CSIAM Trans. Appl. Math.
doi:

Vol. **x**, No. x, pp. 1-22
xx 20xx

arXiv:2505.20613v3 [cs.CL] 24 Nov 2025

REAL-Prover: Retrieval Augmented Lean Prover for Mathematical Reasoning

Ziju Shen[1,†], Naohao Huang[2,†], Fanyi Yang[3,†], Yutong Wang[1,†], Guoxiong Gao[1,†], Tianyi Xu[1], Jiedong Jiang[1], Wanyi He[1], Pu Yang[1], Mengzhou Sun[1], Haocheng Ju[1], Peihao Wu[3], Bryan Dai[3] and Bin Dong[4,5,6,*]

[1] *Peking University*
[2] *Renmin University of China*
[3] *Ubiquant*
[4] *Beijing International Center for Mathematical Research and the New Cornerstone Science Laboratory, Peking University*
[5] *Center for Machine Learning Research, Peking University*
[6] *Center for Intelligent Computing, Great Bay Institute for Advanced Study, Great Bay University*
[†] *Equal contribution.*
Code: https://github.com/frenzymath/REAL-Prover

Abstract. Nowadays, formal theorem provers have made monumental progress on high-school and competition level mathematics, but few of them generalize to more advanced mathematics. In this paper, we present REAL-Prover, a new open-source stepwise theorem-prover for Lean 4 to push this boundary. This prover, based on our fine-tuned large language model (REAL-Prover-v1) and integrated with a retrieval system (Leansearch-PS), notably boosts performance on solving college-level mathematics problems. To train REAL-Prover-v1, we developed HERALD-AF, a data extraction pipeline that converts natural language math problems into formal statements, and a new open-source Lean 4 interactive environment (Jixia-interactive) to facilitate synthesis data collection. In our experiments, our prover using only supervised fine-tuning achieves competitive results with a 23.7% success rate (Pass@64) on the ProofNet dataset—comparable to state-of-the-art (SOTA) models. To further evaluate our approach, we introduce FATE-M, a new benchmark focused on algebraic problems, where our prover achieves a SOTA success rate of 56.7% (Pass@64).

AMS subject classifications: 68T15, 03B35, 68T20

Key words: Neural Theorem Proving, Large Language Models, Formal Reasoning, Proof Search

*Corresponding author. *Email addresses:* dongbin@math.pku.edu.cn

 ©20xx Global-Science Press

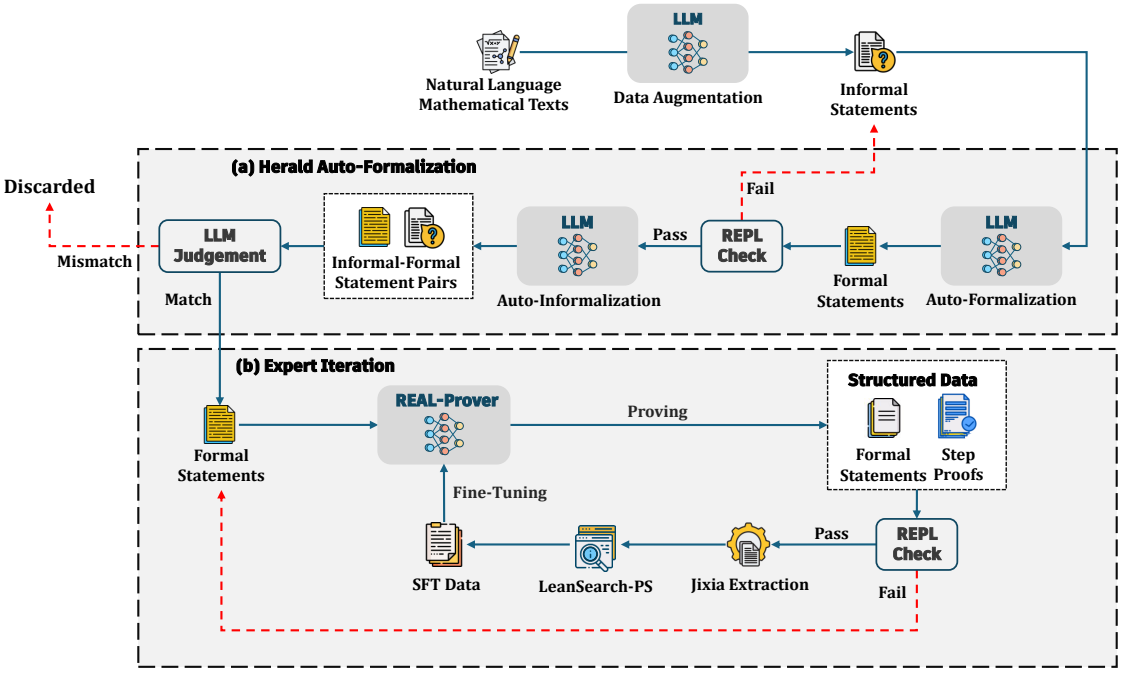


Figure 1: **Overview of the REAL-Prover model training pipeline.** The figure illustrates two components: (a) the HERALD-AF pipeline, which translates informal mathematical statements into formal ones; (b) the expert iteration pipeline, which iteratively refines the prover model.

1 Introduction

The formalization of mathematics has long been a great challenge at the intersection of programming and mathematical logic. Over the past decades, theorem provers have made steady progress. In particular, proof assistants have been used to formalize complex theorems such as the Four-Color Theorem [1] and the Feit-Thompson Odd-Order Theorem [2]. These achievements underscore the potential of formal methods to resolve research-level proofs.

Modern Interactive Theorem Provers (ITPs) like Lean [3] and Coq [4] have significantly improved usability and automation, lowering the barrier for adoption. Lean, for example, provides a user-friendly environment and a large community-driven library of mathematics (Mathlib) [5]. In early 2025, Lean's Mathlib had grown to encompass more than 210,000 formally proven theorems in various domains. Meanwhile, utilities like LeanSearch [6] also emerge to help users navigate easily in these numerous domains and facts, saving their efforts on formal proof drafting.

Recent advances in large language models (LLMs) have significantly advanced automated theorem proving, successfully generating formal proofs for high-school-level

math competitions [7–15]. Nonetheless, extending these approaches to undergraduate and graduate-level mathematics remains a significant challenge due to data sparsity, the intricacies of domain-specific reasoning, and the requirement for precise retrieval and application of extensive lemma libraries, where even minor inaccuracies can compromise the correctness of purely neural proof generation.

To address these challenges, we propose **REAL-Prover** (REtrieval-Augmented Lean Prover), a retrieval-augmented and stepwise framework for automated theorem proving in Lean. Each model invocation is conditioned on the current proof state and enhanced through the retrieval of semantically relevant theorems from Mathlib. The framework comprises three key components. **LeanSearch-PS** performs semantic premise selection through a two-stage training strategy that improves both retrieval precision and generalization. **HERALD-AF** extends the HERALD pipeline to systematically translate informal mathematical statements into verified Lean expressions, enabling scalable data generation. **Jixia-Interactive** provides a Lean 4 interactive environment that support efficient interaction between prover and compiler in stepwise proving. For evaluation, we construct **FATE-M** (Formal Algebra Theorem Evaluation–Medium), a benchmark designed for graduate-level reasoning in algebraic domains including groups, rings, and fields. REAL-Prover achieves a state-of-the-art accuracy of 56.7% on FATE-M and demonstrates competitive performance on miniF2F and ProofNet, confirming its effectiveness in complex formal reasoning.

1.1 Related works

Automated Theorem Proving Automated Theorem Proving in formal languages, particularly within the framework of ITPs, is a rapidly advancing research area in artificial intelligence. LLM-based provers can be broadly categorized by their proof generation strategies: whole-proof generation that aims to produce complete proof in a single pass, and proof-step generation that iteratively refines proofs through ITPs.

The whole-proof generation paradigm focuses on end-to-end synthesis of formal proof scripts from theorem statements. Early attempts like DeepSeek-Prover-V1 [16] demonstrated the feasibility of this approach in Lean 4. Subsequent improvements in DeepSeek-Prover-V1.5 [13] introduced hybrid techniques combining Reinforcement Learning from Proof Assistant Feedback (RLPAF) with Group Relative Policy Optimization [17], along with a 'truncate-and-resume' mechanism that allows error correction through partial stepwise verification. Goedel-Prover [18] addressed data scarcity through auto-formalization, first converting natural language mathematics into formal statements before generating proofs through iterative training. While these approaches benefit from large-scale supervised fine-tuning and reduced ITP interaction, they struggle with error accumulation in long logical chains and implicit state inference.

This limitation has driven the development of interactive proof-step generation approaches that leverage ITPs feedback during the proving process. Systems like BFS-Prover [14] employ expert iteration with Best-First Search, using Lean compiler feedback

to guide tactic generation. HunyuanProver [19] enhances search efficiency through specialized critic models that evaluate intermediate proof states. The LeanDojo framework introduces ReProver [20], which adopts a distinct retrieval-based approach using Dense Passage Retriever [21] to directly predict relevant premises. LeanAgent [22] extends this paradigm through lifelong learning, maintaining a dynamic knowledge base to progressively master theorem proving across multiple repositories. These interactive methods typically combine expert iteration, enabling real-time adaptation to proof state changes while managing computational complexity through guided tree search.

Formal Statement Benchmark A fundamental aspect of developing and evaluating theorem provers is the utilization of formal statement benchmarks. Prominent examples include miniF2F [23], which offers a curated set of Olympiad-level formalization problems, and ProofNet [24], which targets undergraduate-level mathematical reasoning. These benchmarks serve as standardized tools to assess the capabilities of theorem provers in generating correct and complete proofs. As the proficiency of provers advances, there is an increasing need for more challenging benchmarks, such as graduate-level or research-oriented problems, to rigorously evaluate their reasoning depth, adaptability, and ability to handle complex or previously unseen formalizations.

1.2 Our Contributions

Our main contributions are summarized as follows:

- **Retrieval-Augmented Proof System.** We present a unified proof system that integrates data collection, training, and inference. The system includes three main components. HERALD-AF provides a pipeline for translating informal mathematical statements into formal Lean expressions. REAL-Prover implements a stepwise theorem-proving framework with LeanSearch-PS for semantic premise selection and employs an expert-iteration training paradigm. Jixia-Interactive facilitates efficient and stable interactions with the Lean 4 prover during both training and inference. The integration of these components significantly enhances proof performance on graduate-level formal mathematics.
- **State-Tactic Pair Dataset.** We release a comprehensive dataset consisting of 55k state–tactic pairs generated through our expert iteration process. Each data instance encapsulates a formal proof state, its surrounding context, and the corresponding tactic sequence, providing high-quality supervision for both proof search optimization and policy learning.
- **Specialized Benchmark for Abstract Algebra.** We present FATE-M (Formal Algebra Theorem Evaluation–Medium), a specialized benchmark comprising paired informal-formal problems in graduate-level abstract algebra. FATE-M complements existing benchmarks such as miniF2F [23] and ProofNet [24], establishing an effective framework for rigorous formal reasoning assessment.

2 Method

To construct a high-performance automated prover, we propose REtrieval-Augmented Lean Prover (REAL-Prover), a stepwise proof system for Lean 4. To supply REAL-Prover with a high-quality training corpus, we first introduce Hierarchy and Retrieval-based Translated Lean Dataset Auto-formalization (HERALD-AF) in Section 2.1, a pipeline that translates informal mathematical problems into formal Lean 4 statements. HERALD-AF converts informal mathematical problems into formal Lean 4 statements, yielding a large corpus of candidate theorems. Since these statements still lack certified proofs, in Section 2.2, we build a complementary pipeline—again driven by REAL-Prover—that synthesizes and automatically verifies the required proofs, completing the training loop.

2.1 HERALD-AF

Our formal statement generation framework builds upon the Hierarchy and Retrieval-based Translated Lean Dataset (HERALD) [25], which establishes a protocol translating informal statements into formal statements. Figure 1(a) schematizes HERALD-AF pipeline.

Before applying HERALD-AF, we perform a preprocessing step to extract informal statements from a natural language mathematics corpus. This begins with the use of regular expressions to identify theorem blocks and exercises within the text. Subsequently, an LLM is employed to generate and refine the extracted informal statements.

Once the informal statements are obtained, we apply HERALD-AF to translate them into formal statements. This translation pipeline comprises three key stages: Auto-Formalization, Auto-Informalization, and LLM Judgement. In the Auto-Formalization stage, an open-source model Herald-translator [25] is used to generate multiple candidate formal statements from each informal input. During Auto-Informalization, these formal candidates are translated back into natural language using DeepSeek-V3 [26]. Finally, in the LLM Judgement stage, a general LLM evaluates whether each back-translated statement faithfully reflects the original input. Only those formal statements that pass this consistency check are retained.

2.2 REAL-Prover

From the HERALD-AF pipeline, we obtained a collection of formal statements without corresponding proofs. To generate complete proofs and achieve the second goal, we introduce the REAL-Prover system, illustrated in Figure 2. We developed a new Lean 4 interactive environment, Jixia-interactive, which receives Lean tactics and updates the proof state. For each formal statement, Jixia-interactive initializes the proof state. REAL-Prover constructs a search tree to explore possible proof paths. At each node, a LLM generates multiple candidate tactics. Jixia-interactive then applies these tactics, discards invalid ones, and advances to new proof states. Similar to BFS-Prover [14], each node is

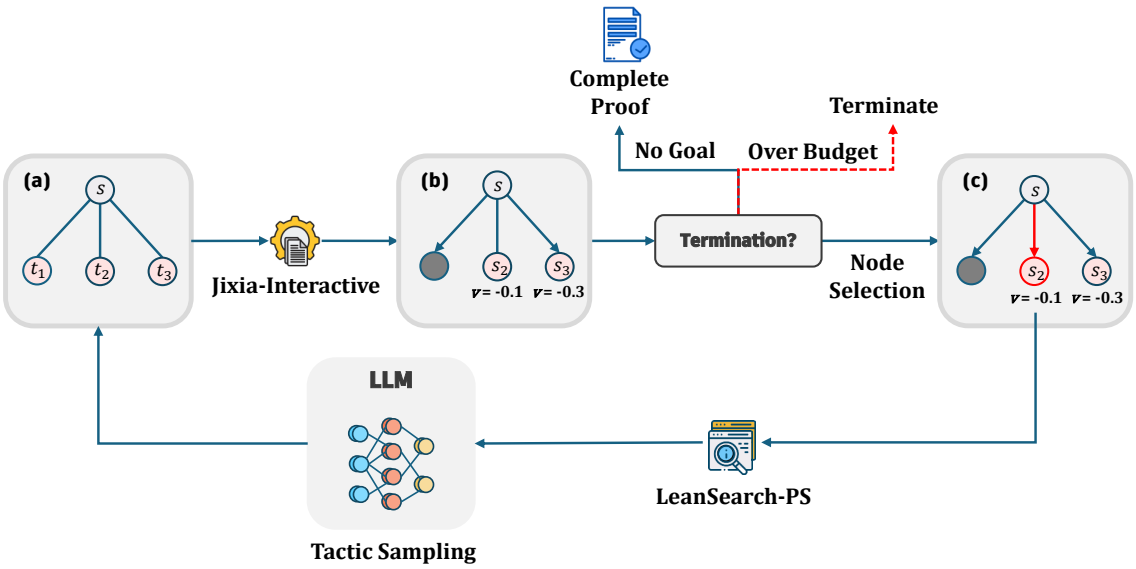


Figure 2: **Overview of REAL-Prover.** This figure illustrates three main processes: From (a) to (b), the prover passes several tactics to the Lean 4 interactive environment (Jixia-interactive) and receives corresponding scored state nodes. From (b) to (c), the prover selects a state node based on its score and checks whether the proof is complete. From (c) to (a), the prover using LLM with retrieval information from LeanSearch-PS generate new tactics.

scored using the scoring function: $\frac{\sum_{t=0}^{L-1} \log p(a_t|s_t)}{L^\alpha}$, where the s_t represents the proof state and a_t is the tactic applied at step t. $p(a_t|s_t)$ is the model's predicted probability of applying tactic a_t at state s_t. L is the total length of the path from the root to the current state s_L and α is a hyper-parameter in $[0,1]$. We adopt a best-first search strategy, selecting the node with the highest score to expand further by generating new tactics. This iterative process continues until either the proof is completed or a predefined search budget is exhausted. Once completed, the full proof can be reconstructed from the search tree. To enhance our prover's performace, we add retrieval information from Mathlib using LeanSearch premise selection (LeanSearch-PS), which extends the work of LeanSearch [6]. We will discuss the details of LeanSearch-PS.

2.2.1 LeanSearch-PS

In our work, premise selection addresses the challenge of identifying the most relevant theorems based on the current Lean proof state. To enhance this process, we developed LeanSearch-PS, a retrieval system built around a high-performance embedding model. This model projects both Lean proof states and Mathlib theorems into a shared semantic vector space. During proof generation, the current proof state is embedded, and a nearest-

neighbor search is performed to retrieve the top-k most relevant theorems. These selected premises are then supplied as context to our language model, helping it produce more effective and accurate proof steps.

To train the embedding model, we construct a dataset of (s,t_{pos}) pairs extracted from Mathlib using the Jixia tool [27]. Here, s denotes a Lean proof state, and t_{pos} denotes the theorem successfully applied at that step. The training process adopts a two-stage framework inspired by transfer learning strategies commonly used in text retrieval [28]. In both stages, we employ the InfoNCE loss [29] defined as

$$L(q,k_i) = -\log \frac{\exp(q \cdot k_0/\tau)}{\sum_{i=0}^{n} \exp(q \cdot k_i/\tau)}, \tag{2.1}$$

where k_0 is the embedding of positive example, $k_1,\ldots,k_n$ are the embedding of negative examples, and τ is a temperature hyperparameter. In the subsequent description, we will omit explicit mention of the embedding for brevity.

The training process consists of the following two stages.

- **Initial Training:** We treat the proof state s as the query q and its corresponding theorem t_{pos} as the positive example k_0. The in-batch negatives strategy is used, where other premises in the same batch serve as negative examples. This enables contrastive learning of premise relevance across proof states.
- **Hard Negative Enhanced Training:** To refine fine-grained discrimination, we leverage the initially trained model to mine hard negatives—challenging but incorrect premises for each state. This yields triplets $(s,t_{\text{pos}},t_{\text{hard_neg}})$, where $t_{\text{hard_neg}}$ denotes a mined hard negative premise. During this stage, we use s as q, t_{pos} as k_0, and $t_{\text{hard_neg}}$ as k_i $(i=1,\ldots,n)$. The resulting model after this phase constitutes our final embedding model.

2.2.2 Expert Iteration

Another key component of REAL-Prover is the prover model within the tactic generator. To train a strong prover model, we combine expert iteration to collect synthetic data, as shown in Figure 1(b). We begin by fine-tuning a prover model using open-source datasets. This initial model is then employed within REAL-Prover to solve formal statements produced by HERALD-AF, generating additional high-quality data. The newly collected data is incorporated into the supervised fine-tuning dataset to train an improved model. This refined model is then used to tackle challenging formal statements that the previous model could not solve. We update our model continuously in this iteration process.

3 Experiment Setup

3.1 Implementation Detail

REAL-Prover Training We perform supervised fine-tuning on the base models Qwen2.5-Math-7B [30] using a learning rate of 5×10^{-5} with a cosine decay scheduler and a maxi-

mum context length of 8192 tokens. Our training data comprises the following sources, totaling 210,420 state-tactic pairs:

- Post-processed formal proofs collected via expert iteration. Each proof is parsed into state-tactic pairs using Jixia, and the corresponding prompts are enriched with natural language descriptions, the proof context, the current proof state, and retrieval information from LeanSearch-PS. The dataset comprises 15,071 algebra problems from undergraduate and graduate mathematics texts, and 19,633 NuminaMath problems [31], resulting in 25,818 and 30,589 state-tactic pairs, respectively, after post-processing.
- Post-processed, human-annotated datasets were constructed following the same procedure as expert-iteration proofs. Based on 196 annotated undergraduate-level algebra problems, the datasets contain 18,669 state-tactic pairs. Annotations were provided by undergraduate and graduate students majoring in mathematics, ensuring high quality.
- Formal proofs are extracted from Mathlib and parsed into state-tactic pairs using Jixia. Prompt inputs are further augmented with retrieval information from LeanSearch-PS, while natural language descriptions are omitted. The resulting dataset comprises 92,152 state-tactic pairs.
- The augmented Lean-Workbook dataset [32]. We enhance this dataset by adding retrieval information from LeanSearch-PS to each example, while keeping the original structure intact. The resulting dataset comprises 43,192 state-tactic pairs.

LeanSearch-PS Training We develop two dense retrievers from E5-mistral-7b-instruct [33] with Tevatron [34] training framework. For both initial training and hard negative enhanced training, we use LoRA to accelerate training, and set the learning rate as 2×10^{-5}, the batch size as 128, the number of epoch as 1, and other hyperparameters as their default values. Queries are formal statements with a maximum length of 128 and passages are formal theorems with a maximum length of 256. For the hard negative examples, we first embed all statements and theorems with the initial trained embedding model, and then randomly select one passage from the top 30 to top 100 most similar ones for each query as its hard negative premise.

REAL-Prover Inference In our experiments, considering the overall computational budget, we set the number of passes and samples per step to 64. We set our LLM's temperature to 1.5. Additionally, we set the hyperparameter α of the score function to 0.5, consistent with the configuration used in BFS-Prover [14]

3.2 Benchmark

As our prover highlights its capabilities on college-level mathematics, we created our own benchmark, FATE-M, which focuses specifically on algebraic formal statements to

test model's performance on rich algebra structures. In addition, we also follow prior work [11, 18, 19, 35] to evaluate our prover's general capabilities using the ProofNet and MiniF2F benchmarks.

ProofNet ProofNet consists of 185 validation problems and 186 test problems, drawn from widely used undergraduate pure mathematics textbooks. The dataset spans a variety of topics, including real and complex analysis, linear algebra, abstract algebra, and topology.

MiniF2F MiniF2F includes 244 validation problems and 244 test problems, collected from a diverse set of mathematical sources such as the AIME, AMC, and IMO competitions. This benchmark focuses on Olympiad-level problems covering core areas of elementary mathematics, including algebra, number theory, and mathematical induction.

FATE-M (Formal Algebra Theorem Evaluation-Medium) FATE-M is a benchmark designed to evaluate theorem-proving capabilities in Lean4 for undergraduate-level abstract algebra. It consists of 141 problems formalized in Lean4, most accompanied by natural language comments in English or Chinese. Focusing on core topics in abstract algebra, FATE-M spans problems of simple to moderate difficulty, all of which rely on definitions and lemmas already formalized in Lean's Mathlib. To our knowledge, it is the first benchmark specifically targeting this domain, offering a specialized tool for assessing formal reasoning in algebra. Future expansions of the FATE series will include more advanced topics, such as commutative algebra, with increased difficulty.

The benchmark was created by extracting problems from 12 university-level abstract algebra textbooks (for a full list, see Section B), which were then formalized by students. Each formalized statement was rigorously verified by Mathlib contributors and PhD students in algebra to ensure it accurately represents the original mathematical meaning. Additionally, Lean-checked proofs are provided for all statements to guarantee their correctness.

We present two examples to illustrate the nature of the benchmark:

```
import Mathlib

example {G : Type*} [Group G] {A B : Subgroup G} :
    (∃ C : Subgroup G, C.carrier = (A.carrier ∪ B.carrier)) ↔ A ≤ B ∨ B ≤ A :=
     by
```

This is a medium-level problem in the benchmark. This problem translates to the following natural language statement: Let A, B, and C be subgroups of a group G. If $C = A \cup B$, then either $A \subseteq B$ or $B \subseteq A$.

```
import Mathlib

open Classical

/-Prove that a group of order $p^{2}, p$ a prime, has a normal subgroup of order
    $p$.-/
example {G : Type*} {p : ℕ} [Group G] [Fintype G] (pp : p.Prime) (ord :
    Fintype.card G = p ^ 2) : ∃ P : Subgroup G, (P.Normal) ∧ (Fintype.card P =
    p) :=
```

This problem represents one of the more challenging exercises in the benchmark. The natural language translation precisely matches the Lean code comment. An equivalent formulation would be: Let p be a prime number. Then every group of order p^2 contains a normal subgroup of order p.

4 Quantitative Result

Our main results are on ProofNet and FATE-M, both of which consist of college-level mathematics problems that require the use of numerous theorems. To evaluate the generalization ability of our system, we also test it on MiniF2F, a benchmark in a different domain than ProofNet and FATE-M. The results are shown below.

4.1 ProofNet Result

In the ProofNet benchmark, we compare REAL-Prover with several leading provers, Geodel-Prover [18], DeepSeek-Prover-V1.5 [13]. For whole-proof system, the sampling budget in table means the number of generation. And for tree-search systems, the sampling budget is M×N, where M is the total number of passes and N is the number of every step generation. The experimental results demonstrate that REAL-Prover with retrieval archieved 23.7% success rate only using supervied fine-tune. Surpassing DeepSeek-Prover-V1.5-RL under a 3200-sampling budget and DeepSeek-Prover-V1.5-RL+RMaxTS under 1×3200-sampling budget, which are trained with reinforcement learning.

4.2 FATE-M Result

The performance on the FATE-M benchmark is presented in Table 2. Given the availability of open-source models and pipelines, we evaluate only Goedel-Prover and DeepSeek-Prover by using their official repositories. In this experiment, our prover achieves a state-of-the-art result with a Pass@64 of 56.7%. This demonstrates that our prover exhibits strong capability in solving college-level algebraic problems.

Table 1: Comparison with state-of-the-art models (all 7B parameters) on ProofNet.

Prove System	Sampling Budget	ProofNet Test
Whole-proof systems		
Goedel Prover [18]	32	15.6 %
DeepSeek-Prover-V1.5-SFT [11]	128	15.9 %
DeepSeek-Prover-V1.5-RL [11]	128	18.2 %
Tree search systems		
DeepSeek-Prover-V1.5-RL + RMaxTS [13]	1 × 3200	21.6 %
REAL-Prover (**ours**)	64 × 64	**23.7** %

Table 2: Comparison with other system (all 7B parameters) on FATE-M.

Prove System	Sampling Budget	FATE-M Test
Whole-proof systems		
Goedel Prover [18]	128	18.7 %
DeepSeek-Prover-V1.5-RL [11]	128	31.2 %
Tree search systems		
DeepSeek-Prover-V1.5-RL + RMaxTS [13]	64 × 64	41.8 %
REAL-Prover (**ours**)	64 × 64	**56.7** %

4.3 MiniF2F Result

The MiniF2F benchmark has been the focus of more extensive research, as summarized in Table 3. For tree search systems, serveral provers have K×W×N sampling budget, where K represents the number of passes, W the expansion width, and N the maximum number of proof state expansions per pass. As shown in the table, our model performs relatively poorly compared to state-of-the-art models. We consider two main factors that may contribute to this performance gap. First, MiniF2F primarily features high school-level olympiad problems, which often do not heavily rely on theorems from Mathlib. Consequently, the benefit of retrieval is diminished compared to its effectiveness on college-level problems. Second, reinforcement learning plays a significant role in the training of current state-of-the-art provers. In contrast, our model is trained solely with supervised fine-tuning, making it challenging to compete without the additional boost provided by retrieval mechanisms.

Table 3: Comparison with other system on MiniF2F.

Prove System	Sampling Budget	MiniF2F Test
Whole-proof systems		
DeepSeek-Prover-V1.5-RL [11]	102400	60.2%
Leanabell-Prover-GD-RL [36]	128	61.1%
Goedel-Prover [18]	25600	64.7%
STP [37]	25600	67.6%
Tree search systems		
LLMStep [38]	1×32×100	27.9 %
GPT-f [7]	64×8×512	36.6 %
Hypertree Proof Search [39]	64 × 5000	41.0 %
DeepSeek-Prover-V1.5-RL + RMaxTS [13]	32 × 6400	63.5 %
InternLM2.5-StepProver-BF [12]	256 × 32 × 600	65.9 %
HunyuanProver v16 + BFS + DC [19]	600 × 8 × 400	68.4 %
BFS-Prover [14]	2048 × 2 × 600	70.8 %
REAL-Prover (**ours**)	64 × 64	54.1 %

4.4 Ablation Study

In this section, we will analysis the effective of LeanSearch-PS. To compare with REAL-Prover-v1, we trained a prover model, REAL-Prover-v1-NoRet, which used the same dataset but without retrieval information. And in REAL-Prover, we do not use LeanSearch-PS for REAL-Prover-v1-NoRet model. We evaluate both models on the ProofNet and FATE-M benchmarks with results presented in Table 4. The results show that REAL-Prover-v1 achieves a performance improvement over REAL-Prover-v1-NoRet on both benchmarks. This demonstrates that the retrieval system enhances the prover's performance on college-level mathematics problems.

Table 4: Ablation study on ProofNet Test and FATE-M Test. REAL-Prover-v1-NoRet w/o LeanSearch refers to use the REAL-Prover-v1-NoRet model without LeanSearch-PS. REAL-Prover-v1 w/ LeanSearch refers to use the REAL-Prover-v1 model with LeanSearch-PS enabled.

Prove System	ProofNet Test	FATE-M Test
REAL-Prover-v1-NoRet w/o LeanSearch	22.6%	44.7%
REAL-Prover-v1 w/ LeanSearch	23.7%	56.7%

We present a comparison between proofs with and without LeanSearch-PS. In Figure 3, the proof assisted by LeanSearch-PS successfully retrieves critical lemmas and completes verification, whereas the proof without LeanSearch-PS fails to verify. For problems that can be resolved both with LeanSearch-PS and without LeanSearch-PS, the one with LeanSearch-PS often leads to more accurate and human-readable proof, we provide 2 examples in Section E to illustrate this observation.

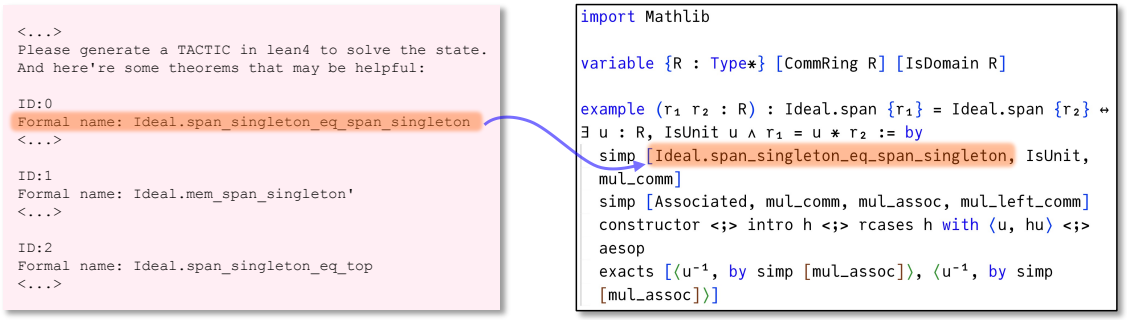


Figure 3: Proof using LeanSearch-PS and retrieval results. The left shows part of our input prompt; the right displays the proof generated by REAL-Prover.

5 Conclusion and Future Work

In this work, we propose REAL-Prover, a stepwise proof system designed to tackle challenging mathematical problems. To support its development, we introduce the HERALD-AF pipeline, which translates informal mathematical statements into formal Lean 4 representations, enabling the construction of a large-scale formal dataset. We also develop a new interactive Lean 4 environment, Jixia-Interactive, which facilitates both the training of our prover model, REAL-Prover-v1, and efficient formal proof generation. Furthermore, we present LeanSearch-PS, a theorem retrieval system that improves theorem selection and boosts overall proof success on college-level mathematical benchmarks. To comprehensively evaluate the prover's effectiveness, we propose the FATE-M benchmark, on which REAL-Prover achieves a state-of-the-art success rate of 56.7%.

Despite these promising results, several limitations remain. The current model is trained purely with supervised objectives, and we have not yet incorporated reinforcement learning (RL) due to its computational demands. Nevertheless, prior studies suggest that RL can unlock latent reasoning capabilities in large language models, indicating that RL-based fine-tuning (e.g., GRPO [17]) constitutes a promising avenue for enhancing performance. In addition, REAL-Prover emits only the final Lean tactics, whereas several leading provers first generate a natural-language chain of thought (CoT) and then translate it into formal steps, enabling powerful test-time reasoning amplification. Introducing an

informal-reasoning layer would let the model draft, filter, and organize candidate arguments before committing to formal tactics, potentially improving both premise selection and high-level proof planning.

Future work will focus on these two complementary directions: incorporating RL-based fine-tuning to deepen reasoning capabilities and generalization, and integrating CoT-driven informal reasoning to bridge natural and formal mathematical discourse. These extensions are expected to yield a more autonomous, interpretable, and robust formal reasoning system, advancing the broader objective of scalable, human-aligned mathematical intelligence.

Acknowledgements

This work is supported in part by National Key R&D Program of China grant 2024YFA1014000, the New Cornerstone Investigator Program, and Ubiquant.

References

[1] Georges Gonthier. The four colour theorem: Engineering of a formal proof. In Deepak Kapur, editor, *Computer Mathematics*, pages 333–333, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[2] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 163–179, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[3] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.

[4] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[5] 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.

[6] Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. A semantic search engine for mathlib4. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8001–8013, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

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

[8] Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023.

[9] Xueliang Zhao, Wenda Li, and Lingpeng Kong. Subgoal-based demonstration learning for formal theorem proving. In *Forty-first International Conference on Machine Learning*, 2024.
[10] Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*, 2024.
[11] Huajian Xin, Daya Guo, Zhihong Shao, Z.Z. Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Advancing theorem proving in LLMs through large-scale synthetic data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024.
[12] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.
[13] Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2025.
[14] 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.
[15] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.
[16] Huajian Xin, Daya Guo, Zhihong Shao, Z.Z. Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Advancing theorem proving in LLMs through large-scale synthetic data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024.
[17] Qihao Zhu Zhihong Shao, Peiyi Wang et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024.
[18] Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.
[19] 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.
[20] Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.
[21] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.
[22] Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. Leanagent: Lifelong learning for formal theorem proving. In *The Thirteenth International Conference on Learning Representations*, 2025.
[23] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark

for formal olympiad-level mathematics. *CoRR*, abs/2109.00110, 2021.

[24] 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.

[25] Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025.

[26] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2025.

[27] frenzymath. jixia: A static analysis tool for lean 4. `https://github.com/frenzymath/jixia`, 2024.

[28] Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. SFR-Embedding-Mistral: Enhance text retrieval with transfer learning. Salesforce AI Research Blog, 2024.

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[30] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[31] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. *Hugging Face repository*, 2024.

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

[33] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[34] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Tevatron: An efficient and flexible toolkit for neural retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 3120–3124, New York, NY, USA, 2023. Association for Computing Machinery.

[35] Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. TheoremLlama: Transforming general-purpose LLMs into lean4 experts. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11953–11974, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[36] Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover: Posttraining scaling in formal reasoning. *arXiv preprint arXiv:2504.06122*, 2025.

[37] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

[38] Sean Welleck and Rahul Saha. Llmstep: Llm proofstep suggestions in lean. *arXiv preprint arXiv:2310.18457*, 2023.

[39] Guillaume Lample, Timothee Lacroix, Marie anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

A Prompt for REAL-Prover

In Lean, a formal proof is a fully constructed proof term that is type-checked and verified by the kernel. It represents a complete and correct derivation of a proposition.
The state after tactics refers to the intermediate proof state during tactic-based proof construction. It includes the list of remaining goals and the local context at that point. Relationship:

1. Tactics are procedural tools used to incrementally construct a formal proof.
2. Each tactic transforms the current proof state by solving or reducing goals.
3. The state after a tactic reflects the goals that still need to be proven after that tactic has been applied.
4. Once all goals are solved, Lean assembles the underlying proof terms generated by the tactics into a complete formal proof.
5. This final term is then type-checked by the kernel to ensure correctness.

In essence, the state after tactics shows where you are in the process of building a formal proof — it's a snapshot of what's left to do before the proof is complete.
Here is the FORMAL PROOF before the current state:

```
import Mathlib
open BigOperators
open Real Nat Topology

theorem parallelogram_midpoints_coincide_extracted {A B C D : ℂ} : (A + B + C
    + D) / 4 = (A + C) / 2 → (B + D) / 2 = (A + C) / 2 → B.re + D.re = 2 *
    ((A + C) / 2).re  := by
simp [add_comm, add_left_comm, add_assoc]
simp [div_eq_mul_inv, mul_add, mul_comm, mul_left_comm]
intro h₁ h₂
```

Here is the current STATE:

```
A : ℂ
B : ℂ
C : ℂ
D : ℂ
h₁ : (A + (B + (C + D))) * 4⁻¹ = (A + C) * 2⁻¹
h₂ : B + D = A + C
⊢ B.re + D.re = A.re + C.re
```

Please generate a tactic in lean4 to solve the state.
And here're some theorems that may be helpful:
ID:0
Formal name:`Complex.add_re`
Informal name: Real Part of Sum of Complex Numbers
Formal statement: `∀ (z w : ℂ), (z + w).re = z.re + w.re`
...
ID:5
Formal name: `add_add_add_comm`
Informal name: Commutativity and Associativity of Addition in Additive Commutative Semigroups
Formal statement: `∀ {G : Type u_3} [inst : AddCommSemigroup G] (a b c d : G), a + b + (c + d) = a + c + (b + d)`

Table 5: REAL-Prover training hyperparameters

Component	**Setting**
Full Fine Tuning	Learning rate : 5×10^{-5}; Scheducler: Cosine Decay
Backbone	Qwen2.5-Math-7B
Length	Prompt Max Length: 8192
Optimizations	`bf16, flash_attn`
Batch Sizes	`train_batch_size = 2`
Training Schedule	3 Epochs

B Source Textbooks for FATE-M

The FATE-M benchmark draws problems from the following abstract algebra textbooks:

1. Pinter, C. C. *A Book of Abstract Algebra* (2nd ed., 2010)
2. Fraleigh, J. B. *A First Course in Abstract Algebra* (8th ed., 2021)
3. Anderson, M. & Feil, T. *A First Course in Abstract Algebra: Rings, Groups, and Fields* (3rd ed., 2015)
4. Rotman, J. J. *A First Course in Abstract Algebra with Applications* (3rd ed., 2005)
5. Hodge, J. K., Schlicker, S., & Sundstrom, T. *Abstract Algebra: An Inquiry-Based Approach* (2014)
6. Herstein, I. N. *Abstract Algebra* (3rd ed., 1996)
7. Grillet, P. A. *Abstract Algebra* (2nd ed., 2007)
8. Choudhary, P. *Abstract Algebra* (2008)
9. Saracino, D. *Abstract Algebra: A First Course* (2nd ed., 2008)
10. Hungerford, T. W. *Abstract Algebra: An Introduction* (3rd ed., 2014)
11. Dummit, D. S. & Foote, R. M. *Abstract Algebra* (3rd ed., 2004)
12. Feng, K. & Zhang, P. *300 Problems in Abstract Algebra* (2009)

C Training Details

REAL-Prover training REAL-Prover training is performed on 8×A800 GPUs (80GB each) for approximately 18 hours, with hyperparameters detailed in Table 5.

LeanSearch-PS training LeanSearch-PS training is performed on 4×L40 GPUs (40GB each) for approximately 12 hours with hyperparameters detailed in Table 6.

Table 6: LeanSearch-PS training hyperparameters

Component	**Setting**
LoRA Fine Tuning	Learning rate : 2×10^{-5}
Backbone	E5-mistral-7b-instruct
Optimizations	`bf16, flash_attn`
Batch Sizes	`train_batch_size = 2`
Training Schedule	1 Epoch

D Broader Impacts

This work explores the use of large language models (LLMs) for automated formal theorem proving. Potential positive impacts include:

1. assisting mathematicians in constructing formal proofs more efficiently by automating tedious or routine steps;
2. facilitating the teaching of formal proof assistants such as Lean in undergraduate and graduate curricula;
3. potentially enabling LLMs to autonomously explore new conjectures or discover novel proofs, contributing to mathematical discovery.

However, several risks and limitations should be acknowledged. First, while LLMs may generate correct formal proofs, these proofs can often be opaque or unintuitive, reducing their explanatory value for human users. Second, excessive reliance on automated proof generation may hinder the development of deep mathematical understanding, particularly in educational settings. Third, the reinforcement learning and training process may be susceptible to reward hacking, especially given the complexity and non-determinism of the Lean environment—raising the risk of generating formally valid but semantically meaningless or misleading proofs that may be overlooked by human reviewers.

Careful evaluation, interpretability mechanisms, and human-in-the-loop verification are essential to ensure that such systems are used responsibly and contribute positively to both research and education.

E Case study

We provide 2 cases in FATE-M to illustrate that even if both REAL-Prover-v1 with LeanSearch-PS and REAL-Prover-v1-NoRet without Leansearch-PS produce the valid results, the one with premise selection often produce faster and readable proofs, as illustrated in Figure 4 and Figure 5.

```
example  {F : Type*} [Field F]
    [Fintype F] [IsAlgClosed F] :
    False := by
  set e : Fintype F :=
    Fintype.ofFinite F
  set L := (AlgebraicClosure F)
  have : Infinite L := by exact
    IsAlgClosed.instInfinite
  have := this
  have := Fintype.ofFinite F
  exact Fintype.false e
```

(a) Proof with LeanSearch-PS.

```
example  {F : Type*} [Field F]
    [Fintype F] [IsAlgClosed F] :
    False := by
  have : Fact (Nat.Prime 2) := ⟨by
    norm_num⟩
  exact not_finite F
```

(b) Proof without LeanSearch-PS.

Figure 4: Compare the proofs with and without LeanSearch-PS. The prover in (a) uses the existing instance 'IsAlgClosed.instInfinite' from Mathlib, resulting in a more readable proof.

```
example {G H : Type*} {p : ℕ}
    [Group G] [Group H] [Fact
    p.Prime] (gp : IsPGroup p G) (f
    : G →* H) (sf :
    Function.Surjective f) :
    IsPGroup p H := by
  exact IsPGroup.of_surjective gp f
    sf
```

(a) Proof with LeanSearch-PS.

```
example {G H : Type*} {p : ℕ} [Group
    G] [Group H] [Fact p.Prime] (gp :
    IsPGroup p G) (f : G →* H) (sf :
    Function.Surjective f) : IsPGroup
    p H := by
  rw [IsPGroup] at gp ⊢
  intro g
  obtain ⟨m, hm⟩ :=sf g
  rcases gp m with ⟨km, hkm⟩
  use km
  rw [← hm, ← map_pow, hkm, map_one
    f]
```

(b) Proof without LeanSearch-PS.

Figure 5: Compare the proofs with and without LeanSearch-PS. The prover in (a) uses the existing instance 'IsPGroup.of_surjective' from Mathlib, resulting in a more readable proof.