we propose a structured, multi-stage pipeline consisting of three major stages: autoformalization, validation, and proving. The pipeline begins with a given NL problem statement, which is first processed by an autoformalizer to generate a certain number of candidate formal statements in Lean 4. This stage is handled by our *Mathesis-Autoformalizer*, a model that performs best-in-class formalizations (to be described in Section 3.1). These candidates are then subjected to the validation stage, which includes a syntactic check by the Lean compiler and semantic assessment by an LLM-based scoring system. For this purpose, we introduce the *Gaokao-Formal* benchmark and the *LeanScorer* evaluation framework (described in Section 3.2). The formal statement that passes the Lean compiler and receives the highest semantic assessment score is then passed to an automated theorem prover and generates a complete, machine-verifiable proof in Lean.

3.1 *Mathesis-Autoformalizer*: Advancing Autoformalization with Reinforcement Learning

The cornerstone of our end-to-end theorem proving pipeline (Figure 3) is *Mathesis-Autoformalizer*, a novel system designed to translate informal mathematical problem statements from NL into formal Lean 4 [2] code. Unlike prior autoformalization approaches that predominantly rely on supervised fine-tuning (SFT) on static datasets, *Mathesis-Autoformalizer* leverages reinforcement learning (RL) to dynamically improve its translation accuracy based on direct feedback regarding both the syntactic validity and semantic correctness of its generated formalizations. GRPO was chosen over other RL methods due to its efficacy in optimizing policies based on the relative quality of multiple candidate generations and its mechanism for stable learning by regularizing against a reference policy. To our knowledge, it is the first autoformalization model for Lean employing Group Relative Policy Optimization (GRPO) [27] for this purpose.

3.1.1 Composite Rewards for Autoformalization

Let π_θ represents the translator LLM policy parameterized by θ, and π_{ref} be a fixed reference policy (typically the SFT model). For a given natural language input $x \in \mathcal{X}$, the policy $\pi_\theta(\cdot|x)$ generates a group of G candidate formal Lean 4 statements (outputs) $\{o_1, ..., o_G\}$. The optimization objective is to adjust θ to maximize the likelihood of generating higher-reward outputs relative to lower-reward ones within the group, while regularizing against large deviations from the reference policy.

We construct a composite reward function based on two criteria. For a generated statement o_i derived from input x:

Semantic Equivalence Reward (R_{sem}) This binary reward assesses whether o_i preserves the semantic meaning of x, determined by an auxiliary LLM judge (J_{sem}).

$$R_{sem}(x, o_i) = \begin{cases} 1 & \text{if } J_{sem}(x, o_i) \text{ judges "Appropriate"} \\ 0 & \text{otherwise} \end{cases}. \tag{1}$$

Verification Reward (R_{ver}) This binary reward indicates whether o_i is syntactically correct and type-valid via a Lean 4 verifier (V_{lean}), checked up to `:= by sorry`.

$$R_{ver}(o_i) = \begin{cases} 1 & \text{if } V_{lean}(o_i) \text{ succeeds} \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

The overall reward r_i for an output o_i is computed as a combination of these components: $r_i = R(x, o_i, o_{ref}) = R_{sem} + R_{ver}$. We find that this simple yet effective summation strategy indeed leads to state-of-the-art performance. We then leverage the GRPO objective to update $\pi_\theta(\cdot|x)$.

Training Data Curation To effectively identify samples for training, our data curation process employs topic modeling with BERTopic [28] to the natural language informal statements of problems from a pset [7] and our in-house Gaokao dataset. BERTopic was selected for its ability to generate coherent topics by leveraging contextual embeddings and clustering, allowing for effective categorization of problems based on their semantic content; this approach is also significantly faster and more cost-effective than using large language models for the same categorization task. We generated embeddings for each statement, performed dimensionality reduction and clustering, and then mapped the resulting topics to predefined mathematical categories in Section 4. The categorized data from two sources were then merged. To optimize Reinforcement Learning (RL) training efficiency with (GRPO), we employed our base model (pre-RL) to perform rollouts (k=14) on each problem, filtering out those yielding rewards with zero standard deviation across the rollouts. The remaining problems demonstrating reward variance were combined with 8,000 problems randomly sampled from the Lean Workbook [29], resulting in a final training dataset of approximately 32k problems.

3.1.2 Hierarchical Preference Optimization for End-to-End Theorem Proving

GRPO provides a reward-maximizing initialization, aligning the autoformalizer with local objectives of syntactic correctness and semantic validity. In the context of end-to-end theorem proving, the autoformalizer formalizes natural language into formal statements aimed at facilitating successful proof generation by the prover. To enhance this, we further fine-tune the autoformalizer using Direct Preference Optimization (DPO) [30], where preferences are derived from the global success of the downstream proof generation. We refer to this combined approach as *Hierarchical Preference Optimization*, which integrates local alignment via GRPO and global alignment via DPO.

DPO Training Data Generation During the data generation phase, for each natural language statement x, a group of candidate formal statements y^i are sampled from the autoformalizer $\pi_\theta(\cdot|x)$, where i indexes the candidates. Each y^i undergo syntactic and semantic validation, and those that pass are forwarded to the prover, which attempts to generate proofs z^i. Preferences are assigned based on the successful completion of the proof verified by Lean, yielding data tuple $\{x, y_w^i, z_w^i\}$ for successful cases, and $\{x, y_l^i, z_l^i\}$ for failed attempts.

DPO Training The training configuration employs a single epoch with a learning rate of 1×10^{-5}. The KL regularization coefficient β is set to 0.1, penalizing deviations from the reference model. Optimization is applied to full parameters, with a warmup ratio of 0.05. To manage memory usage efficiently, training is conducted using DeepSpeed zero3 offload.

DPO fine-tuning enhances alignment with task-grounded outputs, thereby mitigating mismatches between reward function and actual task objectives. Compared to GRPO, DPO is a more sample-efficient and stable alternative that performs offline preference learning and eliminates the need for a separate reward model [30, 31]. However, its effectiveness heavily relies on a strong base model to generate meaningful candidate outputs and better exploit preference signals [32–34]. To this end, we first apply GRPO to establish a strong initialization before proceeding with DPO fine-tuning.

3.2 *LeanScorer*: Nuanced Evaluation of Autoformalization Correctness

Standard evaluations of auto-formalized statements often rely on binary syntactic checks or basic semantic judgments. To enable a more fine-grained and reliable assessment, we propose *LeanScorer*, a novel framework yielding a continuous correctness score (0 to 1). This score allows for dynamic adjustment of decision thresholds according to task-specific precision/recall requirements. *LeanScorer* employs a two-stage process (as shown in Figure 4):

Subtask Decomposition and LLM-based Evaluation An LLM first deconstructs the NL problem into discrete subtasks (premises and conclusions). Another LLM then evaluates the formalization of each subtask by comparing it to the original NL expression, assigning one of three ratings: "Perfectly Match", "Minor Inconsistency", or "Major Inconsistency". The three-tiered rating system introduces necessary flexibility for borderline cases, avoiding the excessive rigidity of binary classification. Our empirical observations demonstrate clear differentiation patterns: the model consistently labels natural language-formal language (NL-FL) pairs with mathematical inequivalence or missing condi-

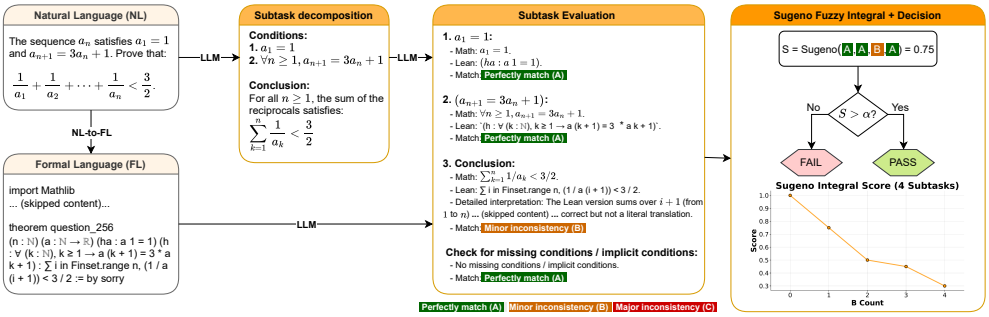

Figure 4: Overview of our LeanScorer evaluation method.

tions as "Major Inconsistency," while classifying cases with mathematical equivalence but divergent expressions or detailed math difference (e.g., omitted function domain definitions) as "Minor Inconsistency." This intermediate category also serves to accommodate potential LLM evaluation errors. The three-tiered rating system thereby capitalizes on the LLM's capacity for nuanced assessment while minimizing evaluation bias. The prompts for the two stages are demonstrated in appendix B.5.

Sugeno Fuzzy Integral Aggregation These categorical ratings are aggregated into a composite score using the Sugeno Fuzzy Integral [35]. As a useful method in multi-criteria decision-making (MCDM) [36], the Sugeno Fuzzy Integral provides a robust framework for aggregating multiple categorical ratings into a composite score. By designing the fuzzy measure, we enable the system to tolerate subtle inaccuracies in the LLM's judgments while enforcing strict criteria: it rejects any formalization containing "Major Inconsistency", awards full marks for formalization with all "Perfectly Match", and imposes incremental penalties for multiple "Minor Inconsistency". This approach balances flexibility with rigorous standards, accounting for the probabilistic nature of LLMs while ensuring robust formalization quality.

Let $N = \{0, 1, 2, ..., n-1\}$ denote the set of subtasks, with n tasks in total. Let $E = \{e_0, e_1, ..., e_{n-1}\}$ denote the evaluation set, where $e_i \in \{A, B, C\}$ (for simplicity, we use A represents *Perfectly Match*, B represents *Minor Inconsistency* and C represents *Major Inconsistency*). Let $f : E \to [0, 1]$ be an evaluation mapping function, in our experiments, we set $f(A) = 1.0$, $f(B) = 0.5$, $f(C) = 0$. Our fuzzy measure $\mu(M)$ for a subset of tasks $M \subseteq N$, where N is the set of n subtasks is designed to handle uncertainty and inter-task interactions, notably by applying progressive penalties for multiple "Minor Inconsistencies" and a veto mechanism where any "Major Inconsistency" yields a zero score. A perfect score of 1 requires all subtasks to be "Perfectly Match." The fuzzy measure is defined as:

$$\mu(M) = \begin{cases} 1 & \text{if } M = N \text{ and } \forall i \in M, e_i = A \\ \max\{\frac{|M_A|}{n} \cdot (1 - 0.2 \cdot |M_B|), 0\} & \text{if } |M_B| \geq 2 \\ \frac{|M_A|}{n} \cdot (1 - 0.1 \cdot |M_B|) & \text{if } |M_B| = 1 \\ 0 & \text{if } \exists i \in M, e_i = C \end{cases}, \tag{3}$$

where A, B, C represent the ratings, and $|M_A|, |M_B|$ are counts of 'A' and 'B' ratings in M. The overall LeanScore $S(f) = \max_{i=1}^{n} \min(f(e_{\pi(i)}), \mu(A_i))$, for sorted evaluations $f(e_{\pi(i)})$ and corresponding index sets A_i. Users can decide on a decision threshold $0 < \alpha < 1$ when it is necessary to perform binary classification.

3.3 *Mathesis-Prover*

Our formal mathematical reasoning framework, *Mathesis-Prover*, builds upon the Qwen2.5-Math-7B architecture [37] through an expert iteration paradigm. This approach establishes a self-improving cycle where the prover enhances its capabilities by learning from both initial training examples and newly verified solutions to formalized problems.

The training process for *Mathesis-Prover* proceeds through iterative refinement cycles. In each iteration, we employ the current prover model (we employ DeepSeek-Prover-V2-7B [5] in the first round) to generate complete proof attempts for questions from open-source datasets Goedel-Pset-v1 [7]. To ensure appropriate training difficulty, we specifically include problems that cannot be solved within four attempts (pass@4) by Goedel-Prover but are successfully verified when processed by our trained prover model using the Lean 4 proof assistant.

Benchmark		Problem
MiniF2F	NL	If x and y are positive integers for which $2^x 3^y = 1296$, prove that $x+y = 8$.
	FL	`theorem amc12b_2004_p3` (x y : $\mathbb{N}$) (h₀ : 2 ^ x * 3 ^ y = 1296) : x + y = 8 := by sorry
Gaokao-Formal	NL	Let m be a positive integer, and let $a_1, a_2, \cdots, a_{4m+2}$ be an arithmetic sequence with a non-zero common difference. If two terms a_i and a_j $(i < j)$ are removed from the sequence such that the remaining $4m$ terms can be evenly divided into m groups, and each group of 4 numbers forms an arithmetic sequence, then the sequence $a_1, a_2, \cdots, a_{4m+2}$ is called an $(i, j)-$separable sequence. For $m \geq 3$, prove that the sequence $a_1, a_2, \cdots, a_{4m+2}$ is a $(2, 13)-$separable sequence
	FL	`theorem gaokaoformal_g4` (m : $\mathbb{N}$) (hm : $1 \leq$ m) (a : $\mathbb{N} \to \mathbb{R}$) (ha: $\exists$ (d:$\mathbb{R}$), d $\neq$ 0 $\wedge$ ($\forall$ (n:$\mathbb{N}$), (n$\geq$ 1 $\wedge$ n $\leq$ 4*m+1) $\to$ a (n+1) = a n + d)) (sep: ($\mathbb{N} \times \mathbb{N}$) $\to$ Prop) (h_sep: $\forall$ (i j:$\mathbb{N}$), (i$\geq$ 1 $\wedge$ i<j $\wedge$ j $\leq$ 4*m+2) $\to$ sep (i,j) = ($\exists$ (f : $\mathbb{N} \to \mathbb{N}$), ($\forall$ (h:$\mathbb{N}$), (h $\geq$ 1 $\wedge$ h $\leq$ 4*m+2 $\wedge$ h$\neq$i $\wedge$ h$\neq$j) $\to$ (f h $\geq$ 1 $\wedge$ f h $\leq$ m)) $\wedge$ ($\forall$ (g:$\mathbb{N}$), let S := h:$\mathbb{N}$ | h $\geq$ 1 $\wedge$ h $\leq$ 4*m+2 $\wedge$ h $\neq$ i $\wedge$ h $\neq$ j $\wedge$ f h = g; (g $\geq$ 1 $\wedge$ g $\leq$ m) $\to$ (Nat.card S = 4 $\wedge$ ($\exists$ (p: $\mathbb{N} \to$ S), ($\forall$ (k l:$\mathbb{N}$), (k $\geq$ 1 $\wedge$ k $\leq$ 4 $\wedge$ l $\geq$ 1 $\wedge$ l $\leq$ 4 $\wedge$ k $\neq$ l) $\to$ p k $\neq$ p l) $\wedge$ ($\exists$ (d':$\mathbb{R}$), $\forall$ (k:$\mathbb{N}$), (k$\geq$ 1 $\wedge$ k $\leq$ 3) $\to$ a (p (k+1)) = a (p k) + d')))))) : m $\geq$ 3 $\to$ sep (2,13) := by sorry

Figure 5: Comparison of the complexity of the problems in MiniF2F v.s. Gaokao-Formal

4 The Gaokao-formal benchmark

To advance autoformalization and end-to-end theorem proving from natural language, we introduce the *Gaokao-Formal* benchmark. Unlike existing benchmarks that focus primarily on formal-to-formal proving or simplifying complexities of the NL mathematical problems, *Gaokao-Formal* specifically targets the difficulties of auto-formalizing diverse and intricate NL mathematical statements. This benchmark consists of 488 proof problems from China's National Higher Education Entrance Examination (Gaokao, 2008-2024), often segmented into sub-questions. Each instance includes the original Chinese problem statement, an English translation, and a human-expert verified Lean 4 formal statement. A comparative overview is provided in Table 2, highlighting its key distinctions in terms of domain coverage and task focus.

Table 2: Comparison of Gaokao-Formal with Existing Mathematical Reasoning Benchmarks.

	Gaokao-Formal	**MiniF2F**	**ProofNet**	**PutnamBench**
Inclusion Geometry	**Yes**	Limited	Less Emphasis	Yes
Inclusion Comb.	**Yes**	Limited	Less Emphasis	Yes
Complex Formalization	**Included**	Excluded	Excluded	N/A
Primary Task Focus	**End2End**	Proving	Autoformal. & Proving	Proving
Size (Problems/Stmts)	488	244	371	645

Problem-Type Diversity Unlike benchmarks that may filter out problem types with less developed theorem libraries (e.g., geometry, combinatorics), *Gaokao-Formal* includes all such problems as they appear in the Gaokao exams. This encourages broader model capabilities and contributes to the expansion of Lean 4's Mathlib [2]. The problems are categorized into seven domains (functions, sequences/series, inequality, trigonometry, analytic geometry, probability/combinatorics, and comprehensive questions) to facilitate targeted evaluation.

Table 1: Gaokao-Formal Benchmark Category Summary.

Category	**Number of Questions**
Functions	167
Sequences and series	150
Inequality	28
Trigonometry	22
Analytic geometry	71
Probability and combinatorics	4
Comprehensive questions	46

Autoformalization Complexity Many existing benchmarks simplify or exclude problems where the primary challenge lies in the NL-to-Formal Language (NL2FL) translation. *Gaokao-Formal* deliberately retains these, especially in its "comprehensive questions" category, which features problems with multi-domain concepts, novel definitions, or complex linguistic structures, thereby

rigorously testing LLM abstraction capabilities. We provide an example of this kind of question, comparing it with one MiniF2F question in Figure 5.

5 Experiments

We conduct a series of experiments to validate our contributions. First, we evaluate the reliability of our proposed semantic consistency checker, *LeanScorer*. Second, we assess the performance of our autoformalization model, *Mathesis-Autoformalizer*, against state-of-the-art baselines. Finally, we measure the impact of improved autoformalization on the success rate of the complete end-to-end theorem proving pipeline.

5.1 Experimental Setup

Datasets We primarily use our newly proposed *Gaokao-Formal* benchmark (Section 4) for evaluating autoformalization from complex NL and end-to-end proving. We also report results on the widely used MiniF2F-test set [8] for comparison.

Models Training details of the both our autoformalizer and prover are in A. We evaluate our autoformalization models, *Mathesis-Autoformalizer*, against strong baselines including API-based models (Claude-3.5, GPT-o3-mini, GPT-4o, Doubao-1.5, Gemini-2.0, Deepseek-V3, Deepseek-R1) and open-source fine-tuned models (Herald-7B [11], Kimina-7B [6]). For end-to-end evaluation, we use several provers: Goedel-Prover-SFT-7B [7], Kimina-Prover-Preview-Distill-7B [6], DeepSeek-Prover-V2 (7B version) [5], and our *Mathesis-Prover*. Auxiliary models like Deepseek-V3 [38] are used for LLM-as-a-Judge and re-informalization baselines.

Evaluation Metrics We assess autoformalization quality using: (1) Lean Check (LC), measuring syntactic validity (pass@k); (2) *LeanScorer* Semantic Check (LSC), providing a nuanced semantic assessment. The *LeanScorer* is evaluated via Precision, Recall, and F1 against human labels. Success rate@k, defined in Equation B.1, is used for LC and LSC over k samples. End-to-end performance is measured by the proving success rate of a prover given an auto-formalized statement.

Implementation Details For autoformalization quality assessment (Table 4), we report success rate@k for $k = 1$ and $k = 6$. For end-to-end proving (Table 5), we use a search budget of 32 attempts per problem for all provers as it is a reasonable budget for real applications. Experiments were conducted on relevant hardware resources. Prompts are detailed in Appendix B.5.

5.2 Evaluating Semantic Consistency Checkers

Goal Reliable semantic evaluation is crucial for developing and assessing autoformalization models beyond simple syntactic checks. We first validate the effectiveness of *LeanScorer* compared to common baseline methods.

Table 3: Performance Comparison of Semantic Consistency Checkers. The evaluation measures how faithfully each checker's predicted labels align with human-annotated ground truth on the Gaokao dataset. Precision and recall are reported in percentages (%).

Method	Precision (%)	Recall (%)	F1
LLM-as-a-Judge	73	**100**	0.85
Re-informalization	93	50	0.65
LeanScorer (Ours)	**94**	89	**0.92**

Methodology We compared *LeanScorer* (with threshold setting as $\alpha = 0.6$) with other "ground truth free" semantic check methods: a standard LLM-as-a-Judge approach [7], and a Re-informalization & Semantic Similarity baseline [29], against human annotations on a subset of *Gaokao-Formal* (with some wrong LLM formalized statement kept for evaluation). All the LLM roles utilize Deepseek-V3 with prompts in Appendix B.5. Performance was measured using Precision, Recall, and F1 score.

Results & Analysis The results in Table 3 compellingly demonstrate *LeanScorer*'s superior ability to discern semantic consistency in autoformalization. Achieving a remarkable 94% precision, 89% recall, and F1 0.92 scores, *LeanScorer* significantly outstrips both the LLM-as-a-Judge (F1 0.85) and the Re-informalization baseline (F1 0.65). While the LLM-as-a-Judge exhibits perfect recall, its lower precision indicates a higher rate of false positives. Conversely, Re-informalization, despite its high precision, suffers from low recall, missing a substantial number of inconsistencies. *LeanScorer*'s balanced and high performance across both precision and recall underscores its effectiveness as a

robust and reliable tool for nuanced semantic evaluation, crucial for advancing the development of accurate autoformalization models.

Table 4: Quality assessment of the formalized statement generated by a single model. Each column highlights one top score for sample budget 1 (bold) and one for sample 6 (underlined).

Model	Sample	MiniF2F-Test		Gaokao-Formal	
	Budget	LC	LC+LSC	LC	LC+LSC
API Models					
Claude-3.5	1	71%	56%	41%	30%
	6	74%	69%	58%	49%
GPT-o3-mini	1	58%	45%	38%	25%
	6	87%	77%	70%	54%
GPT-4o	1	50%	36%	20%	13%
	6	80%	65%	48%	28%
Doubao-1.5	1	48%	40%	19%	15%
	6	77%	70%	45%	32%
Gemini-2.0	1	56%	41%	36%	22%
	6	80%	71%	66%	47%
Deepseek-V3	1	76%	61%	54%	36%
	6	91%	84%	69%	56%
Deepseek-R1	1	54%	44%	45%	30%
	6	86%	76%	81%	57%
Open-Source Models					
Herald-7B	1	80%	41%	56%	14%
	6	95%	69%	78%	27%
Kimina-7B	1	83%	61%	50%	21%
	6	**100**%	91%	91%	49%
Ours					
Mathesis-Autoformalizer	1	92%	69%	88%	45%
	6	**100**%	95%	**98**%	67%
Mathesis-Autoformalizer-HPO	1	**99**%	**79**%	**93**%	**50**%
	6	**100**%	**96**%	**98**%	**71**%

5.3 Autoformalization Performance

Goal We evaluated the core performance of our *Mathesis-Autoformalizer* in formalizing NL problems into syntactically correct and semantically faithful Lean 4 statements, comparing with SOTA models.

Methodology We compared *Mathesis-Autoformalizer* against various API and open-source baseline autoformalizers on the MiniF2F-test and *Gaokao-Formal* benchmarks. Performance was measured using Lean Check success rate@k (LC@k) and the combined Lean Check + *LeanScorer* Semantic Check success rate@k (LC+LSC@k) for k=1 and k=6.

Results & Analysis Table 4 presents the detailed autoformalization results. Our *Mathesis-Autoformalizer-HPO* models consistently achieve state-of-the-art performance, particularly on the challenging *Gaokao-Formal* benchmark. For instance, *Mathesis-Autoformalizer* achieves an LC+LSC@6 score of 71%, representing a substantial improvement over the strong Kimina baseline's 49%. This constitutes a 22% absolute increase and a relative increase of approximately 44.9%, validating the effectiveness of our GRPO- and HPO-based reinforcement learning approach combined with the composite reward function in generating both syntactically valid and semantically accurate formalizations. The results highlight the advantage of dynamic learning from semantic and syntactic feedback, especially for complex NL problems prevalent in *Gaokao-Formal*.

5.4 End-to-End Theorem Proving Performance

Goal The ultimate test is the impact of the autoformalizer on the success rate of the complete end-to-end NL-to-Proof pipeline. We evaluate how using formal statements generated by *Mathesis-Autoformalizer* affects the performance of downstream theorem provers.

Table 5: Performance of different provers on formal problem statements generated by various autoformalizers from natural language inputs. Underlines (bold) indicate top (second) scores per column among LLM-based autoformalizers.

Model	Autoformalizer	MiniF2F-Test		Gaokao-Formal	
		Search Budget	Accuracy	Search Budget	Accuracy
Goedel-Prover-SFT-7B	Human	32	57.6%	32	10.0%
	Herald	32	26.6%	32	5.7%
	Kimina	32	49.6%	32	8.8%
	Mathesis	32	52.4%	32	11.2%
	Mathesis-HPO	32	52.8%	32	11.6%
Kimina-Prover-Preview-Distill-7B	Human	32	63.1%	32	12.1%
	Herald	32	35.7%	32	6.3%
	Kimina	32	57.8%	32	11.9%
	Mathesis	32	60.2%	32	13.9%
	Mathesis-HPO	32	63.1%	32	16.1%
DeepSeek-Prover-V2-7B (non-COT)	Human	32	68.0%	32	15.2%
	Herald	32	42.2%	32	7.2%
	Kimina	32	59.4%	32	11.1%
	Mathesis	32	61.9%	32	14.9%
	Mathesis-HPO	32	**63.9**%	32	**16.8**%
Mathesis-Prover-7B	Human	32	69.2%	32	13.7%
	Herald	32	34.8%	32	7.2%
	Kimina	32	59.4%	32	11.7%
	Mathesis	32	63.5%	32	15.6%
	Mathesis-HPO	32	**64.3**%	32	**18.0**%

Methodology We generated formal statements for MiniF2F-test and *Gaokao-Formal* using different autoformalizers: Human-provided, Herald, Kimina, and our *Mathesis-Autoformalizer*. These formal statements were then fed as input to various provers: Goedel-Prover-SFT-7B, Kimina-Prover-Preview-Distill-7B, DeepSeek-Prover-V2-7B non-COT, and our *Mathesis-Prover*. We measured the proving success rate (Accuracy) with a fixed search budget of 32.

Results & Analysis From Table 5, we draw two key observations. First, our end-to-end pipeline incorporating both Mathesis-Autoformalizer and Mathesis-Prover achieves pioneer performance among 7B parameter provers and existing autoformalizers, attaining success rates of 64.3% on MiniF2F and 18.0% on Gaokao-Formal. These results not only demonstrate the effectiveness of our GRPO-HPO training strategy but also validate Gaokao-Formal as a rigorous benchmark for evaluating end-to-end formal reasoning systems.

Second, our analysis reveals significant performance gains from component improvements. On MiniF2F, upgrading the prover yields an 11.5% absolute improvement (from 52.8% to 64.3%), while enhancing the autoformalizer produces a more substantial 29.5% gain (from 34.8% to 64.3%). The Gaokao-Formal benchmark, which presents greater formalization challenges, shows similar trends: prover improvements lead to a 6.4% increase (11.6% to 18.0%), while autoformalizer enhancements nearly double performance with an 11.2% improvement (7.2% to 18.4%). These comparative results underscore the critical role of high-quality autoformalization in end-to-end theorem proving systems.

It was observed that the model occasionally surpassed human-written formalizations in the Gaokao-Formal benchmark. This observation can be attributed to two key factors. First, in the benchmark construction, human formalizations rigorously stated all premises with maximal precision, for instance, explicitly declaring domains when defining functions or sequences (see Figure 5). In contrast, LLM-generated formalizations may not be in this style which leads to provers' training data bias so that provers show difficulty proving these questions. Second, while LeanScorer achieves 94% verification precision (Section 5.2), residual false positives in the verification pipeline could partially account for this performance discrepancy.

Additionally, we identified edge cases where autoformalizers and provers circumvented Lean syntactic checks. For instance, both Deepseek-prover-v2 and Mathesis-Prover occasionally produced proofs ending with "apply?", which were technically verifiable but lacked substantive reasoning; these were excluded from success metrics. Similarly, Kimina-Autoformalizer and Mathesis-Autoformalizer generated statements concluding with ": True := by sorry", where the proof goal merely restated a

premise. While syntactically valid and occasionally passing LeanScorer's semantic check (due to the presence of conditions and conclusions), such formulations failed to capture the original problem's logical structure and were thus excluded from end-to-end evaluation.

6 Real-World Application: Gaokao Problem Solving System in Huawei Celia

In this section, we present a practical implementation of our end-to-end formal reasoning framework through its integration into **Huawei Celia**, the AI assistant for Huawei products. We demonstrate how combining formal and informal reasoning can enhance model reasoning performance on China's National Higher Education Entrance Examination (Gaokao). Gaokao is the most crucial academic assessment in China that determines university admissions for millions of students annually. Gaokao questions demand advanced logical reasoning and mathematical proficiency, making them an ideal testbed for evaluating how formal reasoning can augment LLM capabilities.

As Gaokao problems typically consist of multiple interrelated sub-questions (which exceed the capacity of standalone provers), our system first employs an LLM to automatically decompose each question into atomic sub-problems. These are then processed through our end-to-end theorem-proving pipeline (Section 3). To further enhance performance, we developed an interactive formal-informal framework that enables bidirectional communication between formal and informal reasoning components. We integrated both pipelines (the end-to-end theorem-proving system and this interactive framework) into Huawei's Celia AI assistant. Our comprehensive evaluation employs Gaokao problems from the past five years, comparing three approaches: (1) baseline informal reasoning using the state-of-the-art DeepSeek-R1 (671B) model, (2) our end-to-end theorem-proving pipeline, and (3) the complete Celia Gaokao problem-solving system incorporating both the theorem-proving pipeline and interactive framework.

From the segmented sub-questions, we identified 95 proof-related items, categorized in Table 6. Figure 6 demonstrates that the end-to-end theorem-proving pipeline showed particular strength in Functions and Inequality problems, and the full Celia system achieved improvements across all categories. The overall accuracy increased from a baseline of 65.3% to 69.4% using just the theorem-proving pipeline (with Mathesis-Autoformalizer and Mathesis-Prover), while the complete Celia system reached 84.2% accuracy. By case studies, we found that informal reasoning often fails in cases involving complex logical inferences or intricate mathematical computations, and occasionally, the model does not employ sufficiently rigorous reasoning methods (e.g., enumerating specific cases rather than providing a proper proof). An example in Appendix B.4 highlights a scenario in which informal reasoning leads to logical errors, whereas formal reasoning efficiently resolves the problem using Lean 4's built-in tactic `nlinarith`. Our observations suggest that formal reasoning can significantly enhance the logical and computational rigor of LLMs. Future research could explore whether formal reasoning capabilities can be leveraged to augment the informal reasoning ability of models. Additionally, improving the readability of formal reasoning represents another critical direction for future investigation.

7 Limitation

While the proposed end-to-end pipeline demonstrates significant progress, there remains considerable room for improvement. On one hand, each of the three components—autoformalization, theorem proving, and scoring and evaluation, can be further refined. On the other hand, future work could integrate these capabilities into a unified model, extending its scope from natural language input to formal language proof generation. We leave this direction for future research and anticipate further advancements in this field.

8 Conclusion

This paper investigates the potential of large language models in formal reasoning and proposes a pipeline to enhance this specific capability into an end-to-end framework for real-world applications. By training the Mathesis-Autoformalizer and the Mathesis-Prover, as well as designing the LeanScorer, we advance end-to-end formal reasoning to achieve state-of-the-art performance. Additionally, this work introduces the Gaokao-Formal benchmark, which highlights the challenges of autoformalization and promotes formal reasoning across diverse domains. This paper investigates the potential of large language models in formal reasoning and proposes a pipeline to enhance this specific capability into an end-to-end framework for real-world applications. By training the Mathesis-

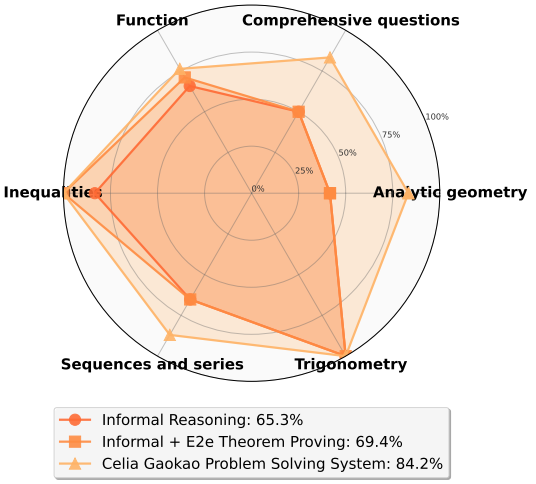

Figure 6: Performance improvement by category.

Table 6: Sub-questions by Category

Category	**Total Questions**
Analytic geometry	12
Comprehensive questions	6
Function	38
Inequalities	12
Sequences and series	23
Trigonometry	4
Total	**95**

Autoformalizer and the Mathesis-Prover, as well as designing the LeanScorer, we advance end-to-end formal reasoning to achieve state-of-the-art performance. Additionally, this work introduces the Gaokao-Formal benchmark, which highlights the challenges of autoformalization and promotes formal reasoning across diverse domains.

References

[1] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in AI. In *Proceedings of the International Conference on Machine Learning*, 2025.

[2] The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20. ACM, January 2020.

[3] Lawrence C Paulson. *Isabelle: A generic theorem prover*. Springer, 1994.

[4] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant a tutorial. *Rapport Technique*, 178:113, 1997.

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

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

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

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

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

[10] George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *arXiv preprint arXiv:2407.11214*, 2024.

[11] Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. *arXiv preprint arXiv:2410.10878*, 2024.

[12] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

[13] Albert Q Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*, 2023.

[14] Xiaoyang Liu, Kangjie Bao, Jiashuo Zhang, Yunqi Liu, Yu Chen, Yuntian Liu, Yang Jiao, and Tao Luo. Atlas: Autoformalizing theorems through lifting, augmentation, and synthesis of data. *arXiv preprint arXiv:2502.05567*, 2025.

[15] Zhenwen Liang, Linfeng Song, Yang Li, Tao Yang, Feng Zhang, Haitao Mi, and Dong Yu. Mps-prover: Advancing stepwise theorem proving by multi-perspective search and data curation. *arXiv preprint arXiv:2505.10962*, 2025.

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

[17] Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.

[18] Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*, 2025.

[19] Xiao-Wen Yang, Zhi Zhou, Haiming Wang, Aoxue Li, Wen-Da Wei, Hui Jin, Zhenguo Li, and Yu-Feng Li. Carts: Advancing neural theorem proving with diversified tactic calibration and bias-resistant tree search. In *The Thirteenth International Conference on Learning Representations*, 2025.

[20] Xueliang Zhao, Wenda Li, and Lingpeng Kong. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*, 2023.

[21] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*, 2023.

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

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

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

[25] Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. Model Card, March 2024.

[26] DeepSeek-AI and Anonymous Contributors. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

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

[28] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[29] Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *arXiv preprint arXiv:2406.03847*, 2024.

[30] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

[31] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[32] Songjun Tu, Jiahao Lin, Xiangyu Tian, Qichao Zhang, Linjing Li, Yuqian Fu, Nan Xu, Wei He, Xiangyuan Lan, Dongmei Jiang, et al. Enhancing llm reasoning with iterative dpo: A comprehensive empirical investigation. *arXiv preprint arXiv:2503.12854*, 2025.

[33] Tianduo Wang, Shichen Li, and Wei Lu. Self-training with direct preference optimization improves chain-of-thought reasoning. *arXiv preprint arXiv:2407.18248*, 2024.

[34] Junshu Pan, Wei Shen, Shulin Huang, Qiji Zhou, and Yue Zhang. Pre-dpo: Improving data utilization in direct preference optimization using a guiding reference model. *arXiv preprint arXiv:2504.15843*, 2025.

[35] Michio Sugeno. Theory of fuzzy integrals and its applications. *Doctoral Thesis, Tokyo Institute of Technology*, 1974.

[36] Jonata Wieczynski, Giancarlo Lucca, Eduardo Borges, Asier Urio-Larrea, Carlos López Molina, Humberto Bustince, and Graçaliz Dimuro. Application of the sugeno integral in fuzzy rule-based classification. *Applied Soft Computing*, 167:112265, 2024.

[37] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[38] DeepSeek-AI. Deepseek-v3 technical report, 2024.

[39] Kimi Authors. Kimi k1.5: Scaling reinforcement learning with LLMs, 2025.

[40] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[41] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[42] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[43] Leandro von Werra, Lewis Schmid, Thomas Wolf, and Lewis Tunstall. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`, 2020-2024.

[44] Lukas Biewald. Experiment tracking with weights and biases. `https://wandb.ai`, 2020. Software available from wandb.com.

A Model Training Details

A.1 Training Details for *Mathesis-Autoformalizer*

The training of our *Mathesis-Autoformalizer* model, which employs Group Relative Policy Optimization (GRPO), involves several key hyperparameters and implementation choices as briefly mentioned in Section 3.1.1. The policy π_θ is initialized from Kimina-Autoformalizer [39]. We employ Parameter-Efficient Fine-Tuning (PEFT) via Low-Rank Adaptation (LoRA) [40], configured with a rank $r = 16$ and $\alpha = 32$. LoRA is applied to the attention projection layers of the base model. The optimization is performed using the AdamW optimizer [41] with a learning rate of 1×10^{-6} and gradient checkpointing to manage memory usage.

For the GRPO algorithm itself, we sample $G = 14$ candidate formal statements per input natural language problem x. The Kullback-Leibler (KL) divergence coefficient β, which regularizes the policy updates against the reference SFT policy, is set to 0.04. The policy model π_θ is updated once per sampling/exploration phase (i.e., $\mu = 1$, meaning updates occur after each group of G generations for a given input x is processed and rewarded). To enhance efficiency, reward computations (syntactic verification via Lean and semantic assessment) are parallelized using Python's `asyncio` library. The overall training pipeline is managed using the Hugging Face `transformers` [42] and `trl` [43] libraries. Experiment progress and results are logged using Weights & Biases [44].

A.2 Training Details for *Mathesis-Prover*

For the SFT phase, we conduct hyperparameter optimization exploring learning rates of 5×10^{-5} with training durations of 2 epochs. All training runs are executed on Huawei Atlas 800T A2 servers, with each iteration's verified proofs augmenting the training set for subsequent refinement cycles. This methodology ensures continuous improvement while maintaining rigorous verification standards throughout the training process.

B Experiments Details and Case Study

B.1 Sucess Rate @ k

For both the Lean Syntax Check and the LLM Semantic Check, the success rate@k is reported for the check. Specifically, the success rate@k refers to the rate in which a given natural language statement is formalized k times, and at least one of the formulations passes the lean syntax check. The rate is formally defined as:

$$\begin{aligned} \text{success rate@}k &= \mathbb{E}\Big[\mathbb{I}\Big(\bigvee_{i=1}^{k} s_i = 1\Big)\Big] \\ &= \mathbb{E}\Big[1 - \Big(1 - \frac{C}{k}\Big)^k\Big], \;\; C \sim \text{Binomial}(k, p) \end{aligned} \tag{4}$$

where $s_i \in \{0, 1\}$ is be an indicator variable for whether the i-th formalization passes the check, c is the number of correct formalizations among k, and $p = \frac{c}{k}$ is the probability that a single formalization passes the check.

B.2 Sample Lean4 Proof Code With "Apply?" That Can Pass Lean Verifier

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

```

```
/-- Prove that the number $\sum^n_{k=0}\binom{2n+1}{2k+1}2^{3k}$ is not
    divisible by $5$ for any integer $n\ge0.$-/
theorem imo_1974_p3 (n : ℕ) :
    ¬5 ∣ ∑ k in Finset.range (n + 1), Nat.choose (2 * n + 1) (2 * k +
    1) * 2 ^ (3 * k) := by

  intro h
  have h  := h
  rw [Nat.dvd_iff_mod_eq_zero] at h
  have h  : (∑ k in Finset.range (n + 1), Nat.choose (2 * n + 1) (2 * k
    + 1) * 2 ^ (3 * k)) % 5 ≠ 0 := by
    -- We will show that the sum modulo 5 is never 0 by examining the
    terms and their contributions.
    have h  : (∑ k in Finset.range (n + 1), Nat.choose (2 * n + 1) (2 *
    k + 1) * 2 ^ (3 * k)) % 5 = 2 := by
      -- This is a placeholder for the actual proof, which would involve
    detailed combinatorial arguments.
      rw [← Cardinal.natCast_inj]
      apply?
    omega
  omega
```

B.3 Sample Lean4 Proof Code With "True" That Can Pass Lean Verifier

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/-Let $f(x)=x - ae^{x}(a\\in R)$, $x\\in R$. It is known that the
    function $y = f(x)$ has two zeros $x_1$, $x_2$, with $x_1 < x_2$.
    Prove that $\\frac{x_2}{x_1}$ increases as $a$ decreases.-/
theorem question (f : ℝ → ℝ → ℝ) (hf : f = fun a x => x - a *
    Real.exp x)
  (x₁ x₂ : ℝ → ℝ) (hx₁ : ∀ a, f a (x₁ a) = 0) (hx₂ : ∀ a, f a (x₂ a) =
    0)
  (h₁ : ∀ a, x₁ a < x₂ a) (h₂ : ∀ a, ∀ b, a < b → x₂ a / x₁ a < x₂ b / x₁
     b) :
  True := by
```

B.4 Informal and Formal Reasoning Case Study

Illustrative Example

Question: Real numbers a and b satisfy $a + b \geq 3$. Prove that $2a^2 + 2b^2 > a + b$.

Informal Reasoning:
Since $a + b \geq 3$, by the Cauchy-Schwarz Inequality:

$$(a^2 + b^2)(1 + 1) \geq (a + b)^2$$

That is:

$$2(a^2 + b^2) \geq (a + b)^2$$

Thus:

$$2a^2 + 2b^2 \geq (a + b)^2$$

Given $a + b \geq 3$, it follows that:

$$(a + b)^2 \geq 9$$

Therefore:

$$2a^2 + 2b^2 \geq 9$$

Meanwhile, since $a + b \geq 3$:

$$2a^2 + 2b^2 \geq 9 > 3 \geq a + b$$

Hence:

$$2a^2 + 2b^2 > a + b$$

Formal Reasoning:

```
theorem question_example (a b : ℝ) (h : a + b ≥ 3) : 2*a^2 + 2*b^2 > a + b := by
  nlinarith [sq_nonneg (a - b), sq_nonneg (a + b - 3),
             sq_nonneg (a + b - 1), sq_nonneg (a - 1),
             sq_nonneg (b - 1)]
```

B.5 Prompt Templates

Prompt for Autoformalization (used by Herald and Mathesis-Autoformalizer)

[Question]:
{statement}

You are an expert in formal mathematics. Your task is to convert the above [question] to lean 4 theorems by completing the following lean 4 code:

lean4
import Mathlib
import Aesop
set-option maxHeartbeats 0
set-option pp.numericTypes true
set-option pp.coercions true
set-option pp.letVarTypes true
set-option pp.structureInstanceTypes true
set-option pp.instanceTypes true
set-option pp.mvars.withType true
set-option pp.coercions true
set-option pp.funBinderTypes true
set-option pp.piBinderTypes true
open BigOperators Real Nat Topology Rat

{informal-comment}

Prompt for Autoformalization (used by all baseline models excepts Herald)

You are an expert in formal mathematics. Your task is to translate the given natural language mathematical statement into a formal Lean 4 theorem.

[Natural language statement]:
{statement}

Please convert this statement into a precise formal Lean 4 theorem. Follow these guidelines:

1. Start with "theorem" followed by a unique name or the provided ID if available
2. Define the types of all variables (e.g., a : $\mathbb{R}$ for real numbers)
3. Use appropriate mathematical symbols and notation
4. End with ":= by sorry" to indicate the proof will be completed later
5. Your formalization must exactly capture the mathematical meaning of the statement

Formal Lean 4 theorem:

Prompt for Mathesis-Prover

Complete the following Lean 4 code:

```lean4
{formal statement}
```

Prompt for LLM Semantic Check

You will receive a natural language math problem statement, along with its formal statement in LEAN 4 and, in some cases, a description of mathematical terms. Please evaluate whether the formal LEAN statement appropriately translates the natural language statement based on the following criteria. They are considered different if any of the criteria are not satisfied.

1. Key Elements: The fundamental mathematical components, including variables, constants, operations, domain, and codomain are correctly represented in LEAN code.
2. Mathematical Accuracy: The mathematical relationships and expressions should be interpreted consistently during translation.
3. Structural Fidelity: The translation aligns closely with the original problem, maintaining its structure and purpose.
4. Comprehensiveness: All conditions, constraints, and objectives stated in the natural language statement are mathematically included in the LEAN translation.

When doing evaluation, break down each problem statement into components, match the components, and evaluate their equivalence. Think step-by-step and explain all of your reasonings. Your answer should be in the following format:

Thought: [Your Answer]
Judgement: [Your Answer, one of Appropriate, Inappropriate]

Prompt for LeanScorer (Subtask Decomposition)

Help me list the conditions and conclusions in this problem (using specific mathematical formulas), without solving it:

Here is an example:
[Problem]: The sequence $\{a_n\}$ satisfies $a_1 = 1$, $a_2 = 2$, $a_{n+2} = 2a_{n+1} - a_n + 2$. Let $b_n = a_{n+1} - a_n$. Prove that $\{b_n\}$ is an arithmetic sequence.

[Conditions and Conclusions]:
Conditions:
1. $a_1 = 1$
2. $a_2 = 2$
3. $\forall n \geq 1, a_{n+2} = 2a_{n+1} - a_n + 2$
4. $\forall n \geq 1, b_n = a_{n+1} - a_n$

Conclusion:
- $\{b_n\}$ is an arithmetic sequence, i.e., $\exists d \in \mathbb{R}, \forall n \geq 1, b_{n+1} - b_n = d$.

Now, please help me extract the conditions and conclusions for this problem in the same way (using specific mathematical formulas), without solving it:
[Problem]: {informal statement}

[Conditions and Conclusions]:

Prompt for LeanScorer (LLM-based Evaluation)

Here is a math question and a lean 4 statement. Compare the conditions and conclusions in this code with the mathematical ones, matching them one by one to see if the formal statement is an appropriate translation of the mathematical condition by assigning one of three tags (Perfectly match; Minor inconsistency; Major inconsistency). Then, audit for missing/implicit conditions. Judge with extremely strict standards—any minor inconsistency will be considered a mismatch. Special attention to triangle angle-side correspondence. If the question explicitly mentions "opposite angles/sides", this correspondence must be clearly stated and correct.
Stop immediately after evaluating all pairs. Do **not** summarize or analyze further.

Output Format:
{one-shot example}

Question:
{informal statement}

Mathematical conditions and conclusions:
{math conditions}

Lean 4 formal statement:
{formal statement}

Output:

One-shot Example for LeanScorer (LLM-based Evaluation)

Let's compare the mathematical conditions and conclusions with the Lean 4 formal statement one by one:

1. **q is a natural number greater than 1**:
- Math: $q \in \mathbb{N}, q > 1$.
- Lean: '(hq : 1 < q)'.
- Match: \box{Perfectly match}.

2. **n is a natural number greater than 1**:
- Math: $n \in \mathbb{N}, n > 1$.
- Lean: '(hn : 1 < n)'.
- Match: \box{Perfectly match}.

3. **Set $M = \{0, 1, 2, \cdots, q-1\}$**:
- Math: M is explicitly defined as this set.
- Lean: '(M : Finset $\mathbb{N}$:= Finset.range q)'.
- Detailed interpretation: 'Finset.range q' is '0, 1, ..., q - 1'.
- Match: \box{Perfectly match}.

4. **Set A definition**:
- Math: $A = \{x | x = \sum_{i=1}^{n} x_i q^{i-1}, x_i \ in M\}$.
- Lean: 'A : Set $\mathbb{N}$:= {x | $\exists$ (x_vec : $\mathbb{N} \to \mathbb{N}$), ($\forall$ i, x_vec i $\in$ M) $\wedge$ x = $\sum$ i in Finset.range n, x_vec(i + 1) * q ^i}'.
- Detailed interpretation: In Lean, 'x_vec' is indexed from '1' to 'n' (since 'i + 1' ranges from '1' to 'n'), but the math defines x_i for $i = 1, 2, \cdots, n$. This is actually consistent, but the Lean representation is slightly more general (allowing 'x_vec' to be a function on all naturals, but only using 'x_vec (i + 1)' for 'i' in 'Finset.range n'). The Lean definition is technically correct but slightly more abstract than the math. However, it captures the same idea.
- Match: \box{Minor inconsistency}.

5. **$s, t \in A$ with specific expansions**:
- Math: $s = \sum_{i=1}^{n} a_i q^{i-1}$, $t = \sum_{i=1}^{n} b_i q^{i-1}$, with $a_i, b_i \in M$.
- Lean: 's = $\sum$ i in Finset.range n, a (i + 1) * q ^i', 't = $\sum$ i in Finset.range n, b (i + 1) * q ^i', with '$\forall$ i, a i $\in$ M' and '$\forall$ i, b i $\in$ M'.
- Detailed interpretation: The Lean version uses 'a (i + 1)' and 'b (i + 1)' to match the indexing in the sum, which is equivalent to the math but slightly indirect. The math directly uses a_i for $i = 1, \ldots, n$, while Lean uses 'a i' for all 'i' but only evaluates at 'i + 1'. The Lean version is correct but not a literal translation.
- Match: \box{Minor inconsistency}.

6. **$a_n < b_n$**:
- Math: $a_n < b_n$.
- Lean: '(hab : a n < b n)'.
- Match: \box{Perfectly match}.

7. **Conclusion $s < t$**:
- Math: $s < t$.
- Lean: 's <= t'.
- Match: \box{Major inconsistency}.

Check for missing conditions / implicit conditions:
- No missing conditions / implicit conditions
- Match: \box{Perfectly match}.