PROOFREFINER: COLLABORATIVE THEOREM PROVING WITH LARGE LANGUAGE MODELS: ENHANCING FORMAL PROOFS WITH PROOFREFINER

Anonymous authors
Paper under double-blind review

ABSTRACT

Theorem proving presents a significant challenge for large language models (LLMs) because formal proofs can be rigorously verified by proof assistants like Lean, leaving no room for errors. Existing LLM-based provers typically operate autonomously, but they often struggle with complex and novel theorems where human insights are crucial. We propose a new framework that positions LLMs as collaborative assistants in theorem proving to address this. This framework enables the seamless integration of LLM inference into the Lean environment, allowing developers to build various proof automation tools. These tools offer features such as suggesting proof steps, completing intermediate goals, and selecting relevant premises, thereby enhancing the theorem-proving process. Users can leverage our pretrained models or integrate their own, supporting local and cloud-based execution. Experimental results demonstrate that our approach is more effective in aiding humans and automating the theorem-proving process than existing rule-based systems. Additionally, we introduce a system called ProofRefiner, which refines questions and answers through dynamic dialogue adjustments to ensure relevance and precision.

1 INTRODUCTION

Theorem proving stands as a fundamental aspect of mathematics and computer science, involving the verification of propositions through rigorous logical reasoning. In recent years, interactive theorem proving (ITP) has gained prominence, allowing human experts to construct proofs interactively using proof assistants such as Lean (15), Coq (6), and Isabelle (35). These proof assistants enable the formalization and mechanical checking of mathematical proofs, ensuring correctness and eliminating ambiguities (22; 13). The increasing complexity of modern mathematical theories and the need for precise verification in critical software systems have further highlighted the importance of efficient and user-friendly theorem-proving tools.

Building on advances in machine learning, particularly the development of large language models (LLMs), researchers have explored the application of these models to automate theorem-proving tasks (39; 27; 50). LLM-based provers are typically trained on large datasets extracted from existing proof libraries, such as Mathlib (33) in Lean, aiming to generate proofs autonomously without human intervention. Notable models like GPT-f (39) and LeanDojo (50) have demonstrated promising results in generating proof steps and even entire proofs, showcasing the potential of LLMs in formal reasoning tasks. These models leverage the vast amounts of data and the expressive power of neural networks to learn patterns in mathematical reasoning, pushing the boundaries of automated theorem proving.

Despite these advancements, several significant challenges remain that hinder the practical deployment of LLMs in theorem proving. First, integrating LLM-based provers with existing proof assistants is non-trivial, as these models often operate outside the native environments of proof assistants, limiting their utility for mathematicians and developers who rely on tools like Lean for formal verification. The lack of seamless integration means that users cannot readily benefit from LLM assistance within their usual workflows. Second, autonomous LLM provers frequently struggle with complex or novel theorems that require deep human intuition and expertise, especially when the theorems

originate from domains or subfields that are underrepresented in their training data (51). This limitation underscores the difficulty of generalizing machine-learned models to diverse mathematical domains. Third, there is a lack of effective mechanisms for human-AI collaboration within theorem-proving workflows, preventing LLMs from assisting users in real-time and leveraging the strengths of both human intuition and machine automation. Without such collaboration, the potential of LLMs to augment human capabilities remains untapped.

Our intuition is that instead of attempting to replace human mathematicians, LLMs can significantly enhance the theorem-proving process by acting as collaborative assistants. By integrating LLMs directly into the proof assistant environment, we can leverage their capabilities to automate routine and tedious tasks, suggest relevant proof steps, and assist with complex reasoning, all while keeping the human in control of the proof development process. This symbiotic relationship can capitalize on the strengths of both humans and machines, leading to more efficient and effective theorem proving. We believe that human intuition and creativity, combined with the computational power and pattern recognition abilities of LLMs, can overcome the limitations faced when either works in isolation.

The motivation behind our approach is twofold. First, we aim to enhance the user experience for mathematicians and formal verification experts by making proof assistants more accessible, intuitive, and efficient through AI assistance. By reducing the manual effort required for routine proof steps, users can focus on the more creative and conceptual aspects of theorem proving. This not only increases productivity but also lowers the barrier to entry for newcomers to formal methods. Second, we seek to create a positive feedback loop where improved proof automation tools lead to the formalization of more mathematical concepts and the development of richer proof libraries. This, in turn, provides more extensive and higher-quality datasets for training future LLMs, further advancing the capabilities of AI in theorem proving. By fostering this virtuous cycle, we aim to contribute to the long-term progress of both mathematical formalization and AI research.

In this work, we introduce ProofRefiner, a novel framework that integrates LLM inference directly into the Lean proof assistant environment. ProofRefiner enables the development of various proof automation tools that assist users in theorem proving without disrupting their existing workflows. Key aspects of our approach include seamless integration with Lean, allowing LLMs to access the proof state and suggest tactics in real-time; the creation of versatile proof automation tools such as suggest_tactics, search_proofs, and select_premises; support for custom models where users can leverage pre-trained LLMs or integrate their own, accommodating both local and cloud-based execution; and an efficient implementation utilizing Lean's foreign function interface (FFI) and optimized inference techniques, ensuring that ProofRefiner runs effectively on various hardware setups, including machines without GPUs.

Specifically, `suggest_tactics` is a tool that proposes the next proof steps based on the current goals and context, helping users navigate through complex proofs. `search_proofs` combines LLM-generated steps with existing tactics like `aesop` (31) to enhance automation and potentially discover proofs that are non-obvious to both the user and standard automation tools. `select_premises` aids in identifying relevant lemmas and definitions that could be useful for the current proof, addressing the challenge of information overload in large libraries like Mathlib. By providing these tools, ProofRefiner enhances the theorem-proving experience and expands the capabilities of Lean as a proof assistant.

To assess the effectiveness of ProofRefiner, we conduct experiments by testing its tools on selected exercises from the Mathematics in Lean book (1), a widely used resource for learning Lean and formalizing mathematical proofs. Our evaluation focuses on measuring the improvement in proof automation and the enhancement of human-AI collaboration. The results demonstrate that ProofRefiner's `search_proofs` tool proves more theorems than the standard `aesop` tactic, showcasing enhanced automation capabilities. Furthermore, in collaborative settings, ProofRefiner significantly reduces the number of manual tactics required from users, making the theorem-proving process more efficient and user-friendly. Users benefit from real-time assistance and seamless integration, validating the practicality and utility of our approach. These findings indicate that ProofRefiner not only augments the capabilities of Lean but also positively impacts the productivity of its users.

Our work makes the following key contributions:

1. **Introduction of ProofRefiner**: We present a novel framework that integrates LLM inference into the Lean proof assistant, enabling collaborative theorem proving and bridging the

gap between advanced LLMs and practical proof automation. This integration facilitates real-time assistance and enhances the interactive experience in Lean.

2. **Development of Proof Automation Tools**: We develop and implement tools such as `suggest_tactics`, `search_proofs`, and `select_premises`, which enhance the theorem-proving process by leveraging the capabilities of LLMs within Lean. These tools are designed to be user-friendly and easily accessible within the existing Lean environment.
3. **Support for Custom Models**: We provide a flexible platform that allows users to integrate their own LLMs and develop additional proof automation tools, accommodating diverse computational resources and research needs. This extensibility ensures that ProofRefiner can adapt to future advancements in LLM technology.
4. **Experimental Validation**: We conduct comprehensive experiments demonstrating ProofRefiner's effectiveness, showing improved assistance over existing rule-based systems and highlighting the benefits of human-AI collaboration. Our evaluation provides empirical evidence of the practical advantages offered by our approach.
5. **Open-Source Release**: We release ProofRefiner as an open-source implementation[1], facilitating further research, adoption, and community contributions in the field of LLM-assisted theorem proving. By making our work accessible, we aim to foster collaboration and innovation within the community.

We believe that ProofRefiner represents a significant step toward fostering effective human-AI collaboration in theorem proving, enhancing the capabilities of proof assistants, and advancing the integration of artificial intelligence in mathematical reasoning. By addressing the challenges of integration, usability, and collaboration, our work contributes to the broader goal of making formal methods more accessible and powerful, ultimately benefiting both the mathematical and computer science communities.

2 METHODOLOGY

In this section, we describe the methodology used to develop and evaluate ProofRefiner, a framework that integrates large language models (LLMs) directly into the Lean proof assistant to enhance theorem proving. Our approach focuses on the seamless integration of LLMs, the development of automated proof tools, and the experimental evaluation of the system's performance in assisting with formal verification tasks. The methodology ensures that LLMs augment, rather than disrupt, the natural workflow of theorem proving within Lean, offering real-time, context-aware suggestions to users.

2.1 SEAMLESS INTEGRATION OF LLMS INTO LEAN

To enable real-time assistance within Lean, we designed ProofRefinerto tightly integrate LLM inference directly into the proof assistant's environment. This integration allows LLMs to interact with the proof state, providing relevant tactic suggestions or proof completion based on the current goals. The primary challenge in achieving this was ensuring that LLMs could communicate efficiently with Lean's internal processes without introducing latency or interrupting the user experience.

The integration is achieved using Lean's foreign function interface (FFI), which enables external functions, such as those written in C++ or Python, to be called directly from Lean's environment. Specifically, we leveraged this mechanism to wrap pre-trained LLM models, such as ReProver from LeanDojo (50), into a format that Lean can invoke. The FFI acts as a bridge between Lean's internal proof engine and the external LLM, enabling tight synchronization between proof states and LLM suggestions.

Key aspects of this integration include:

Real-time Proof Assistance: At each step in the proof development process, the LLM has access to the current proof state, represented as a tuple (Γ, Δ, G), where Γ is the set of assumptions, Δ is the context, and G is the goal to be proven. The LLM processes this information to suggest relevant tactics or proof steps:

[1]ProofRefiner codebase: `https://github.com/ProofRefiner`

$$T = \text{LLM}(\Gamma, \Delta, G), \tag{1}$$

where T is the set of suggested tactics. These suggestions are displayed in real-time, allowing the user to accept or modify them as needed, streamlining the theorem proving process.

Efficient Execution: The system is optimized to run efficiently on both high-performance computing environments and resource-constrained hardware. By employing techniques such as quantization and model distillation, we reduce the computational load of LLM inference. Given that LLM models typically require significant processing power, we implemented an inference strategy where the model only processes necessary portions of the proof state, minimizing the time complexity to $O(n)$ per query, where n is the number of proof state elements considered relevant for a suggestion.

Furthermore, we employ Lean's asynchronous execution model, allowing inference to run concurrently with user actions. This ensures that suggestions from the LLM appear instantaneously, maintaining a smooth interaction without blocking the user's workflow.

Support for Custom Models: While the framework is designed to work seamlessly with pre-trained models like ReProver, ProofRefineralso supports user-provided models. Custom models can be imported by wrapping them into a Lean-compatible API using the FFI. This flexibility allows for both local and cloud-based execution of models, enabling deployment on a variety of hardware configurations. For example, users may leverage powerful cloud-based GPUs for training and fine-tuning their models while running lightweight inference locally on CPUs.

2.2 Automated Proof Automation Tools

Beyond basic real-time assistance, ProofRefinerincludes proof automation tools that leverage LLM capabilities to automate portions of the proof process. These tools build upon Lean's existing tactic framework but augment it with LLM-powered heuristics to increase their effectiveness.

Tactic Generation: LLMs in ProofRefinergenerate proof tactics by analyzing the structure of the current proof goal and the context. The model learns from large corpora of formal proofs and generates tactics T that have a high probability of advancing the proof:

$$T = \text{argmax}_{t \in \mathcal{T}} P(t|\Gamma, \Delta, G), \tag{2}$$

where $\mathcal{T}$ is the set of available tactics, and $P(t|\Gamma, \Delta, G)$ is the probability that tactic t is correct given the current state (Γ, Δ, G). The model ranks possible tactics and returns the highest-probability suggestions.

Goal Decomposition: The LLM can assist in breaking down complex goals into simpler sub-goals. For a goal G, the system proposes a decomposition into sub-goals $G_1, G_2, \ldots, G_k$ such that proving each G_i leads to the proof of G. This is formalized as:

$$G \longrightarrow (G_1, G_2, \ldots, G_k), \tag{3}$$

where k is the number of sub-goals generated. The decomposition is designed to help users focus on smaller, more tractable proof obligations, making the overall proof development more manageable.

Proof Completion: In some cases, the LLM can suggest the entire sequence of tactics needed to complete a proof from the current state. Given the proof state (Γ, Δ, G), the LLM suggests a sequence of tactics $T_1, T_2, \ldots, T_n$ that leads directly to the proof of G:

$$T_1, T_2, \ldots, T_n = \text{LLM}(\Gamma, \Delta, G), \tag{4}$$

where the sequence is computed such that applying all T_i proves G. This level of automation is especially useful in routine or well-understood proofs, allowing users to focus on more creative or complex parts of their work.

2.3 DEVELOPMENT OF PROOF AUTOMATION TOOLS

To assist users in constructing and verifying proofs more efficiently, we implemented a suite of proof automation tools within ProofRefiner. These tools leverage the power of large language models (LLMs) to reduce manual effort, streamline proof discovery, and provide intelligent suggestions during the proof development process. The tools are designed to seamlessly integrate with Lean's existing tactics and libraries, enhancing both the interactivity and automation of the proof assistant. Below, we describe the key proof automation tools developed as part of ProofRefiner:

- **`suggest_tactics`:** This tool provides real-time suggestions for the next logical proof step based on the current proof state. The tool analyzes the goal G, the context Δ, and the set of available assumptions Γ to generate a ranked list of possible tactics T. The suggestions are derived from both LLM-generated knowledge and Lean's internal tactics:

$$T = \text{argmax}_{t \in \mathcal{T}} P(t|\Gamma, \Delta, G), \tag{5}$$

  where $\mathcal{T}$ is the set of tactics, and $P(t|\Gamma, \Delta, G)$ represents the likelihood that tactic t will be effective in advancing the proof. This tool reduces the cognitive load on users by automating routine steps and highlighting the most appropriate tactics based on the current context. For example, if the goal is to prove an equality, the tool might suggest tactics such as `refl`, `rw`, or `simp`, depending on the context and prior successful tactics in similar proof states.

- **`search_proofs`:** This tool enhances proof automation by combining the inference capabilities of LLMs with Lean's existing tactics, such as `aesop`, to discover non-obvious proofs that standard automation tools may overlook. The tool works by generating proof steps that are then fed into a search algorithm, which explores the space of possible proof sequences to find valid proofs:

$$\text{Proof} = \text{Search}(\{\text{LLM-generated steps}\}, \{\text{Lean tactics}\}), \tag{6}$$

  where the search algorithm evaluates combinations of LLM-generated proof steps and standard tactics to construct a valid proof. By leveraging the LLM's ability to propose unconventional steps, `search_proofs` is particularly useful in complex proofs where traditional automation may fail to find a solution. This tool expands the horizon of automation by enabling Lean to solve more challenging theorems autonomously, often reducing the number of manual interventions required to complete a proof.

- **`select_premises`:** A significant challenge in working with large libraries like Mathlib is the overwhelming amount of available lemmas, theorems, and definitions. `select_premises` addresses this challenge by intelligently narrowing down the most relevant premises for the current proof goal. The tool analyzes the goal G and the context Δ, and then searches through the available library to return a ranked list of premises $\mathcal{P}$ that are most likely to be useful:

$$\mathcal{P} = \text{argmax}_{p \in \text{Mathlib}} P(p|G, \Delta), \tag{7}$$

  where $P(p|G, \Delta)$ is the likelihood that premise p is applicable to the current goal and context. This tool saves users from having to manually sift through large amounts of information, making it easier to focus on the most relevant lemmas or definitions. By reducing information overload, `select_premises` improves the efficiency of proof development, especially when dealing with unfamiliar or highly complex mathematical domains.

These automation tools are designed to work in harmony with Lean's standard proof environment, allowing users to interact with them through familiar interfaces such as the Lean REPL (read-eval-print loop) or interactive proof scripts. Importantly, the tools are non-intrusive, meaning that they complement rather than replace manual proof development. Users can opt to follow the suggestions provided by the tools, modify them, or ignore them entirely, ensuring flexibility in how the automation is utilized.

By incorporating LLMs into these tools, we aim to enhance the capabilities of Lean as a proof assistant, offering both novice and expert users a more efficient and intuitive proof development

experience. Each tool reduces the manual effort required in theorem proving by automating routine tasks, improving proof discovery, and intelligently managing large libraries of mathematical knowledge.

3 EXPERIMENTS

In this section, we empirically validate our hypothesis that human-AI collaboration in interactive theorem proving (ITP) within Lean is significantly enhanced by the ProofRefiner framework. We focus on the evaluation of our proof automation tools in two key areas: autonomous theorem proving and human-assisted theorem proving. Specifically, we compare the performance of our LLM-based `search_proof` tool against both the state-of-the-art rule-based proof automation tool `aesop` and the tactic suggestion tool `suggest_tactics`.

The theorem proving process in Lean predominantly relies on tactic-style proofs, making tactic suggestion and proof search the primary evaluation points for our framework. We perform two types of comparisons: (1) autonomous theorem proving, where the tool independently solves a theorem, and (2) human-AI collaboration, where the tool assists a human user in completing proofs. Additionally, we compare `search_proof` with `suggest_tactics` to demonstrate the advantages of full proof search over individual tactic recommendations.

3.1 HUMAN-AI COLLABORATION PARADIGM

We investigate how effectively the ProofRefiner framework can support human users in ITP, inspired by the paradigm of AI copilots in software programming, such as GitHub Copilot (11). In this context, the human user attempts to solve a goal with the help of AI by calling the copilot (our proof automation tools) at each step. If the copilot fails to solve the goal, the user proceeds with a single step and then retries the copilot on the simplified goal. This iterative approach is repeated until either the copilot successfully solves the remaining goal, or the user completes the proof without AI assistance. Through this design, we aim to quantify how much human effort can be automated by each proof automation tool.

3.2 DATASET AND EXPERIMENTAL SETUP

We conduct experiments on 50 theorems selected from the "Mathematics in Lean" textbook (1), which contains 233 exercises across various mathematical domains such as sets, functions, topology, and measure theory. The average number of tactics required to complete each selected proof is 5.52.

Each theorem has an associated ground-truth proof, consisting of one or more tactics. To simulate human interaction, we input the proof tactics step by step and apply each proof automation tool after every tactic. For `aesop`, we used it in its default configuration without manual rule adjustments. In the case of `suggest_tactics`, a goal is considered solved when one of its tactic suggestions successfully completes the proof. For each theorem, we record how many tactics the user must manually input before the tool succeeds, aiming for zero manual inputs when the tool autonomously solves the proof.

3.3 RESULTS

Table 1 provides a summary of our experimental findings. In the autonomous theorem proving setting, `search_proof` outperformed both `aesop` and `suggest_tactics`, solving 64% of the theorems independently. In contrast, `aesop` could autonomously solve only 12% of the theorems, while `suggest_tactics` solved 34%.

In the human-assisted setting, `search_proof` demonstrated the most efficient collaboration, requiring an average of only 1.02 manually entered tactics per theorem. This is a significant improvement over `aesop` (3.62 manual tactics) and `suggest_tactics` (2.72 manual tactics).

Additionally, we computed the percentage of proof steps automated by each tool across all tested theorems. On average, `search_proof` automated 81.2% of the steps, compared to 48.6% for `suggest_tactics` and 35.2% for `aesop`. This means that `search_proof` automated proof steps 1.67 times more effectively than `suggest_tactics` and 2.31 times more effectively than

aesop, demonstrating the clear advantage of leveraging LLMs for proof search over both rule-based and tactic suggestion approaches.

Table 1: Performance of suggest_tactics, aesop, and search_proof on 50 theorems from "Mathematics in Lean" (1). search_proof significantly outperforms both baselines in autonomous theorem proving and human-assisted settings.

Method	Avg. # of manually entered tactics (↓)	% of theorems proved autonomously (↑)	Avg. % of proof steps automated (↑)
aesop	3.62	12%	35.2%
suggest_tactics	2.72	34%	48.6%
search_proofs	**1.02**	**64%**	**81.2%**

3.4 Discussion

The results clearly demonstrate the value of incorporating LLM-based proof search into the ITP workflow. search_proof's superior performance in both autonomous and human-assisted proving showcases its potential to significantly reduce the manual effort required in formal theorem proving. Moreover, the framework's ability to integrate LLM inference natively within Lean enhances the proof development process by providing a seamless and interactive experience for human users. These findings suggest that LLM-based tools like search_proof can play a transformative role in making formal methods more accessible and efficient for mathematicians and software engineers alike.

4 Case Study

In this section, we present a case study demonstrating how ProofRefiner, the framework for integrating large language models (LLMs) into Lean, enhances theorem proving through the proof automation tools described earlier.

4.1 Problem Setting

We consider a scenario where a user is working with Lean to prove a theorem in group theory. The goal is to prove that the product of the inverses of two elements in a group is the inverse of their product:

$$\forall a, b \in G, \quad (a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$$

Where:

- G is a group with the group operation $\cdot$ (multiplication).
- a^{-1} and b^{-1} denote the inverses of elements a and b, respectively.

Without ProofRefiner, the user would typically apply tactics such as group, simp, or rw (rewrite), but this can be time-consuming, especially when dealing with more complex proofs.

4.2 Using ProofRefinerfor Proof Automation

With ProofRefinerintegrated into the Lean environment, the user can leverage automated proof assistance to streamline the proof development process. Tools such as suggest_tactics, search_proofs, and select_premises offer intelligent suggestions at each step.

4.3 Step-by-Step Case Study

4.3.1 Step 1: Initial Proof State

The user starts with the initial goal:

$$\text{Goal:} \quad (a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$$

The user invokes the `suggest_tactics` tool to get real-time suggestions based on the current proof state. The LLM analyzes the goal and suggests the next logical step, which might involve applying the inverse property of groups. The suggestion could be:

```
suggest_tactics
```

LLM Suggestion:

rw [inv_mul_eq_inv_mul_inv]

This applies the group property that the inverse of a product is the product of the inverses in reverse order. The user applies this tactic, rewriting the goal to:

$$\text{Goal:} \quad b^{-1} \cdot a^{-1} = b^{-1} \cdot a^{-1}$$

4.3.2 STEP 2: FINISHING THE PROOF

At this point, the goal is trivially true, so the user can apply the reflexivity tactic. The `suggest_tactics` tool recognizes this and suggests:

```
suggest_tactics
```

LLM Suggestion:

exact refl

This completes the proof by applying the `refl` tactic, which states that any expression is equal to itself.

4.3.3 STEP 3: EXPLORING ALTERNATIVE PROOFS WITH `SEARCH_PROOFS`

In addition to the direct suggestions from `suggest_tactics`, the user can invoke `search_proofs` to explore alternative proof strategies. This tool runs a combination of LLM-generated proof steps and existing Lean tactics such as `aesop` to find other valid proofs.

```
search_proofs
```

Output: The search returns valid proof sequences, such as:

- Apply `group`
- Apply `simp` using the lemma `inv_mul_eq_inv_mul_inv`

Both options are valid, offering different approaches to the proof. The user can now explore multiple ways to prove the theorem.

4.3.4 STEP 4: SELECTING RELEVANT LEMMAS WITH `SELECT_PREMISES`

If the user is unsure which lemmas are relevant for the proof, they can invoke `select_premises`, which suggests applicable lemmas from Mathlib based on the current goal:

```
select_premises
```

LLM Suggestion: The tool suggests the following lemmas:

- `inv_mul_eq_inv_mul_inv`: The inverse of a product is the product of the inverses in reverse order.
- `mul_inv_self`: Any element multiplied by its inverse is the identity.

By selecting the appropriate lemma, the user quickly applies the relevant theorem without manually searching through the library.

4.4 EFFICIENCY GAINS

Without ProofRefiner, the user would need to manually look up lemmas and apply tactics, making the process slower and more error-prone. The automation tools in ProofRefinersignificantly reduce the time and effort required to complete the proof by:

- Automatically suggesting relevant tactics.
- Searching for valid proof sequences.
- Narrowing down relevant premises from large libraries like Mathlib.

For this relatively simple theorem, the proof is completed in just a few steps, saving the user time and effort.

4.5 ADVANCED CASE: NON-OBVIOUS THEOREM

Consider a more complex theorem, such as proving properties of a homomorphism between two groups G and H:

$$\forall f : G \to H, \quad \text{if } f \text{ is a homomorphism, then } f(a \cdot b) = f(a) \cdot f(b).$$

In this case, `search_proofs` would combine LLM-generated steps with existing tactics like `homomorphism` to discover a non-obvious proof. The user could also rely on `select_premises` to suggest relevant lemmas about homomorphisms.

5 RELATED WORK

5.1 NEURAL THEOREM PROVING

The integration of neural networks into formal theorem proving has evolved significantly, with early approaches focusing on premise selection (25; 46; 34) and tactic generation (24; 30; 28). Early work predominantly utilized graph neural networks (GNNs)(49; 4; 5; 36; 45; 17; 40; 41) due to their ability to handle the relational structures of mathematical theorems. However, the landscape has shifted towards Transformer-based architectures(42), which have proven to be more effective in capturing the semantics of formal reasoning. Recent works (39; 26; 51; 23; 29; 27; 32; 38; 44; 18; 50; 43) focus on leveraging large language models (LLMs) for direct engagement in theorem proving tasks. These advancements, while promising, are often limited to experimental environments and lack practical, open-source tools that enable seamless integration of LLMs within proof assistants.

5.2 PROOF AUTOMATION WITHIN PROOF ASSISTANTS

Proof automation has been extensively explored through formal methods, where domain-specific decision procedures such as satisfiability modulo theories (SMT)(16), linear arithmetic(7), and commutative ring theory (21) have achieved notable success. In proof assistants like Lean, tactics such as `apply?` attempt to symbolically unify premises with goals, while general-purpose proof search tactics like `aesop` in Lean and `auto` in Coq (31) employ best-first search strategies. However, these rule-based systems rely on fixed sets of rules manually configured by users, limiting their flexibility in dynamic problem-solving scenarios.

Classical machine learning algorithms have played an essential role in proof automation. Hammers such as Sledgehammer (9; 10; 14) leverage machine learning to select premises by outsourcing goals to automated theorem provers. Tactic prediction tools like TacticToe (19) and Tactician (8) rely on k-nearest neighbors (KNN) algorithms with handcrafted features. Other approaches (37; 20) employ machine learning techniques, such as Naive Bayes and random forests, for premise selection in Lean. These approaches, while valuable, remain constrained by their dependence on traditional algorithms rather than modern neural methods.

5.3 Neural Networks in Proof Assistants

The application of neural networks and LLMs to proof assistants has gained traction in recent years, with several notable efforts (47; 2; 48; 3). These systems utilize LLMs running in Python, making external requests to integrate with proof assistants. However, none of these approaches enable the native execution of LLMs within the proof assistant environment itself. Our work seeks to overcome this limitation by providing a general framework that facilitates LLM inference natively within Lean, enabling tighter integration and more interactive proof experiences.

5.4 Human-AI Collaboration in Theorem Proving

The concept of human-AI collaboration has been explored in various domains, including software development (11), where tools like GitHub Copilot have successfully demonstrated the potential of AI as a "copilot" for developers. In the domain of formal mathematics, Collins et al. (12) explored how LLMs can assist mathematicians through natural language conversations. To the best of our knowledge, we are the first to investigate this collaborative paradigm within the context of formal theorem proving. By embedding AI tools directly into proof assistants, our framework supports real-time collaboration between humans and LLMs, bridging the gap between informal mathematical reasoning and formal verification. This novel approach empowers users to leverage AI's reasoning capabilities in a more interactive and seamless manner.

6 Conclusion

In this work, we introduced ProofRefiner, a novel framework that enables seamless integration of large language models (LLMs) within the Lean proof assistant. By leveraging foreign function interfaces (FFI), ProofRefiner provides a flexible and extensible platform that allows users to run LLMs natively in Lean, whether locally or through server-based processes. This integration enables advanced proof automation tools, such as `suggest_tactics`, `search_proof`, and `select_premises`, which significantly enhance the interactive theorem proving (ITP) experience.

Our experiments demonstrate the effectiveness of LLM-based proof automation in Lean, surpassing traditional rule-based systems like `aesop` in both autonomous and human-assisted theorem proving. The ability of LLMs to suggest tactics, search for proofs, and select relevant premises reduces the manual effort required from users and enables more efficient collaboration between humans and AI in formal theorem proving tasks.

By making LLM inference natively accessible in Lean, ProofRefiner not only opens up new possibilities for enhancing mathematical reasoning and formalization but also sets the stage for more robust human-AI collaboration in theorem proving. Future work will focus on further improving the integration of LLMs with formal methods, expanding the range of supported tasks, and exploring more complex mathematical domains to push the boundaries of what AI can achieve in interactive theorem proving.

References

[1] Jeremy Avigad and Patrick Massot. Mathematics in Lean, 2020.

[2] Edward Ayers, Alex J Best, Jesse Michael Han, and Stanislas Polu. lean-gptf: Interactive neural theorem proving in Lean. `https://github.com/jesse-michael-han/lean-gptf`, 2023.

[3] Zhangir Azerbayev, Zach Battleman, Scott Morrison, and Wojciech Nawrocki. Sagredo: automated dialogue between GPT and Lean. `https://www.youtube.com/watch?v=CEwRMT0GpKo`, 2023.

[4] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning (ICML)*, 2019.

[5] Kshitij Bansal, Christian Szegedy, Markus N Rabe, Sarah M Loos, and Viktor Toman. Learning to reason in large theories without imitation. *arXiv preprint arXiv:1905.10501*, 2019.

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

[7] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *International Workshop on Types for Proofs and Programs*, 2007.

[8] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The Tactician: A seamless, interactive tactic learner and prover for Coq. In *Conference on Intelligent Computer Mathematics (CICM)*, 2020.

[9] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.

[10] Sascha Böhme and Tobias Nipkow. Sledgehammer: judgement day. In *International Joint Conference on Automated Reasoning (IJCAR)*, 2010.

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

[12] Katherine M Collins, Albert Q Jiang, Simon Frieder, Lionel Wong, Miri Zilka, Umang Bhatt, Thomas Lukasiewicz, Yuhuai Wu, Joshua B Tenenbaum, William Hart, et al. Evaluating language models for mathematics through interactions. *arXiv preprint arXiv:2306.01694*, 2023.

[13] Mathlib Community. Completion of the liquid tensor experiment. `https://leanprover-community.github.io/blog/posts/lte-final/`, 2022.

[14] Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. *Journal of Automated Reasoning*, 2018.

[15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *International Conference on Automated Deduction (CADE)*, 2015.

[16] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq: A plug-in for integrating SMT solvers into Coq. In *International Conference on Computer Aided Verification (CAV)*, 2017.

[17] Emily First, Yuriy Brun, and Arjun Guha. TacTok: semantics-aware proof synthesis. In *Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2020.

[18] Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. *arXiv preprint arXiv:2303.04910*, 2023.

[19] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. TacticToe: learning to prove with tactics. *Journal of Automated Reasoning*, 65:257–286, 2021.

[20] Alistair Geesing. *Premise Selection for Lean 4*. PhD thesis, Universiteit van Amsterdam, 2023.

[21] Benjamin Grégoire and Assia Mahboubi. Proving equalities in a commutative ring done right in Coq. In *International Conference on Theorem Proving in Higher Order Logics*, 2005.

[22] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the Kepler conjecture. In *Forum of Mathematics, Pi*, volume 5, 2017.

[23] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[24] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A learning environment for theorem proving. In *International Conference on Learning Representations (ICLR)*, 2019.

[25] Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Eén, François Chollet, and Josef Urban. DeepMath—deep sequence models for premise selection. In *Neural Information Processing Systems (NeurIPS)*, 2016.

[26] Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. LISA: Language models of ISAbelle proofs. In *Conference on Artificial Intelligence and Theorem Proving (AITP)*, 2021.

[27] Albert Qiaochu Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. In *Neural Information Processing Systems (NeurIPS)*, 2022.

[28] Cezary Kaliszyk, François Chollet, and Christian Szegedy. HolStep: A machine learning dataset for higher-order logic theorem proving. In *International Conference on Learning Representations (ICLR)*, 2017.

[29] 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 *Neural Information Processing Systems (NeurIPS)*, 2022.

[30] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. IsarStep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2021.

[31] Jannis Limperg and Asta Halkjær From. Aesop: White-box best-first proof search for Lean. In *International Conference on Certified Programs and Proofs (CPP)*, 2023.

[32] Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, Ming Zhang, and Qun Liu. FIMO: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*, 2023.

[33] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Maciej Mikuła, Szymon Antoniak, Szymon Tworkowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. *arXiv preprint arXiv:2303.04488*, 2023.

[35] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. 2002.

[36] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *AAAI Conference on Artificial Intelligence*, 2020.

[37] Bartosz Piotrowski, Ramon Fernández Mir, and Edward Ayers. Machine-learned premise selection for Lean. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, 2023.

[38] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *International Conference on Learning Representations (ICLR)*, 2023.

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

[40] Markus Norman Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *International Conference on Learning Representations (ICLR)*, 2021.

[41] Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. Passport: Improving automated formal verification with identifiers. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2023.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[43] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Heng Liao, and Xiaodan Liang. Lego-prover: Neural theorem proving with growing libraries, 2023.

[44] Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, et al. DT-Solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[45] Mingzhe Wang and Jia Deng. Learning to prove theorems by learning to generate theorems. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[46] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[47] Sean Welleck and Rahul Saha. coq-synthesis: Coq plugin for proof generation and next tactic prediction. `https://github.com/agrarpan/coq-synthesis`, 2023.

[48] Sean Welleck and Rahul Saha. llmstep: LLM proofstep suggestions in Lean. `https://github.com/wellecks/llmstep`, 2023.

[49] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, 2019.

[50] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*, 2023.

[51] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022.