# Benchmarking Automated Theorem Proving with Large Language Models

**Vanessa Lama**[*1], **Catherine Ma**[*2], **Tirthankar Ghosal**[1]
[1]Oak Ridge National Laboratory, Oak Ridge, TN, USA
[2]Pomona College, Claremont, CA, USA
lamav@ornl.gov, zmbg2022@mymail.pomona.edu, ghosalt@ornl.gov
[*]Equal contribution

## Abstract

Theorem proving presents a significant challenge for large language models (LLMs) due to the requirement for formal proofs to be rigorously checked by proof assistants, such as Lean, eliminating any margin for error or hallucination. While existing LLM-based theorem provers attempt to operate autonomously, they often struggle with novel and complex theorems where human insights are essential. Lean Copilot is a novel framework that integrates LLM inference into the Lean proof assistant environment. In this work, we benchmark performance of several LLMs including general and math-specific models for theorem proving using the Lean Copilot framework. Our initial investigation suggests that a general-purpose large model like LLaMa-70B still has edge over math-specific smaller models for the task under consideration. We provide useful insights into the performance of different LLMs we chose for the task.

## 1 Introduction

As mathematical problems become increasingly intricate, the task of formalizing and generating verifiable math proofs becomes proportionally more challenging. The translation process from informal theorems and proofs to a standardized, machine-verifiable, formal language, requires much effort and expertise from human mathematicians, creating a steep learning curve. This challenge has raised great interest in the potential of using AI to aid in the math formalization process, and more generally in automated theorem proving(ATP). The integration of AI, specifically large language models(LLMs), in math formalization and theorem proving can not only accelerate the proof discovery process, but can also enhance the reliability and rigor of mathematical arguments by minimizing human error.

While the ultimate goal may be to achieve fully autonomous proof generation without human assistance, current ATP systems based on language models struggle when dealing with more complex proof problems. Typically, these language models employed for theorem proving come from larger base models like BERT and GPT, then finetuned on large amounts of mathematical text data such as Mathlib(mathlib Community, 2020). This limitation seen likely stems from the models' lack of flexibility when encountering mathematical areas not adequately covered by their training data. To address this inflexibility with autonomous proof generation, there are Interactive Theorem Proving (ITP) systems where proof assistants- software frameworks built for math formalization like Coq, Isabelle, and Lean- are used in conjunction with human mathematicians in the proving process. This integration allows for the aid of proof automation tools with human intuition. The use of ITP system proof assistants has become increasingly prevalent for ensuring a level of rigor and standard in formalizing mathematical language.

Lean is one such popular proof assistant as well as functional programming language for formalizing mathematics that supports ITP by offering a framework for writing and verifying proofs. Lean uses tactic style proving where proofs are generated step by step using tactics, or instructions used to manipulate the current state of a proof to the next state. From the start of a proof, users will continue the proof using the appropriate tactic based on the user's knowledge of the math problem and tactics. While a powerful proof assistant for mathematicians, it still requires a great deal of effort to find the ideal tactics. This is a common barrier across all ITP systems; thus, the integration of Large Language Models (LLMs) with proof assistants was introduced to offer intuitive, automated assistance in generating and verifying mathematical proofs. LLMs can automate tedious aspects of proof writing, such as identifying relevant lemmas and theorems and drawing from extensive math-



*Proceedings of the 1st Workshop on NLP for Science (NLP4Science)*, pages 208–218

ematical training data to uncover overlooked but crucial insights.

However, despite the potential of LLMs, existing LLM-based provers cannot assist humans in an interactive, seamless manner. Current LLM-based systems are typically trained and evaluated following machine learning standards that rely on extracted datasets from an ITP's codebase rather than within the proof assistant itself. This disconnection results in models that, while effective in a controlled environment, are difficult to integrate into the practical workflows of proof assistants. This gap between LLM training environments and proof assistant usage highlights a critical need for systems to bridge the two and enable more effective human-AI collaboration in theorem proving.

Recently, the effort to combine proof assistant, Lean, with LLMs was achieved through a new framework LeanCopilot. LeanCopilot is an open-source framework that supports users to bring in pre-trained LLMs and use/build LLM-based proof automation tools natively in Lean (Song et al., 2024). This makes LLM-based proof automation available in Lean and increasing accessibility in math formalization. It is based on LeanDojo's Reprover algorithm for tactic generation and can be brought in as a package in Lean through an IDE of choice (Yang et al., 2023). It is able to run LLMs on most laptops without the need for GPUs, a feature that increases accessibility for LLM-based proof automation. While users can create their own tools, LeanCopilot also comes with a suite of built in proof automation tools. These tools were built using CTranslate2's C++ library for efficient LLM inference with Transformer models, running it via Lean's foreign function interface(FFI). A more comprehensive review of these tools can be found in following sections:

- *suggest_tactics*: Analyzes the current proof state and recommends relevant tactics
- *search_proof*: Construct full proof for theorem
- *select_premises*: Identifies relevant premises for current proof goal

The performance of LLMs for theorem proving tasks can vary widely depending on the complexity of the proofs and the model architecture. Recent research has demonstrated that while LLMs can effectively automate portions of proof generation, their success is highly contingent on the difficulty of the mathematical problems being tackled and the specific design of the model (Xin et al., 2024; Song et al., 2024). This variability in performance motivates our work to benchmark LLMs specifically in the context of ITP, as understanding these models' strengths and weaknesses is crucial for improving their utility in automating mathematical proof generation.

In our research, we utilize LeanCopilot's ability to bring in LLMs to evaluate the performance of various LLMs for theorem proving. LLMs are commonly benchmarked for theorem proving by testing how many theorems they can generate proofs for. We evaluate the performance of LLMs trained for math tasks, Pythia2.8b, Llemma7b, LeanStarPlus7b, LeanStarCot7b, against general LLMs, Llama3-70b and ByT5. We replicate the experiments used to evaluate LeanCopilot's built-in proof automation tools but instead to benchmark LLMs for proof generation. We evaluate the performance of these different LLMs for ITP using LeanCopilot's built-in proof automation tool, `suggest_tactics`. This benchmarking will allow us to evaluate how different LLMs perform in assisting with theorem proving tasks across a selection of proof problems sourced from the Mathematics in Lean textbook. By systematically evaluating different LLMs, we aim to identify key factors that contribute to successful proof generation and highlight areas where further advancements are needed.

## 2 Related Works

The roots of ITP lie in the broader field of automated reasoning, which emerged as a distinct area of study in the mid-20th century. While ATP systems aimed to fully automate the process of deriving proofs, early researchers recognized the limitations of these systems, particularly in handling complex, domain-specific proofs that required a deeper level of human intuition and insight.

The inception of ITP was driven by the need to integrate human expertise into the proof construction process, allowing for a relationship between automated tools and human mathematicians. One of the earliest milestones in this direction was the development of the LCF (Logic for Computable Functions) theorem prover by Robin Milner in the 1970s (Milner, 1972). LCF introduced a novel approach that combined a small trusted kernel, which ensured the soundness of proofs, with a flexible

and extensible user interface that allowed human interaction. The LCF approach set a precedent for future ITP systems by emphasizing the importance of human oversight in the verification process.

The 1980s and 1990s saw the emergence of some of the most influential ITP systems, notably Coq and Isabelle. These systems were built on the foundational ideas of the LCF approach but introduced significant innovations that expanded the scope and applicability of ITP.

Coq, developed by Thierry Coquand and Gérard Huet, was based on the Calculus of Inductive Constructions (CIC), a powerful type theory that enabled the formalization of a wide range of mathematical concepts (Coquand and Huet, 1988). Coq's ability to handle inductive types and support constructive mathematics made it a versatile tool for both theorem proving and the extraction of certified programs. Coq's interactive environment allowed mathematicians to build proofs incrementally by using a rich set of tactics to guide the proof process while relying on the underlying formalism to ensure correctness.

Isabelle, developed by Lawrence Paulson, took a different approach by providing a generic framework that could support multiple logics (Paulson, 1986). Isabelle's most notable contribution was its use of higher-order logic (HOL), which allowed for the formalization of more complex mathematical structures and proofs. Isabelle's architecture was designed to be highly modular, enabling users to extend the system with custom proof strategies and tactics. This flexibility made Isabelle particularly popular in both academia and industry for formal verification tasks.

As ITP systems matured, the focus shifted towards enhancing their automation capabilities while preserving the essential role of human interaction. The concept of proof tactics, first introduced in LCF, became central to this effort. Tactics are commands that automate common proof steps, allowing users to delegate routine tasks to the computer prover while focusing on more challenging aspects of the proof.

These developments set the stage for the creation and release of Lean, a modern proof assistant developed by Leonardo de Moura and his team at Microsoft Research (Moura and Ullrich, 2021). First released in 2013, Lean was designed with a focus on combining expressive power, automation, and user-friendly interaction. It builds on ideas from earlier systems but introduces several unique features that distinguish it in the landscape of ITP tools.

Lean is based on a version of dependent type theory, similar to Coq, but it emphasizes a more unified approach to proof automation and user interaction. This is exemplified by Lean's tactic framework, which allows users to construct proofs incrementally by applying tactics—commands that automate specific proof steps. Lean's programming framework also enables users to write custom tactics in the Lean language itself, making it highly extensible and adaptable to different domains of mathematics and computer science.

Lean's integration of a tactic language is an evolution of the earlier LCF and Coq systems, which introduced tactics as a means of automating common proof steps. Lean's tactic framework has been further enhanced to support more sophisticated proof strategies, making it an effective tool for both novice users and expert mathematicians. This combination of user interaction and automation makes Lean a useful tool in the formalization of mathematics and the verification of complex systems, enabling better integration of human expertise with automated proof checking.

### 2.1 LeanCopilot

The creation of LeanCopilot opens a new avenue of accessibility for mathematicians who hope to use LLMs in their math formalization research.

Figure 1 provides a flowchart of the algorithm used by LeanCopilot for selecting relevant premises to generate tactics and full proofs. The specific algorithm highlighted for premise selection is LeanDojo's reprover algorithm which is based on dense passage retrieval. It selects top relevant premises from mathlib, a library of formalized math theorems, lemmas, and definitions across various subjects of math and uses them to generate tactics. This is the algorithm used in `suggest_tactics` and by `select_premises`. LeanCopilot's search_proof is also based off Lean's rule-based proof search tool `aesop`. `aesop` implements a tree-based search over a user-defined set of proof rules to generate the full proof (Limperg and From, 2023). However, because the proof rule search space is predefined by the user, `aesop` lacks flexibility considering it depends heavily how advantageous the rule set is. Every proof goal in the process uses the same predefined rule set even though different goals may call for different rules. `search_proof` adds to `aesop` by using

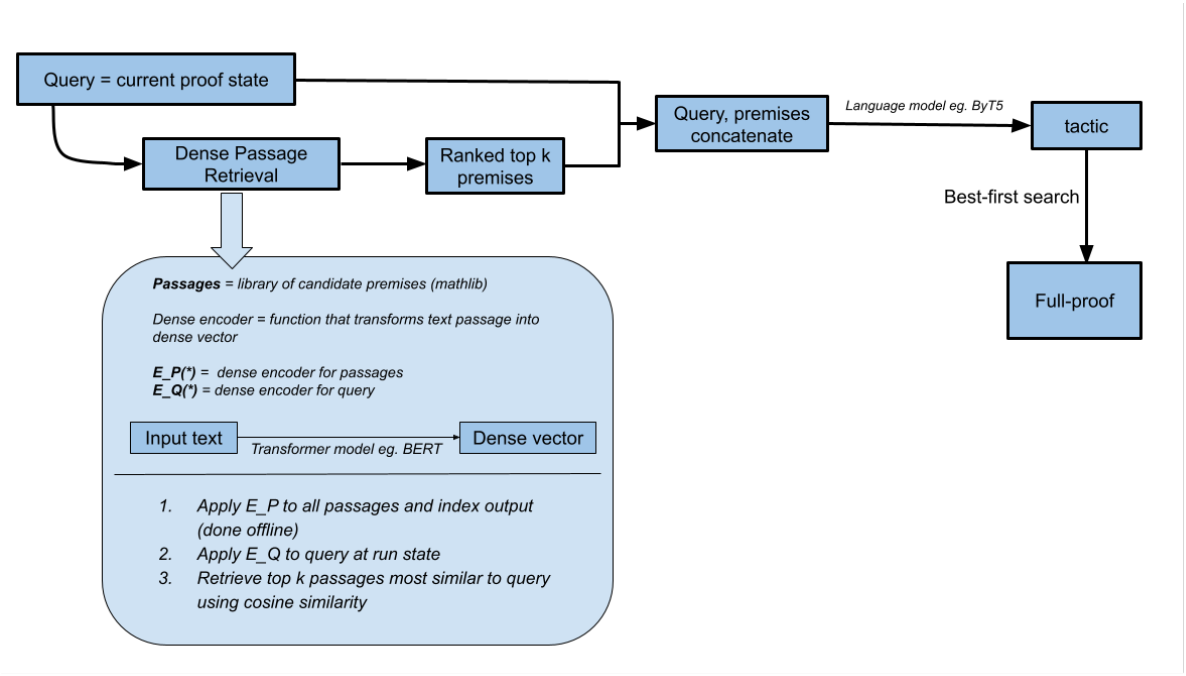


Figure 1: Flowchart of LeanCopilot proof generation algorithm used in `suggest_tactics`, `select_premises`, and `search_proof`

`suggest_tactics` to generate goal-dependent tactics for every goal thus making the rule set custom to each proof goal. In LeanCopilot's evaluations, `search_proof` outperformed `aesop` for both autonomously generating proofs and when intereacting with human users. In our experiments to benchmark LLMs, we will be swapping out the language model used to generate the tactics as seen in 1. LeanCopilot uses the language model ByT5 as its base model for its proof automation tools unless another LLM is specififed by the user.

However even with the introduction of Leancopilot, there is a lack of recorded evaluation of the capabilities of LLMs for ITP systems. We aim to bridge this gap by benchmarking LLMs to further realize how we may improve models for theorem proving. The ongoing integration of Lean with other tools and platforms, along with the potential of LLMs to enhance automation, suggests a promising future for Lean in the broader landscape of theorem proving.

## 3 Methodology

This section outlines our approach for benchmarking the capabilities of different LLMs, both math-specific and general LLMs, in assisting with theorem proving using LeanCopilot. We replicate the benchmarking experiments used to evaluate the proof-automation tools in LeanCopilot, but instead to benchmark different LLMs for proof generation. In their experiments the authors (Song et al., 2024) evaluated LeanCopilot's `suggest_tactics` and `search_proof` against preexisting lean proof automation tool `aesop`. Currently, LeanCopilot doesn't support bringing in different LLMs for their tools `search_proof` and `select_premise`, thus we will only benchmark using `suggest_tactics`.

We benchmark the following LLMs:

1. **ByT5:** Based on the T5 (Text-to-Text Transfer Transformer) architecture, specifically the T5-Small variant, which has 60 million parameters (Xue et al., 2021). It operates on byte-level inputs, eliminating the need for tokenization and making it effective for handling diverse and irregular text formats. In LeanCopilot, ByT5 serves as the foundational model, providing capabilities for generating and manipulating formal mathematical proofs.

2. **Pythia-2.8b:** This model is a specialized version of the Pythia language models, fine-tuned on the Leandojo dataset (Song et al., 2024), which consists of a curated collection of formalized mathematics in Lean. With 2.8 billion parameters, it is designed to excel in theorem proving tasks within the Lean framework. The fine-tuning on Leandojo enhances

the model's ability to generate contextually accurate proof steps and better understand the nuances of mathematical formalization, making it a valuable tool for formalizing and verifying mathematical proofs in Lean.

3. **Llama-3 70b:** LLama-3 (Large Language Model for AI) is a highly advanced transformer-based model containing 70 billion parameters (et al., 2024). Its substantial parameter count allows it to capture intricate patterns in language, making it highly effective for complex reasoning tasks, including formal theorem proving. As one of the larger models in our study, LLama-3 70B provides a benchmark for evaluating the performance of large-scale general language models in formal mathematics.

4. **Llemma7b:** Llemma7b is a mid-sized language model with 7 billion parameters, optimized for balancing computational efficiency with performance (Azerbayev et al., 2024). While not as large as LLa-ma3 70b, it offers significant capabilities in understanding and generating mathematical proofs. Its reduced size allows for more accessible deployment in resource-constrained environments, without compromising on the quality of theorem proving assistance.

5. **LeanStarPlus7b:** Lean-STaR is a framework designed to enhance language models in ATP by integrating informal reasoning with formal proof steps (Lin et al., 2024). Building on the Self-Taught Reasoner (STaR) framework (Zelikman et al., 2022), Lean-STaR introduces the concept of generating "thoughts"—natural language rationales—prior to each tactic. It operates in two phases: first, retrospective thoughts are generated by analyzing human-written proofs from Mathlib, creating a thought-augmented dataset; second, this data is used to fine-tune a tactic predictor model, which is further optimized through expert iteration. This approach significantly improves theorem-proving capabilities, as demonstrated on the miniF2F-test benchmark, where Lean-STaR achieves state-of-the-art results, surpassing previous models in pass rates.

6. **LeanCotPlus7b:** LeanCotPlus is an extension of the LeanCot model, optimized for improved interaction with the Lean proof assistant. This model builds upon LeanCot's foundational capabilities with additional enhancements aimed at increasing its effectiveness in theorem proving and proof automation. LeanCotPlus incorporates advanced techniques for better understanding and generating mathematical proofs.

We use the "Mathematics in Lean" textbook as our benchmarking data (Avigad et al., 2021). The textbook covers the math formalization process through various topics in math such as topology and logic in the Lean language. We randomly selected 50 proof problems in the textbook and evaluate how well each LLM performed at generating the full proof of each problem. Proof problems in the textbook contain "ground-truth" tactics that kick start each proof. Following the LeanCopilot experiment procedure, we will enter each ground-truth tactic one by one. After entering each tactic, we will prompt Lean with either `aesop`, `suggest_tactics`, or `search_proof` to attempt to solve the remaining proof goals. We will record the number of tactics the user had to input as prompts before the tool successfully completes the proof. The list of generated, suggested tactics is ranked top to bottom by likelihood of solving the proof. We choose the top tactic to input to eliminate human bias. For `suggest_tactics`, we consider the proof complete when in the list of suggested tactics, there exists a tactic that solves the current goal.

```
theorem primes_infinite : ∀ n, ∃ p > n, Nat.Prime p := by
  intro n
  have : 2 ≤ Nat.factorial (n + 1) + 1 := by
    sorry
  rcases exists_prime_factor this with ⟨p, pp, pdvd⟩
  refine' ⟨p, _, pp⟩
  show p > n
  by_contra ple
  push_neg  at ple
  have : p ∣ Nat.factorial (n + 1) := by
    sorry
  have : p ∣ 1 := by
    sorry
  show False
  sorry
```

Figure 2: Ground truth proof for proving there are infinitely many primes from Mathematics in Lean Textbook. The proof is purposefully not completed as an exercise for users to fill in each `sorry` with remaining tactics.

As an example of how a proof from the textbook looks, figure 2 shows the proof there exists

infinitely many prime numbers in Lean. The theorem asserts that for any natural number $n$, there exists a prime number $p$ such that $p > n$. Then the `intro` tactic introduces the variable $n$ into the proof context. This sets the stage for proving the existence of a prime number greater than $n$. Next we establish that $2 \leq (n+1)! + 1$. The factorial function grows rapidly, so adding 1 ensures that the result is greater than 1 and thus has a prime factor. The `rcases` tactic is used to deconstruct the result of the `exists_prime_factor` theorem, which guarantees the existence of a prime factor $p$ of $(n+1)! + 1$. Here, $pp$ asserts that $p$ is prime, and $pdvd$ asserts that $p$ divides $(n+1)! + 1$. The `refine'` tactic is used to fill in part of the goal, specifically stating that $p$ is the desired prime number and that $p > n$ needs to be shown. To prove $p > n$, we use a proof by contradiction. The `by_contra` tactic assumes the opposite, $p \leq n$, and the `push_neg` tactic simplifies this assumption. If $p$ divides both $(n+1)!$ and $(n+1)! + 1$, then it must divide their difference, which is 1. But no prime number can divide 1, leading to a contradiction. This final contradiction establishes that $p$ must be greater than $n$. The `sorry` placeholders represent steps where detailed proofs need to be filled in to . The tactics that exist already in the proof, are considered our "ground truth tactics". For our experiments, each `sorry` is considered a problem where we replace `sorry` with `suggest_tactics`.

## 4 Datasets on which the LLMs under consideration were trained

In this section, we discuss the datasets on which the models were trained to get an idea of their internal knowledge base.

### 4.1 ByT5

ByT5, the base model used in LeanCopilot, was initially trained on a multilingual corpus covering a broad spectrum of languages and domains. For the theorem proving tasks, it was fine-tuned on formal mathematics datasets, including those from Mathlib, to enhance its performance in proof generation.

### 4.2 Pythia2.8b

The Pythia model suite was trained on the Pile dataset, a comprehensive collection of English-language texts specifically designed for large-scale language model training (Gao et al., 2020). The Pile is highly regarded in the machine learning community because it is openly accessible, performs well across various tasks.

### 4.3 Llama3 70b

The Llama3 (et al., 2024) model was trained using a curated dataset from various sources, with data up to the end of 2023. This dataset underwent extensive cleaning and de-duplication processes to ensure high-quality tokens, focusing on removing personally identifiable information (PII) and unsafe content. The web data, which formed a significant part of the dataset, was processed using custom parsers to extract clean and relevant text while preserving the structure of mathematical and code content. The data mix for Llama3 was meticulously determined through experiments, resulting in a composition of 50% general knowledge, 25% mathematical and reasoning, 17% code, and 8% multilingual tokens, ensuring a balanced and comprehensive pre-training corpus.

### 4.4 Llemma7b

The Llemma7b model was trained on the Proof-Pile-2, a 55-billion-token dataset that combines scientific papers, web data rich in mathematical content, and mathematical code (Azerbayev et al., 2023). The dataset includes the AlgebraicStack, an 11-billion-token collection of source code from 17 languages, which emphasizes numerical, symbolic, and formal mathematics. Additionally, the training utilized OpenWebMath (Paster et al., 2024), a 15-billion-token dataset of mathematically focused web pages, and the ArXiv subset from the RedPajama dataset (Computer, 2023), contributing 29 billion tokens of scientific papers. The final data mix was heavily skewed towards mathematical and scientific content, with 95% coming from Proof-Pile-2 and small portions from general domain data and GitHub repositories.

### 4.5 LeanStarPlus7b & LeanCotPlus7b

Both models were trained and evaluated using datasets specifically curated from Lean's Mathlib (mathlib Community, 2020), the largest collection of formalized mathematics in the Lean theorem prover. Additionally, miniF2F, a standard benchmark in the formal verification community, was used to evaluate the models' performance (Zheng et al., 2022). This dataset contains a diverse set of

formalized theorems that challenge the models' ability to generalize across different mathematical domains.

## 5 Results & Discussion

In this section, we will detail the results of our experiments with the *Mathematics in Lean* textbook problems. Fifty (50) proof problems were randomly selected from the textbook for our experiments. As a reminder, for `suggest_tactics`, since the list of tactics are ranked from most likely to complete proof to least likely, *we select the first tactic in the list of generated tactics as our input to the proof*. In the following Figures, we exemplify the solution with one problem from the textbook. This example problem is *to prove the product of an even number with any natural number will be an even number*. This problem is straightforward for most with a math background, thus is a good example to understand the results produced by different models. We observe similar behaviour with our selected test-set.

### 5.1 ByT5

The results for ByT5 are shown in Figures 3 and 4. In Figure 3 we have the list of generated tactics for the first tactic. The majority of tactics from the list are some variant of the `intro` tactic aside from the tactics `norm _num`, which is used to perform numerical simplifications and arithmetic reasoning, and `simp [even_mul]`, which applies simplification rules specifically related to the property of even multiplication. ByT5 is the only model where the tactic `simp_all` was used after an `intro` tactic. In this case, it led to the immeadiate resolution of the proof since the tactic applies a broad set of simplification rules. While efficient, this may highlight a limitation of highlevel automation where the rules are not explicitly revealed.

### 5.2 LeanCotPlus7b & LeanStarPlus7b

Both LeanCoTPlus7b and LeanStarPlus7b generated the same full proof script as seen in Figures 5 and 6. This script starts with introducing the variables `m`, `n`, and the hypothesis `h`, then applies case analysis on `h`, and then simplifies the goal using the provided hypothesis and basic arithmetic. This suggests a high level of consistency in how these models approach proof generation for this of theorem. LeanCoTPlus7b and LeanStarPlus7b are indeed

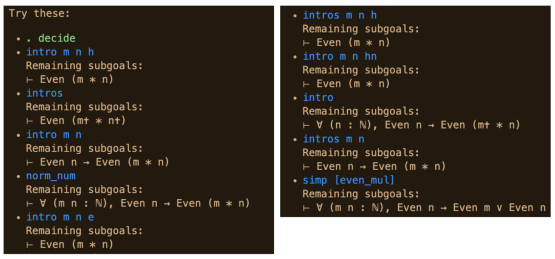


Figure 3: First list of generated tactics from `suggest_tactics` from base model ByT5. Chose first listed tactic `intro m n h` which introduces variables `m` and `n` as natural numbers and `h` as hypothesis that `n` is even.

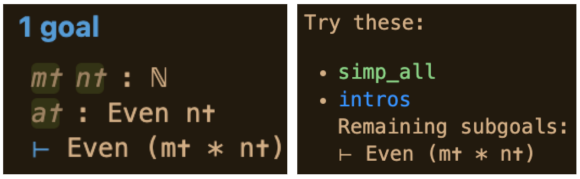


Figure 4: Left: Remaining subgoals after inputting `intro m n h` as the first tactic for ByT5 test. Suggests the renaming of `m`, `n`, and `h` to `mt`, `nt`, and `at` respectively. Remaining goal is to show `mt *nt` is even. Right: Suggested list of tactics by ByT5 after first inputted tactic; chose `simp_all` as the tactic which simplifies the current proof state, resulting in no further goals.

related models, both derived from the Lean ecosystem, which implies that they share foundational techniques and architectures for theorem proving. It is important to highlight, that both models gave the full proof of the theorem, even though it was prompted with `suggest_tactics`. This showcases their ability to generate full proofs, however this is a limitation for users who want to select their own tactic and perform ITP.

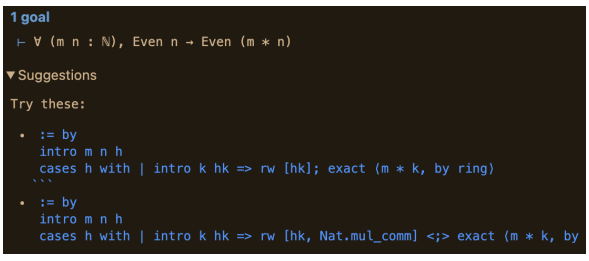


Figure 5: Generated list of tactics by LeanCotPlus7b

### 5.3 Pythia2.8b

In this experiment, Pythia2,8b generates long, broad lists of tactics at each step. This can be seen as advantageous as it may suggest various proof

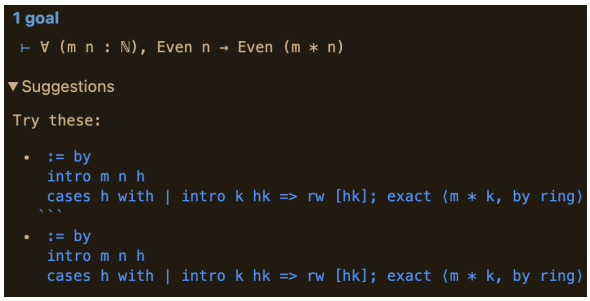


Figure 6: Generated list of tactics by LeanStarPlus7b

directions. This may also suggest that users will need a higher level of mathematical understanding to sift through all the suggested tactics in order to select one.


Figure 7: Top: Generated list of tactics by pythia2.8b. Bottom: Remaining subgoal after iniserting tactic `into m n hmn`. Tactic introduces natural number variables `m` and `n` and hypothesis `hmn`, which asserts `n` is even. Remaining subgoal is to prove `m*n` is even.

### 5.4 Llemma7b

Llemma7b adopts a methodical approach to proving the theorem, utilizing the tactics `even_iff_two_dvd` and `exact dvd_mul_of_dvd_left h m` as seen by Figures 9 and 10. The tactic `even_iff_two_dvd` translates the hypothesis into a form that asserts divisibility by 2, aligning with a common mathematical technique where the evenness of a number is proven by showing divisibility by 2. The use of tactic `dvd_mul_of_dvd_left` then applies this divisibility to the product of two numbers, completing the proof. This approach reflects a strategy often employed by human mathematicians, highlighting Llemma7b's capability to generate proofs that mirror traditional

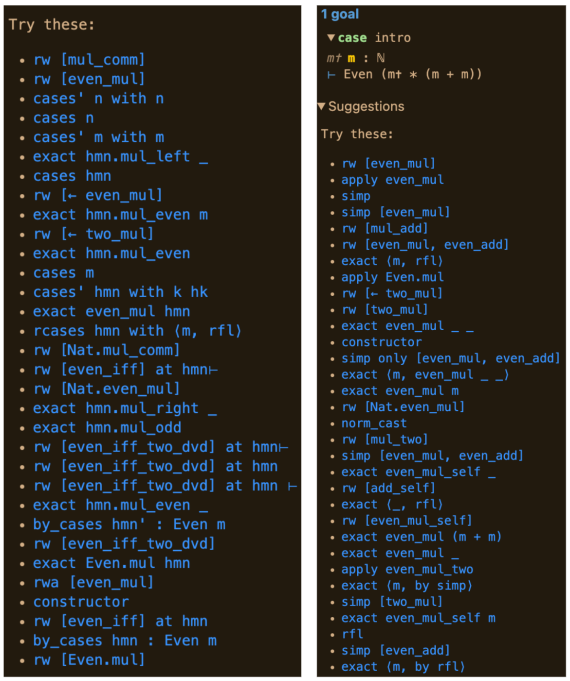


Figure 8: Left: Generated list of tactics for the second proof tactic by Pythia2.8b. Right: Remaining subgoal after inserting tactic `rw [mul_comm]` and suggested tactics for that goal.

mathematical reasoning. In contrast to the other models, which provided a broad list of tactics or rely on high-level automation, Llemma7b's proof method is distinctive in its explicit use of fundamental mathematical properties.

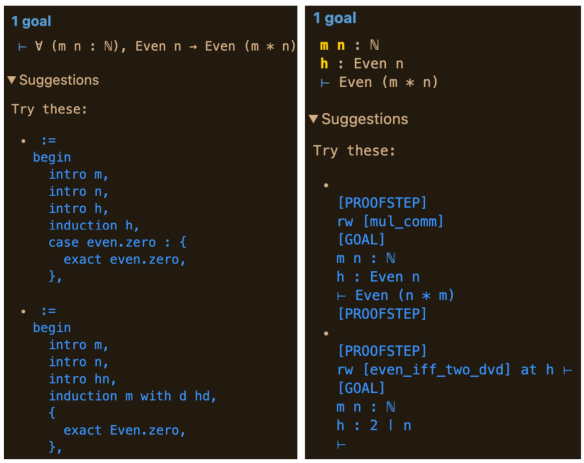


Figure 9: First and second set of tactics generated by Llemma7b

### 5.5 Llama-3 70b

As seen in Figure 11, the proof Llama-3 70b gave involves introducing the variables and hypothesis, deconstructing the hypothesis that n is even into

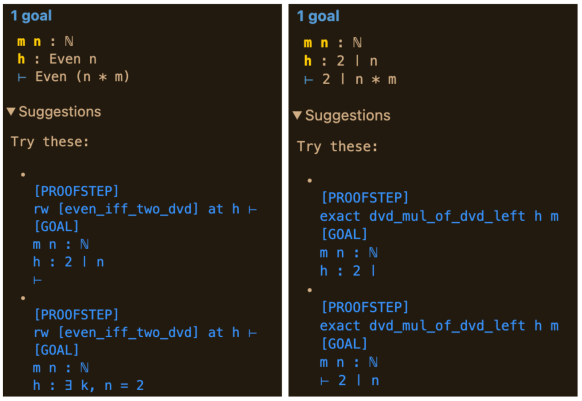


Figure 10: Third and fourth set of tactics generated by Llemma7b

the form 2 * k, and then rewriting the expression for m * n to demonstrate that it is indeed even by using known properties of even numbers. The 70 B model gave us not just one tactic but multiple tactics at a time with a majority of the steps being part of the solution. Similar to Llemma7b, it also included as part of the tactic, what the next proof goal would be for each tactic.

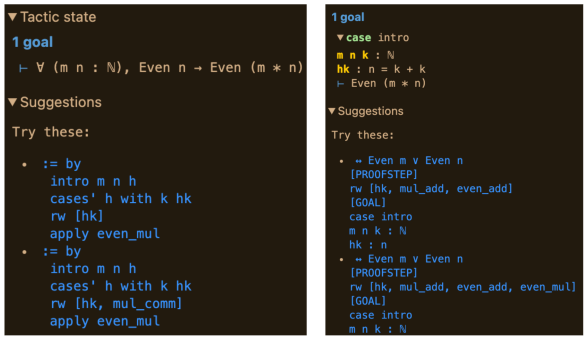


Figure 11: Progression of generated tactics by Llama-3 70b for each step of the proof from left to right

Based on our experiment results as seen in Table 1, LeanCopilot and Llama-3 70B demonstrated the highest levels of autonomy, achieving 100% autonomous proof generation with no human-entered tactics required. Pythia 2.8B also performed well, with 90% of proofs generated autonomously and an average of 0.2 human-entered tactics. LeanStarPlus 7B and LeanCoTPlus 7B both achieved 60% autonomy, with an average of 0.7 human-entered tactics, while Llemma 7B lagged behind with only 30% autonomous proofs and an average of 0.9 human-entered tactics. These results suggest that LeanCopilot and Llama-3 70B are particularly effective in fully automating proof generation, while Llemma 7B may require more human intervention in the proof process.

## 6 Observations

Our findings highlight the capabilities and diversity of modern language models in generating proofs within the Lean proof assistant environment. All models showcased capabilities for proof generation, but as can be seen by LeanCotPlus7b, LeanStarPlus7b, and Llama3-70b in our qualitative exmaple, some models have difficulty with just tactic generation as opposed to proof generation. Additionally, the tendency of some models, such as ByT5, to rely heavily on broad automation tactics like `simp_all` can obscure the underlying reasoning processes and limit interpretability. Interestingly, while Llemma7b produced very intuitive results in our example problem, in our experiments, it was the only model that lagged behind in both autonomous proof and tactic generation. These results highlight the need for future research to develop models that strike a better balance between automation and mathematical transparency.

## 7 Conclusions

In this study, we explored the integration of Large Language Models (LLMs) with Interactive Theorem Proving (ITP) systems, specifically focusing on Lean and the Lean Copilot framework. Our experiments aimed to assess the effectiveness of various LLMs in generating and automating mathematical proofs, highlighting both the potential and limitations of current technologies.

Our results demonstrated that different LLMs exhibit varied capabilities in assisting with theorem proving tasks. As shown in our qualitative example, most models appeared to rely heavily on automation, generating long lists of tactics and/or employing tactics which were vague in mathematical reasoning. Our future work would focus on benchmarking advanced models on complex theorems, developing models that enhance both automation and transparency, enabling more robust and accessible proof generation tools. Our research reinforces the promise of integrating AI with formal proof systems while highlighting areas for continued development.

## 8 Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak

| LLMs | Avg. # human entered tactics | % autonomous proof | Avg. % automated tactics |
|---|---|---|---|
| Lean Copilot | 0 | 100% | 100% |
| Pythia 2.8B | 0.2 | 90% | 85% |
| Llemma 7B | 0.9 | 30% | 55% |
| LeanStarPlus 7B | 0.7 | 60% | 66.67% |
| LeanCoTPlus 7B | 0.7 | 60% | 66.67% |
| Llama-3 70B | 0 | 100% | 100% |

Table 1: Results from suggest_tactics integrated with various LLMs.

Ridge National Laboratory supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

We would like to thank Dr. Tirthankar Ghosal for his guidance and support throughout the research. We would also like to thank Peiyang Song, for answering questions about experiments using Lean Copilot (Song et al., 2024). The code repository is available at: https://code.ornl.gov/v28/atp_lean_copilot.

## References

Jeremy Avigad, Leonardo de Moura, et al. 2021. *Mathematics in Lean*. Lean Prover Community.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *Preprint*, arXiv:2310.10631.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematicians. *ICLR*.

Together Computer. 2023. Redpajama: an open dataset for training large language models.

Thierry Coquand and Gérard Huet. 1988. The calculus of constructions. *Information and Computation*, 76(2):95–120.

Abhimanyu Dubey et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The pile: An 800gb dataset of diverse text for language modeling. *Preprint*, arXiv:2101.00027.

Jannis Limperg and Asta Halkjær From. 2023. Aesop: White-box best-first proof search for lean. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, page 253–266, New York, NY, USA. Association for Computing Machinery.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. 2024. Lean-star: Learning to interleave thinking and proving. *Preprint*, arXiv:2407.10040v1.

The mathlib Community. 2020. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM.

Robert Milner. 1972. Logic for computable functions: description of a machine implementation.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, page 625–635, Berlin, Heidelberg. Springer-Verlag.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2024. Openwebmath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations*.

Lawrence C. Paulson. 1986. Natural deduction as higher-order resolution. *The Journal of Logic Programming*, 3(3):237–258.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Towards large language models as copilots for theorem proving in lean. *Preprint*, arXiv:2404.12534.

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

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2021. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *CoRR*, abs/2105.13626.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan

Prenger, and Anima Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. *Preprint*, arXiv:2306.15626.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Preprint*, arXiv:2203.14465.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *Preprint*, arXiv:2109.00110.