# Lean Copilot: Large Language Models as Copilots for Theorem Proving in Lean

**Peiyang Song** PSONG@CALTECH.EDU
*California Institute of Technology, Pasadena, CA, U.S.A.*

**Kaiyu Yang** KAIYUY@CALTECH.EDU
*California Institute of Technology, Pasadena, CA, U.S.A.*

**Anima Anandkumar** ANIMA@CALTECH.EDU
*California Institute of Technology, Pasadena, CA, U.S.A.*

**Editors:** G. Pappas, P. Ravikumar, S. A. Seshia

## Abstract

Neural theorem proving combines large language models (LLMs) with proof assistants such as Lean, where the correctness of formal proofs can be rigorously verified, leaving no room for hallucination. With existing neural theorem provers pretrained on a fixed collection of data and offering valuable suggestions at times, it is challenging for them to continually prove novel theorems in a fully autonomous mode, where human insights may be critical. In this paper, we explore LLMs as copilots that assist humans in proving theorems. We introduce Lean Copilot, a general framework for running LLM inference natively in Lean. It enables programmers to build various LLM-based proof automation tools that integrate seamlessly into the workflow of Lean users. Lean users can use our pretrained models or bring their own ones that run either locally (with or without GPUs) or on the cloud. Using Lean Copilot, we build LLM-based tools that suggest proof steps, complete proof goals, and select relevant premises. Experimental results on the *Mathematics in Lean* textbook demonstrate the effectiveness of our method compared to existing rule-based proof automation in Lean (AESOP), confirming the significance of having LLMs integrated into the theorem proving workflow in Lean. When assisting humans, Lean Copilot requires only **2.08** manually-entered proof steps on average (3.86 required by AESOP); when automating the theorem proving process, Lean Copilot automates **74.2%** proof steps on average, **85%** better than AESOP (40.1%). We open source all code and artifacts under a permissive MIT license to facilitate further research.

**Keywords:** Neural Theorem Proving, Proof Automation, Large Language Models, Neuro-Symbolic Reasoning, AI for Mathematics

## 1. Introduction

Neural theorem proving (Li et al., 2024) combines the strong learning capability of *neural* networks with the rigor of *symbolic* proof checking. The *neural* component trains LLMs to effectively generate formal proofs, yet LLMs' tendency to hallucinate (Huang et al., 2025a) prevents the generated proofs from guaranteed correct. The *symbolic* component uses proof assistants to verify proof correctness, but the interactive nature of proof assistants require substantial amount of human efforts to formalize proofs. Note that each component's weakness is naturally complemented by the other. Combining both, neural theorem proving trains LLMs to generate candidate proofs and verifies them by proof assistants, forming a powerful neuro-symbolic system for formal reasoning (Yang et al., 2024).

On the symbolic side, Lean (de Moura et al., 2015) has been a widely used proof assistant especially in mathematical theorem proving, thanks to its large math library *Mathlib4* (mathlib

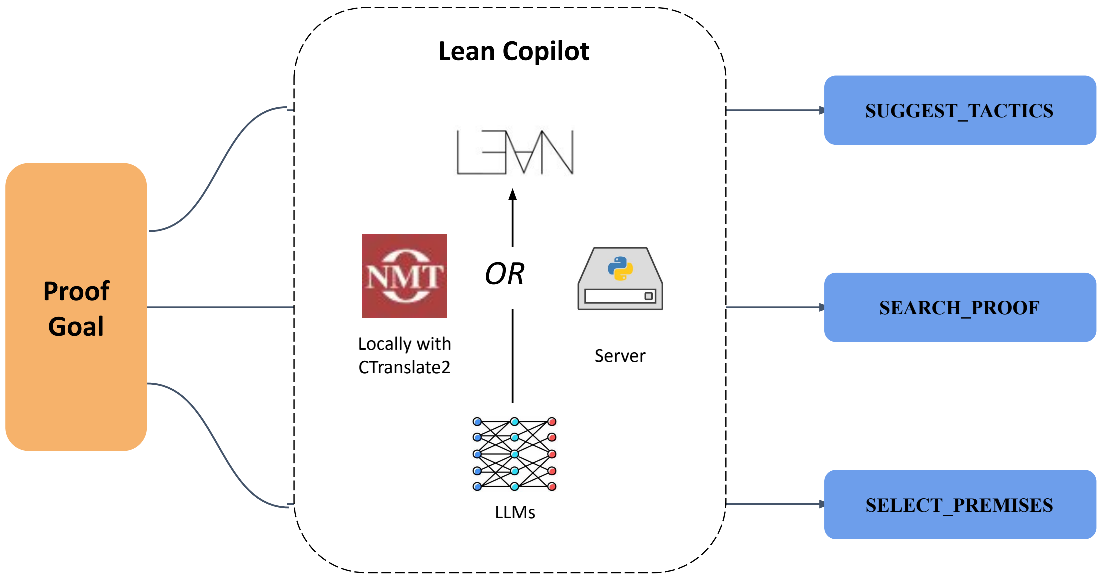


Figure 1: Lean Copilot provides a general framework for running LLM inference in Lean, either locally via CTranslate2 or on a server. This framework enables creating various proof automation tools, including those for tactic suggestion, proof search, and premise selection.

Community, 2020) with over 189K theorems[1] from diverse domains, which is still expanding. On the neural side, LeanDojo (Yang et al., 2023) novelly augments tactic generation with premise retrieval, and pairs tactic (individual proof step in Lean) generation with proof search to form complete proofs in a GPT-*f* (Ayers et al., 2023) style. Later works propose methods to further improve both tactic generation (Lin et al., 2024) and proof search (Huang et al., 2025b). An alternative approach is whole-proof generation (First et al., 2023), where synthetic data is used to address the data scarcity problem (Frieder et al., 2024). By scaling up, this approach has given birth to powerful proof-generation LLMs, marked by DeepSeek-Prover-v1.5 (Xin et al., 2024) and Goedel-Prover (Lin et al., 2025).

All these existing LLM-based provers aim to prove theorems fully autonomously without human intervention. They wrap the proof assistant (e.g., Lean) into a gym-like (Brockman et al., 2016) environment. The model interacts with the proof environment and is evaluated by the number of test theorems it proves. The interaction happens solely on the backend server, without any collaboration with humans. While an autonomous AI mathematician is desirable in the long run, current LLMs often fail to prove theorems that are truly novel or challenging, especially those from a different domain than the training data (Zheng et al., 2022). This is in part because each branch of mathematics uses different lemmas and requires distinct intuitions, limiting the generalization of LLMs. However, with the development in Lean mostly focused on Mathlib4 and other large-scale formalization projects (Gowers et al., 2023; Community, 2022), proving in new domains is inevitably important.

In practice, proving theorems in Lean requires a mix of mathematical intuition and tedious labor such as looking up premise names from the large math library. For mathematicians, the latter is particularly time-consuming. While current LLMs fall short in the former on novel theorems, we hypothesize that these LLMs are able to effectively aid the latter and save human labor. AI should act as a copilot in theorem proving: an assistant that eases routine proof construction while enabling experts to guide the overall process.

1. Statistics obtained from `leanprover-community.github.io/mathlib_stats.html`, Feb. 24, 2025.

**Lean Copilot.** A seamless collaboration between mathematicians and LLMs would require running inference of the pretrained LLMs natively in Lean, which is however a symbolic system not designed for neural network applications. To bridge this gap, we present Lean Copilot, a neuro-symbolic framework for developing LLM-based proof automation in Lean. It addresses a core technical challenge: *running LLM inference in Lean* (Section 3).

Lean Copilot provides two levels of functionalities. First, for Lean users, we build three LLM-based tools to assist theorem proving (Section 4). They offer suggestions for the next tactic (SUGGEST_TACTICS), search for a complete verified proof (SEARCH_PROOFS), and select the most relevant premises to apply next (SELECT_PREMISES). Second, for developers to build other LLM-based tools in Lean, we offer two low-level interfaces that run text-to-text and text-to-vector generations (Appendix B). Under the hood, we support running LLM inference either locally or via a server process. We use ReProver (Yang et al., 2023) as our default model due to its unique capability in premise retrieval, while we support users bringing their own models easily to Lean Copilot as well.

We evaluate (Section 5 & Appendix C) our tools on the whole *Mathematics in Lean* textbook (Avigad and Massot, 2020). SUGGEST_TACTICS and SEARCH_PROOF represent two levels of automation. SUGGEST_TACTICS generates the next *one* tactic based on a proof goal. SEARCH_PROOF further combines single tactic generation with a proof search algorithm (by default a best-first search with LLM confidence score as critic), and generates a complete proof. AESOP automates on the same level by searching for a complete proof, but its tactics are from a pre-defined list rather than generated by LLMs. Results show that SEARCH_PROOFS alone (without humans) can automate **74.2%** proof steps on average, a performance **85%** higher than the original AESOP and **27%** higher than our tactic generation tool SUGGEST_TACTICS. It can also better assist humans than the two baselines, requiring fewer tactics to be entered manually by humans (only **2.08** compared to 3.86 required by AESOP). Its significantly better performance than AESOP confirms the benefit of integrating LLMs into the theorem proving pipeline in Lean, and its enhancement over single tactic generation shows the usefulness of adding a proof search algorithm on the top.

Lean Copilot with our automation tools can be installed easily as a Lean package and run on most hardware, paving the way for wide adoption and impact on the Lean community (Appendix D). They are among the first steps in making LLMs accessible to human users of proof assistants, which we hope will initiate a positive feedback loop where proof automation leads to better data and ultimately improves LLMs on mathema. We open source all code on Github (Appendix A) at `https://github.com/lean-dojo/LeanCopilot` to facilitate future research. We make all evaluation code and detailed results public at `https://github.com/Peiyang-Song/mathematics_in_lean/tree/full-scale-experiment` for full transparency.

## 2. Related Work & Preliminaries

**Neural Theorem Proving.** Neural networks have been used to prove formal theorems by interacting with proof assistants. They can select premises (Irving et al., 2016; Wang et al., 2017; Goertzel et al., 2022; Mikuła et al., 2023), suggest tactics (proof steps) (Whalen, 2016; Huang et al., 2019; Li et al., 2021; Kaliszyk et al., 2017), and generate complete proofs (First et al., 2023; Wang et al., 2024; Xin et al., 2024; Lin et al., 2025). They can also aid in auxiliary tasks that help the development of neural theorem proving, such as conjecturing (Dong and Ma, 2025; Bengio and Malkin, 2024; Johansson and Smallbone, 2023; Poesia et al., 2024) and autoformalization (Wu et al., 2022; Azerbayev et al., 2023b; Jiang et al., 2023; Ying et al., 2024). Early works on neural theorem proving often use graph neural networks (Yang and Deng, 2019; Bansal et al., 2019a,b; Paliwal et al., 2020; Wang and Deng, 2020; First et al., 2020; Rabe et al., 2021; Sanchez-Stern et al., 2023), whereas more recent works focus on Transformer-based (Vaswani et al., 2017) language models (Polu and Sutskever, 2020; Jiang et al., 2021; Zheng et al., 2022; Han et al., 2022; Lample et al., 2022; Jiang et al., 2022; Liu et al., 2023; Polu et al., 2023; Wang et al., 2023b; First et al., 2023; Yang et al., 2023; Wang et al., 2023a). While these works have demonstrated the capability of LLMs in theorem proving, none of them has led to practical and open-source tools enabling LLMs to be used *directly* in proof assistants.

**Automation within Proof Assistants.** Proof automation has been studied extensively using formal methods. Many efficient decision procedures are available for specific domains, such as satisfiability modulo theories (Ekici et al., 2017; Martínez et al., 2019), linear arithmetic (Besson, 2007), and commutative rings (Grégoire and Mahboubi, 2005). Proof automation tools in Lean are usually wrapped up in individual tactics for a seamless integration into the normal workflow. Lean's `apply?` tactic tries to find premises that unify symbolically with the current goal. There are also general-purpose proof search tactics such as AESOP (Limperg and From, 2023) in Lean and AUTO in Coq. They search for proofs by combining a set of rules with algorithms such as best-first search. The rules are configured manually by users and fixed throughout the proof search, instead of being selected or ranked based on remaining proof goals.

Many classical machine learning algorithms have been used for proof automation. Hammers (Blanchette et al., 2016; Böhme and Nipkow, 2010; Czajka and Kaliszyk, 2018) often use machine learning for premise selection, and then outsource the proof goal and the selected premises to external automated theorem provers in first-order logic. TacticToe (Gauthier et al., 2021) and Tactician (Blaauwbroek et al., 2020) predict tactics using the k-nearest neighbors algorithm (KNN) with handcrafted features. Piotrowski et al. (2023) and Geesing (Geesing, 2023) have implemented Naive Bayes, random forests, and kNN within Lean for premise selection. There have been prior and concurrent efforts exploring using neural networks or specifically LLMs in proof assistants (Welleck and Saha, 2023a; Ayers et al., 2023; Welleck and Saha, 2023b; Azerbayev et al., 2023a). All of them run models in Python and make requests to the models from the proof assistant. In contrast, we support running LLMs natively in Lean (details in Section 3).

**Human-AI collaboration in Formal Mathematics.** Collins et al. (2023) has investigated using LLMs to assist human mathematicians by holding conversations in natural language. In formal math, Collins et al. (2024) study the patterns of human-AI interaction for theorem proving in Isabelle, by building an external platform. The paradigm of using LLMs as "copilots" to assist humans originates from software programming, as witnessed by the huge success of tools like GitHub Copilot (Chen et al., 2021; Yetiştiren et al., 2023). Such interactions usually require the AI tools to be incorporated deeply into the same environment where humans are working, so that direct interactions and collaborations can be possible. For neural theorem proving, this means having AI tools natively in the proof assistant environment. To the best of our knowledge, we are the first to address this challenge by building a general framework that can run LLM inference natively in Lean.

## 3. Lean Copilot: A General Framework for Running LLM Inference in Lean

For LLMs to assist humans in Lean, Lean Copilot provides a general framework for *running LLM inference in Lean*, which Lean programmers can use to build various LLM-based applications. Lean is usually fast in offering environment feedback. This enables Lean users to reason coherently without being interrupted by a long waiting time. Lean Copilot also needs to satisfy this requirement. Additionally, theorem proving in Lean can work well on a user's local laptop without GPUs. Lean Copilot needs to avoid adding extra hardware constraints, so it should be able to run efficiently on most hardware even without GPUs.

With the desire for fast inference and low compute requirement, since most mainstream efficient deep learning frameworks are in Python (Wolf et al., 2020; Abadi et al., 2015; Paszke et al., 2019; Chollet et al., 2015), a natural solution is to host the model in Python and make requests to it from Lean (Welleck and Saha, 2023b). However, this approach suffers from the overhead of inter-process communication, and it requires users to perform additional setup steps that do not fit naturally into Lean's conventional workflow. To overcome these issues, Lean Copilot runs LLMs *natively* in Lean through its *foreign function interface (FFI)*. All automation tools offered by Lean Copilot take a tactic form and can be applied just as any other regular tactic in Lean, with Lean Copilot itself wrapped up as a Lean package. This makes the setup and usage seamlessly integrated into Lean's workflow.

**Running LLMs in Lean through FFI.** FFI is a mechanism for programs in one language to call subroutines from another language. Lean is partially implemented in C++ and interoperates efficiently with C++. Programmers can thus declare a function in Lean but implement its body in C++. The implementation is compiled into a shared library and linked to Lean dynamically.

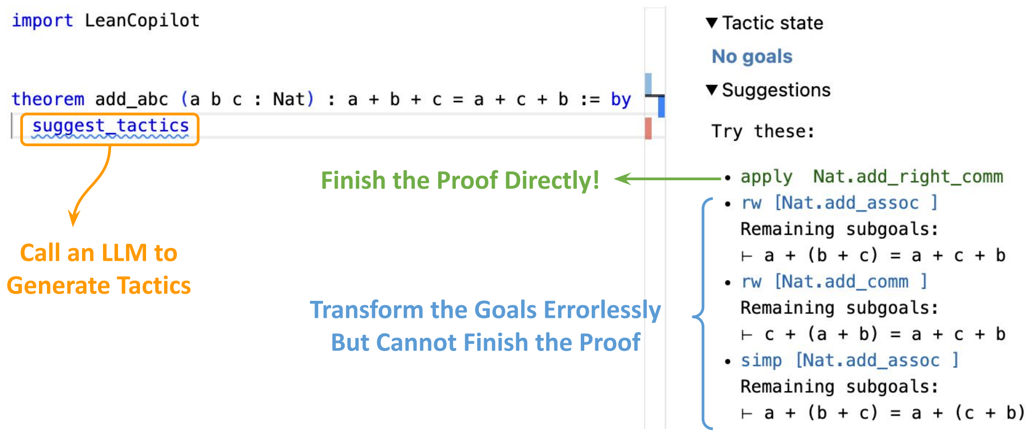
By default, we adopt the pretrained ReProver model from LeanDojo (Yang et al., 2023) for its multi-faceted capability in not only tactic generation but also premise retrieval. ReProver is based on an encoder-decoder Transformer, ByT5 (Xue et al., 2022), that maps an input string to an output string. Lean Copilot makes the model runnable in Lean by wrapping its inference into a C++ function operating on strings, which can be called in Lean through FFI. As in Fig. 1, we can run the model either locally via CTranslate2 or on a server. We use beam search for decoding the output sequence, with configurable hyperparameters such as temperature and the number of desired output sequences. This allows for multiple different output sequences being generated. We do not perform tokenization since ByT5 is a tokenizer-free model that works directly on UTF-8 bytes, so is ReProver.

ReProver additionally uses retrieval to select premises based on proof goals. It conditions on the concatenation of the retrieved premises and the proof goals to generate tactics. Its retriever is based on Dense Passage Retrieval (Karpukhin et al., 2020). With the pre-computed premise embedding from ReProver, we encode the proof goals using ReProver's encoder, and perform one forward pass with the embedding. The encoder is part of the ReProver model and can be run through FFI functions. Similarly, we run the retriever natively in Lean by wrapping its forward pass into a C++ function that takes an encoded state vector as input and outputs a vector of relevancy scores. The function processes the relevancy scores to obtain a number of highest-ranked premises and returns them as a list of strings. It can then be called in Lean through FFI.

Despite the default setup, Lean Copilot is a *general framework* that supports *user-brought models*. In principle, Lean Copilot is able to run any model with minimal changes required. Users just need to wrap it properly to expose the APIs necessary for generation and/or encoding.

## 4. Building LLM-based Proof Automation with Lean Copilot

The general framework for running LLM inference in Lean (Section 3) enables building various LLM-based tools for proof automation. In particular, we build LLM-based tools that automates three important tasks in theorem proving: tactic suggestion (Section 4.1), proof search (Section 4.2), and premise selection (Section 4.3). We also generalize Lean Copilot with low-level interfaces, one for



```
import LeanCopilot

theorem add_abc (a b c : Nat) : a + b + c = a + c + b := by
  suggest_tactics
```

```
▼Tactic state
 No goals
▼Suggestions
 Try these:
 • apply  Nat.add_right_comm
 • rw [Nat.add_assoc ]
   Remaining subgoals:
   ⊢ a + (b + c) = a + c + b
 • rw [Nat.add_comm ]
   Remaining subgoals:
   ⊢ c + (a + b) = a + c + b
 • simp [Nat.add_assoc ]
   Remaining subgoals:
   ⊢ a + (b + c) = a + (c + b)
```

Call an LLM to Generate Tactics
Finish the Proof Directly!
Transform the Goals Errorlessly But Cannot Finish the Proof

Figure 2: The frontend of Lean Copilot's SUGGEST_TACTICS. The user imports Lean Copilot just as a regular Lean package, and uses SUGGEST_TACTICS in a proof, which calls an LLM to generate tactic candidates. After filtering, tactics that transform the proof goals errorlessly (i.e. "tactic states" in Lean) are shown in the InfoView (the view on the right, a vital part in Lean's workflow). The one that finishes the proof alone is colored green; others shown in blue with their respective remaining goals.

text-to-text generation and the other for text-to-vector encoding (details in Appendix B). Developers can use those interfaces to build other proof automation tools, or bring their own models to Lean Copilot. We hope this feature encourages developers to build more proof automation in Lean.

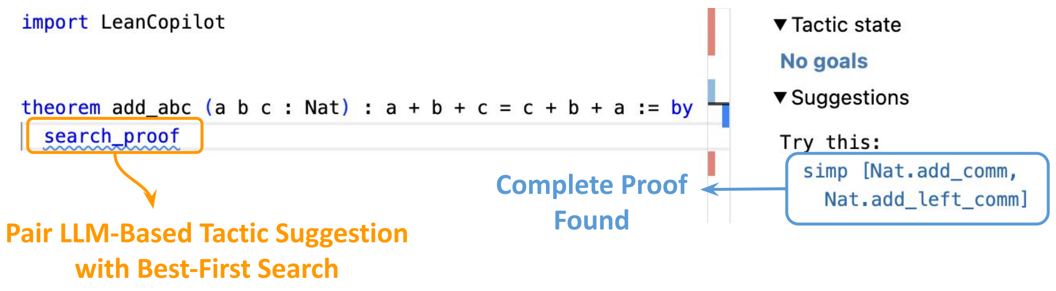
## 4.1. Generating Tactic Suggestions

When humans prove theorems in Lean, they inspect the remaining proof goals to decide for the next tactic (i.e. proof step in Lean). Tactics do not come from a predefined list; they are similar to programs in a domain-specific language (DSL). They can take arbitrary Lean terms (or take none) as parameters, and simpler tactics can be combined into compound ones. Users can also extend existing tactics by defining customized tactics. Due to these complexities, producing the right tactic can be challenging even for experienced Lean users.

**Tactic Suggestion.** We use Lean Copilot to build SUGGEST_TACTICS: a tool using LLMs to generate tactic suggestions. SUGGEST_TACTICS itself is also a tactic. When applied, it feeds the current proof goals into an LLM and obtains a list of tactic candidates. Instead of blindly returning all of them to users, we further check each tactic candidate via a meta-program in Lean that simulates running the tactic candidate on the current proof goals. There are three possible results. If the tactic candidate leads to errors, we drop them from the list. Otherwise, we display them in Lean's InfoView, and color them in green or blue based on whether the tactic alone can finish the whole proof or not.

As a result, only tactics that lead to no errors are shown to the users. We show them together with the remaining proof goals if they were applied. This is especially useful when none of the suggested tactics directly finish a complicated proof. Users can then use the remaining goals to choose tactics that simplify the proof goals in desired directions. Our frontend for displaying suggested tactics in Lean's InfoView is based on Batteries[2], the standard library in Lean. A view of the frontend is in Figure 2. Users can accept a suggested tactic simply by clicking on it. The accepted tactic will automatically replace the SUGGEST_TACTICS tactic and appear at the current position in the proof.

2. Lean 4 Batteries: https://github.com/leanprover-community/batteries.



```
import LeanCopilot

theorem add_abc (a b c : Nat) : a + b + c = c + b + a := by
  search_proof
```

```
▼Tactic state
No goals
▼Suggestions
Try this:
simp [Nat.add_comm,
  Nat.add_left_comm]
```

Figure 3: The frontend of Lean Copilot's SEARCH_PROOF. When SEARCH_PROOF succeeds, the found proof is displayed in the InfoView. It is a complete proof that is able to reduce the remaining goals to "No goals".

## 4.2. Searching for Complete Proofs

While suggesting the next proof step can be helpful, Lean proofs often consist of multiple tactics, and writing them involves trial and error. Neither humans nor machines can consistently produce the right tactic, so they have to backtrack and explore alternatives – a process called *proof search*. To address this need, we combine SUGGEST_TACTICS with AESOP (Limperg and From, 2023), a rule-based, white-box proof search tool in Lean. We empower it with LLM-generated tactics and build LLM-based proof search, wrapped up in a Lean tactic SEARCH_PROOF.

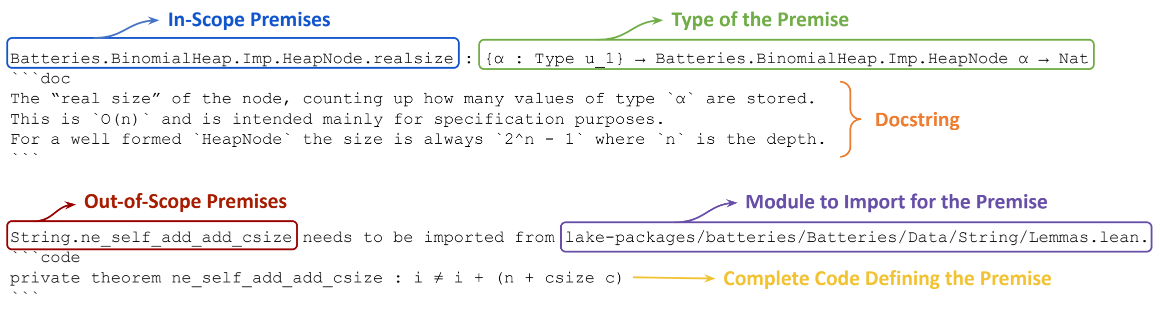
**Aesop.** AESOP implements best-first search and wraps it up in a Lean tactic AESOP. It allows users to configure how the search tree gets expanded. The search tree consists of proof goals as nodes. Initially, it has only the original goal as the root node. At each step, AESOP picks the most promising unexpanded node, expands it by applying tactics from a pre-defined set, and adds the resulting nodes as its children. The proof search succeeds when AESOP finds a path from the root to some goals that can be solved trivially. It may fail because of timeout or when AESOP has run out of tactics to try.

In AESOP, tactics for expanding nodes are drawn from a set called *the rule set*. It is configurable by users before proof search but fixed during the search, i.e., the same rule set is used for expanding all nodes, regardless of the proof goal. Its performance then depends critically on whether the user has configured an effective rule set, which is however often problem-dependent. Therefore, AESOP cannot adaptively decide what tactics to try given intermediate proof goals during the search process.

**Proof Search with LLMs.** To address this problem, SEARCH_PROOF augments AESOP's rule set with goal-dependent tactics generated by SUGGEST_TACTICS. It allows the rule set to be customized for every goal, which makes AESOP substantially more flexible. We make SEARCH_PROOF a drop-in replacement of AESOP: users can easily switch between SEARCH_PROOF and the original white-box AESOP by activating/deactivating an option for LLM-generated tactics. The frontend is similar to SUGGEST_TACTICS, except that a complete proof is shown in the InfoView instead of individual tactics. If SEARCH_PROOF succeeds, the found proof must be correct, so the color of display carries no meaning. Users can similarly click on the found proof to replace the SEARCH_PROOF tactic.

## 4.3. Selecting & Annotating Premises

Another challenging yet important task in theorem proving is to find relevant premises that can potentially advance a proof. Lean has a large mathematical library (mathlib Community, 2020) in addition to the many premises already in Lean's source code and standard library. Searching for the right premises from all the libraries can be extremely hard and labor-intensive. Some works thus


In-Scope Premises
Type of the Premise
Docstring
Out-of-Scope Premises
Module to Import for the Premise
Complete Code Defining the Premise

~~~
Batteries.BinomialHeap.Imp.HeapNode.realsize : {α : Type u_1} → Batteries.BinomialHeap.Imp.HeapNode α → Nat
```doc
The "real size" of the node, counting up how many values of type `α` are stored.
This is `O(n)` and is intended mainly for specification purposes.
For a well formed `HeapNode` the size is always `2^n - 1` where `n` is the depth.
```

String.ne_self_add_add_csize needs to be imported from lake-packages/batteries/Batteries/Data/String/Lemmas.lean.
```code
private theorem ne_self_add_add_csize : i ≠ i + (n + csize c)
```
~~~

Figure 4: Two examples of annotated premises. The premises are selected based on proof goals, and then annotated according to whether they are in scope (i.e., all necessary modules imported, ready to be used) or not. The top one is in scope, annotated with its type information and docstring; the bottom one is out of scope, annotated with the module name that needs importing to use the premise, along with its complete definition code.

try to ease this process by building dedicated library search engines for Lean[3]. Such engines can be helpful when users already know what mathematical premise they want to use, and just need to look up the formal equivalence in Lean. Yet users may not always have a firm idea about what premises to use, and a library search can fail if the developer named a particular premise differently from what the users would. The ability to select premises given proof goals is thus desired.

**Premise Selection.** A body of works tries to automate this process in Lean and other proof assistants (Alemi et al., 2017; Mikuła et al., 2024; Wang et al., 2017). The state-of-the-art tool for premise selection in Lean has been a random forest-based framework (Piotrowski et al., 2023). To bring more advanced neural architectures, we notice that premise selection is an extremely suitable task for a retrieval-augmented LLM such as ReProver, where a retrieval matrix (the *premise embedding*) is trained to estimate the relevancy between a proof goal and candidate premises. Given a proof goal at inference time, we first encode the goal into a vector, and then perform a matrix-vector multiplication between the pre-computed premise embedding and the goal vector. The result is a vector of relevancy scores for the premises, from which we can select the highest ones. For matrix-vector multiplication, we need an efficient matrix multiplication library and a numpy matrix reader in C++. We adopt the matrix multiplication functions from CTranslate2 (Authors, 2020), and a fast numpy file reader in C++ from Libnpy (Lohse, 2017). We call these C++ functions from Lean again using FFI.

**Premise Annotation.** Premises are unlike tactics or proofs: we cannot autonomously verify whether a premise is useful, as there are many ways to apply the same premise in different tactics, which may yield different outcomes. Thus, we help users more easily decide among selected premises by premise annotation via meta-programming. We categorize each selected premise depending on whether its required module is already imported in the current environment. If yes, the premise can readily be used. We then show its type and docstring (if exists). Users can compare the premise type with current proof goals, and read useful explanations from the docstring about the mathematical meaning of a premise. Otherwise, if a premise cannot be directly used due to its module not imported, we provide users with the module name and the complete code that defines this premise. The code helps users see if the definition of an out-of-scope premise is relevant and worth importing the additional module, and if so, knowing which module to import saves efforts. The annotated premises

3. An example is Moogle: https://www.moogle.ai/. It supports searching for premises using English keywords.

are displayed in Lean's InfoView. Figure 4 shows 2 examples of annotated premises, one in scope and the other not. The rest of the frontend looks the same as SUGGEST_TACTICS and SEARCH_PROOF.

## 5. Experiments

We empirically validate our hypothesis that human-AI collaboration is beneficial for theorem proving in Lean. We focus on evaluating SUGGEST_TACTICS and SEARCH_PROOF, as SELECT_PREMISES is mainly a helping tool, and it remains a challenge in the field to effectively evaluate premise selection due to the inherent lack of ground truths. We use AESOP as a state-of-the-art rule-based baseline for proof search in Lean. We compare SEARCH_PROOF with AESOP in two settings: (1) proving theorems fully autonomously (no human intervention) and (2) assisting humans in theorem proving. We also compare SEARCH_PROOF with SUGGEST_TACTICS, to demonstrate the benefit of adding a proof search algorithm on top of tactic suggestion.

We investigate how effectively Lean Copilot can assist humans in theorem proving, in a similar paradigm as humans use copilots in software programming. That is, whenever a goal exists, humans first call the copilot to see if it can solve it directly. If not, humans attempt to proceed one step and try copilots again on the remaining goals. The above procedure is repeated until the copilots successfully solve the remaining goals at a certain step, or when humans solve all steps with no useful help from copilots. We investigate how much human effort can be automated by each proof automation tool, in such an iterative collaboration paradigm. The specific experiment design is as follows.

**Dataset and Experimental Setup.** We perform experiments on theorems from *Mathematics in Lean* (MIL)(Avigad and Massot, 2020): a book for beginners to formalize and prove mathematical theorems in Lean. MIL is newly released on Github in May 2023, with parts of it made public for a short time in 2021. Both potential sources do not lead to concerns about data contamination in our experiment, because their cutoff dates are earlier than ReProver's base model ByT5, and they were not part of ReProver's open-source finetuning data. This may also explain why our tools usually come up with different proofs than ground truths in MIL in the experiment.

MIL contains theorem proving exercises that cover topics widely from sets and functions to topology, calculus, and measure theory. We evaluate on all 168 theorems in the book that have a tactic-style proof. Their proofs have 5.85 tactics on average (983 tactics in 168 theorems). The complete list of theorems, together with their detailed evaluation results, is presented in Appendix C.

Each theorem comes with a ground-truth proof consisting of one or multiple tactics. To mimic a human user, we enter the ground truth tactics one by one. After each tactic, we try to prove the remaining goals using each of the automated tools: SEARCH_PROOF, AESOP, and SUGGEST_TACTICS. For AESOP, we use it out of the box, without manually configuring the rule set. For SUGGEST_TACTICS, we say it proves a goal when one of the generated tactic suggestions can prove the goal. We record the number of tactics entered manually before the tool succeeds, so the number is zero if it can prove the original theorem fully autonomously without requiring any human-entered tactics.

**Results.** Table 1 summarizes experimental results. SUGGEST_TACTICS generates the next *one* tactic based on a proof goal. SEARCH_PROOF further combines *single* tactic generation with a proof search algorithm and generates a complete proof. AESOP automates on the same level by searching for a complete proof, but its tactics are from a pre-defined list rather than generated by LLMs. Our SEARCH_PROOF can prove **63.7%** (107 out of 168) theorems autonomously, which is significantly higher than AESOP and SUGGEST_TACTICS. When used to assist humans, SEARCH_PROOF only

| Method | Avg. # human-entered tactics (↓) | % theorems proved autonomously (↑) | Avg. % proof steps automated (↑) |
|---|---|---|---|
| AESOP | 3.86 | 24.4% | 40.1% |
| SUGGEST_TACTICS | 3.10 | 45.2% | 58.3% |
| SEARCH_PROOFS | **2.08** | **63.7%** | **74.2%** |

Table 1: Performance of SUGGEST_TACTICS, AESOP and SEARCH_PROOF on proving all theorems in "Mathematics in Lean" (Avigad and Massot, 2020) that have tactic-style proofs. SEARCH_PROOF outperforms both baselines in proving theorems autonomously and in assisting human users, requiring fewer tactics entered by humans.

requires an average of **2.08** manually-entered tactics, which compares favorably to AESOP (3.86) and SUGGEST_TACTICS (3.11). Finally, for each theorem, we calculate the percentage of proof steps that are automated by each of the three tools. Averaging the percentage over all tested theorems, we find that SEARCH_PROOF can automate about **74.2%** of the proof steps in a theorem, significantly more helpful than SUGGEST_TACTICS (58.4%) and AESOP (40.1%). SEARCH_PROOF achieves a performance **27%** better than SUGGEST_TACTICS, showing the effectiveness to add the proof search algorithm on top of our LLM-powered single tactic generation; and it's **85%** better than the rule-based baseline AESOP, showing the significantly beneficial role a LLM can play when assisting humans to prove theorems in Lean.

## 6. Conclusion

We have introduced Lean Copilot: a framework for running LLM inference natively in Lean. Using Lean Copilot, we have built LLM-based proof automation tools for generating tactic suggestions (SUGGEST_TACTICS), searching for proofs (SEARCH_PROOF), and selecting premises (SELECT_PREMISES). Lean Copilot also provides general interfaces between LLMs and Lean, allowing users to bring their own models and/or build other proof automation tools. Experimental results on the *Mathematics in Lean* textbook demonstrate the effectiveness of our method compared to existing rule-based proof automation in Lean (AESOP). When assisting humans, Lean Copilot requires only **2.08** manually-entered proof steps on average (3.86 required by AESOP); when automating the theorem proving process, Lean Copilot automates **74.2%** proof steps on average, **85%** better than AESOP (40.1%). These results confirm the benefit of integrating LLMs into the theorem proving pipeline in Lean. We open source all code and artifacts to facilitate future research, and we hope to see more LLM-based proof automation built upon Lean Copilot to help create more high-quality formal data, which would in turn enhance LLMs' capability in formal math.

## Acknowledgments

Peiyang Song was supported by the Summer Undergraduate Research Fellowship at Caltech. Kaiyu Yang was supported by the Computing, Data, and Society Postdoctoral Fellowship at Caltech. Anima Anandkumar is partially supported by the Bren endowed chair and Schmidt Sciences AI2050 senior fellowship. We appreciate the valuable feedback from members of the Anima AI+Science Lab on an initial version of this paper, from the Lean community on the releases of our tool, and from the anonymous reviewers at NeuS 2025. We thank Scott Morrison for suggestions on simplifying Lean Copilot's installation and Mac Malone for helping implement it. We thank Jannis Limperg for supporting our LLM-generated tactics in Aesop.

## References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Alex A. Alemi, Francois Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. Deepmath - deep sequence models for premise selection, 2017.

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://api.semanticscholar.org/CorpusID:268232499.

The OpenNMT Authors. CTranslate2: a c++ and python library for efficient inference with transformer models. https://github.com/OpenNMT/CTranslate2, 2020.

Jeremy Avigad and Patrick Massot. Mathematics in Lean, 2020.

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.

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

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*, 2023b.

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)*, 2019a.

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*, 2019b.

Yoshua Bengio and Nikolay Malkin. Machine learning and information theory concepts towards an ai mathematician, 2024. URL https://arxiv.org/abs/2403.04571.

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

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.

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

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. Internlm2 technical report, 2024. URL https://arxiv.org/abs/2403.17297.

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.

François Chollet et al. Keras. https://keras.io, 2015.

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.

Katherine M. Collins, Albert Q. Jiang, Simon Frieder, Lionel Wong, Miri Zilka, Umang Bhatt, Thomas Lukasiewicz, Yuhuai Wu, Joshua B. Tenenbaum, William Hart, Timothy Gowers, Wenda Li, Adrian Weller, and Mateja Jamnik. Evaluating language models for mathematics through interactions. *Proceedings of the National Academy of Sciences*, 121(24):e2318124121, 2024. doi: 10.1073/pnas.2318124121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2318124121.

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

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

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.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL `https://arxiv.org/abs/2502.00212`.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024. URL `https://arxiv.org/abs/2301.00234`.

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.

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

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.

Simon Frieder, Jonas Bayer, Katherine M. Collins, Julius Berner, Jacob Loader, András Juhász, Fabian Ruehle, Sean Welleck, Gabriel Poesia, Ryan-Rhys Griffiths, Adrian Weller, Anirudh Goyal, Thomas Lukasiewicz, and Timothy Gowers. Data for mathematical copilots: Better ways of presenting proofs for machine learning, 2024. URL `https://arxiv.org/abs/2412.15184`.

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.

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

Zarathustra A. Goertzel, Jan Jakubův, Cezary Kaliszyk, Miroslav Olšák, Jelle Piepenbrock, and Josef Urban. The isabelle enigma, 2022. URL `https://arxiv.org/abs/2205.01981`.

W. T. Gowers, Ben Green, Freddie Manners, and Terence Tao. On a conjecture of marton, 2023. URL `https://arxiv.org/abs/2311.05762`.

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.

Rhys Gretsch, Peiyang Song, Advait Madhavan, Jeremy Lau, and Timothy Sherwood. Energy efficient convolutions with temporal arithmetic. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 354–368, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640395. URL `https://doi.org/10.1145/3620665.3640395`.

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.

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

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2), January 2025a. ISSN 1046-8188. doi: 10.1145/3703155. URL https://doi.org/10.1145/3703155.

Suozhi Huang, Peiyang Song, Robert Joseph George, and Anima Anandkumar. Leanprogress: Guiding search for neural theorem proving via proof progress prediction, 2025b. URL https://arxiv.org/abs/2502.17925.

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.

Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization, 2023. URL https://arxiv.org/abs/2311.03755.

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.

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.

Moa Johansson and Nicholas Smallbone. Exploring mathematical conjecturing with large language models. In *NeSy*, pages 62–77, 2023.

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.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020. URL https://arxiv.org/abs/2004.04906.

Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. Leanagent: Lifelong learning for formal theorem proving, 2024. URL https://arxiv.org/abs/2410.06209.

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.

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.

Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. A survey on deep learning for theorem proving, 2024. URL https://arxiv.org/abs/2404.09939.

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.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-star: Learning to interleave thinking and proving, 2024. URL https://arxiv.org/abs/2407.10040.

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, 2025. URL https://arxiv.org/abs/2502.07640.

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.

Leon Merten Lohse. Libnpy: a simple c++ library for reading and writing of numpy's .npy files., 2017. URL https://github.com/llohse/libnpy.

Guido Martínez, Danel Ahman, Victor Dumitrescu, Nick Giannarakis, Chris Hawblitzel, Catalin Hritcu, Monal Narasimhamurthy, Zoe Paraskevopoulou, Clément Pit-Claudel, Jonathan Protzenko, Tahina Ramananandro, Aseem Rastogi, and Nikhil Swamy. Meta-f*: Proof automation with smt, tactics, and metaprograms, 2019. URL https://arxiv.org/abs/1803.06547.

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. ISBN 9781450370974. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

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.

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, 2024.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context, 2022. URL https://arxiv.org/abs/2110.15943.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

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.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

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.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation, 2024. URL https://arxiv.org/abs/2407.00695.

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

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.

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.

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.

Google Gemini Team. Gemini: A family of highly capable multimodal models, 2024. URL https://arxiv.org/abs/2312.11805.

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.

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, 2023a.

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)*, 2023b.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. Proving theorems recursively, 2024. URL https://arxiv.org/abs/2405.14414.

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

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.

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

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

Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic, 2016. URL https://arxiv.org/abs/1608.02644.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, 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, 2024. URL https://arxiv.org/abs/2408.08152.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics (TACL)*, 10:291–306, 2022.

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

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.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai, 2024. URL https://arxiv.org/abs/2412.16075.

Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt, 2023. URL https://arxiv.org/abs/2304.10778.

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, 2024. URL https://arxiv.org/abs/2406.03847.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL `https://arxiv.org/abs/2403.03507`.

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.

## Appendix A. Code & Artifacts

We open source *all code and artifacts* in our Github repository: `https://github.com/lean-dojo/LeanCopilot`, under a permissive MIT license. We hope that the three automation tools we have built would be able to ease theorem proving in Lean, and we hope the general neuro-symbolic framework of Lean Copilot would encourage developers to continue building more automation tools that accelerate theorem proving in Lean.

We also release all evaluation code for our experiments presented in Section 5 and Appendix C, for full transparency. The code is at `https://github.com/Peiyang-Song/mathematics_in_lean/tree/full-scale-experiment`.

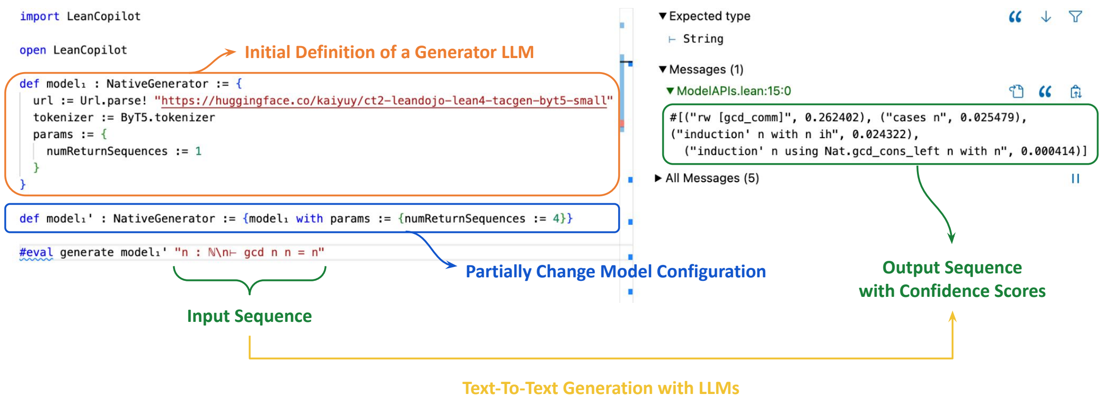
## Appendix B. Generalizing Lean Copilot with Low-Level Interfaces

Section 4 has introduced the proof automation tools that we build for three important tasks in theorem proving in Lean – tactic suggestion, proof search, and premise selection. In this section, we generalize the usability of our Lean Copilot framework for developers or advanced users who wish to engage in more low-level development and build their own proof automation tools. We provide two low-level interfaces in Lean Copilot, which correspond to two prevalent use cases of LLM inference: text-to-text generation and text-to-vector encoding. We introduce the interfaces in this section of Appendix, as they are not immediate tools from Lean Copilot but could be very valuable for developers who wish to build other proof automation tools in Lean.

**Text-to-Text Generation.** One major usage of LLM inference is text-to-text generation. Given a text sequence as input, LLMs generate an output text sequence. This is the mechanism behind SUGGEST_TACTICS and SEARCH_PROOF, where our input text sequence is the remaining goals and our output sequence is a *single* tactic suggestion. The reason why we can get multiple suggestions at once is due to beam search, which returns the top-$k$ (where $k$ is a hyperparameter controlling the number of output sequences to be returned) output sequences.

With this interface, users can directly bring their own model and configure it for generation, as shown in the definition of $model_1$ in Fig. 5 below. A `url` should be provided to load a pretrained LLM from HuggingFace or other platforms, and a `tokenizer` can be specified to tokenize inputs to the LLM. Additional configurations used for generation can be added in the `params` field.

Naturally, when using a LLM, users sometimes would like to try different configurations for generation, which requires changing part of a configuration but not all of it. We support this need by allowing users to change part of an existing model configuration directly, with simple syntax shown in $model_1'$ in Fig. 5. In this case, the user would like to use the same model as $model_1$ but with the number of return sequences set to 4 rather than 1. The infoview (Figure 5 right) shows

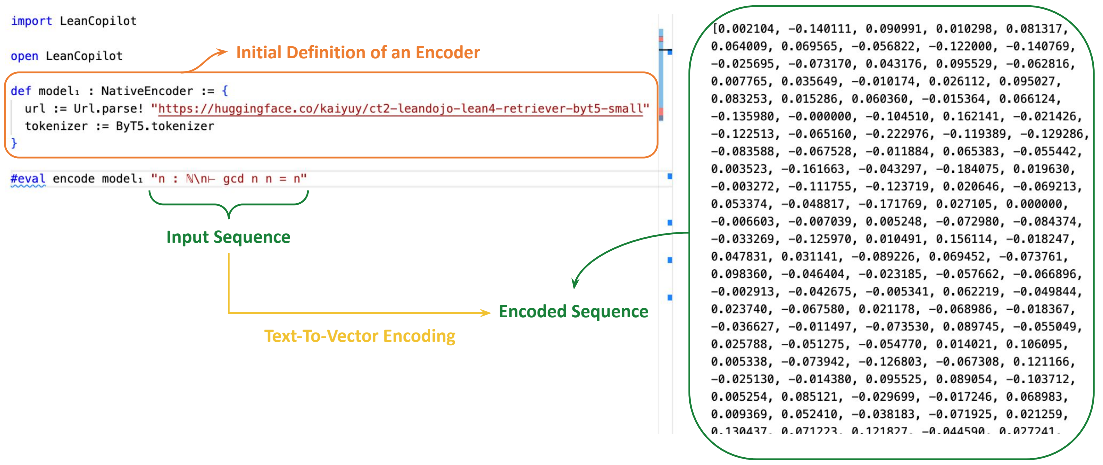

Initial Definition of a Generator LLM
Partially Change Model Configuration
Input Sequence
Output Sequence with Confidence Scores
Text-To-Text Generation with LLMs
Expected type
Messages (1)
ModelAPIs.lean:15:0
All Messages (5)

```
import LeanCopilot

open LeanCopilot

def model₁ : NativeGenerator := {
  url := Url.parse! "https://huggingface.co/kaiyuy/ct2-leandojo-lean4-tacgen-byt5-small"
  tokenizer := ByT5.tokenizer
  params := {
    numReturnSequences := 1
  }
}

def model₁' : NativeGenerator := {model₁ with params := {numReturnSequences := 4}}

#eval generate model₁' "n : ℕ\n⊢ gcd n n = n"
```

```
⊢ String
```

```
#[("rw [gcd_comm]", 0.262402), ("cases n", 0.025479),
("induction' n with n ih", 0.024322),
  ("induction' n using Nat.gcd_cons_left n with n", 0.000414)]
```

Figure 5: We provide an interface for text-to-text generation. Users can specify a model to use and configure it for generation. The model is not fixed after initial definition. If users would like to change part of its configuration, they can easily change certain the corresponding fields without the need to declare a new one. In this example, the user defines $model'_1$ by changing the number of return sequences in $model_1$ from 1 to 4. As a result, in the InfoView, $4$ sequences are returned for the #eval statement when $model'_1$ is used for generation.



```
import LeanCopilot

open LeanCopilot

def model₁ : NativeEncoder := {
  url := Url.parse! "https://huggingface.co/kaiyuy/ct2-leandojo-lean4-retriever-byt5-small"
  tokenizer := ByT5.tokenizer
}

#eval encode model₁ "n : ℕ\n⊢ gcd n n = n"
```

```
[0.002104, -0.140111, 0.090991, 0.010298, 0.081317,
0.064009, 0.069565, -0.056822, -0.122000, -0.140769,
-0.025695, -0.073170, 0.043176, 0.095529, -0.062816,
0.007765, 0.035649, -0.010174, 0.026112, 0.095027,
0.083253, 0.015286, 0.060360, -0.015364, 0.066124,
-0.135980, -0.000000, -0.104510, 0.162141, -0.021426,
-0.122513, -0.065160, -0.222976, -0.119389, -0.129286,
-0.083588, -0.067528, -0.011884, 0.065383, -0.055442,
0.003523, -0.161663, -0.043297, -0.184075, 0.019630,
-0.003272, -0.111755, -0.123719, 0.020646, -0.069213,
0.053374, -0.048817, -0.171769, 0.027105, 0.000000,
-0.006603, -0.007039, 0.005248, -0.072980, -0.084374,
-0.033269, -0.125970, 0.010491, 0.156114, -0.018247,
0.047831, 0.031141, -0.089226, 0.069452, -0.073761,
0.098360, -0.046404, -0.023185, -0.057662, -0.066896,
-0.002913, -0.042675, -0.005341, 0.062219, -0.049844,
0.023740, -0.067580, 0.021178, -0.068986, -0.018367,
-0.036627, -0.011497, -0.073530, 0.089745, -0.055049,
0.025788, -0.051275, -0.054770, 0.014021, 0.106095,
0.005338, -0.073942, -0.126803, -0.067308, 0.121166,
-0.025130, -0.014380, 0.095525, 0.089054, -0.103712,
0.005254, 0.085121, -0.029699, -0.017246, 0.068983,
0.009369, 0.052410, -0.038183, -0.071925, 0.021259,
0.130437, 0.071223, 0.121827, -0.044590, 0.027241,
```

Figure 6: We provide a second interface for text-to-vector encoding. Users can bring their own model and configure it for encoding. A given string as input will be encoded into a vector of floats.

the successfully generated text sequences. The output is from the #eval statement in the codes (Figure 5 left) which uses $model'_1$ for generation. Changing any other field shares the same syntax.

**Text-to-Vector Encoding.** Another popular use of LLM inference is text-to-vector encoding, which encodes an input text sequence into a word vector embedding. This is part of the mechanism behind SELECT_PREMISES, where our input is the remaining goals and our output is an encoded vector embedding for the input. We multiply this goal vector embedding with the pre-computed premise embedding from ReProver to get the scores of candidate premises. With this interface, users can similarly bring their own model and provide any input sequence to be encoded, as shown in Fig. 6.

Such two low-level interfaces together provide users with the freedom to bring their own models and configure the generation or encoding process to their need. Furthermore, while our native interfaces are theoretically able to perform text-to-text generation and text-to-vector encoding for any model, we provide the additional flexibility for users to use a server process instead. This can be valuable especially when users are working on a project interfacing directly between Lean and another programming language like Python.

**Running LLMs in Lean through Server Processes.** To run LLMs in Lean through server processes, users can declare an `ExternalGenerator`, where they specify a model to use, a host name, and a port number. Then once they open the corresponding server in the command line, the specified model will be able to perform text-to-text generation, and display the outputs on the infoview just like the native generators do in Fig. 5. Text-to-vector encoding can be performed in the same way, by declaring an `ExternalEncoder`. In principle, users can bring any models using Lean Copilot through `ExternalGenerator` and `ExternalEncoder`. In order to use them, users just need to expose certain APIs such as model name, input sequence, generation prefix, etc. We provide examples of using this Python API server in our codebase, which current include running models from OpenAI's GPT family (OpenAI, 2024), Anthropic's Claude family (Anthropic, 2024), Google's Gemini family (Team, 2024), InternLM2 (Cai et al., 2024), etc.

## Appendix C. Detailed Experimental Results

We show in Table 2 our complete experimental results on all theorems from Mathematics in Lean (Avigad and Massot, 2020) that have tactic-style proofs. The aggregated statistics are in Table 1. We make all evaluation code public for full transparency, at `https://github.com/Peiyang-Song/mathematics_in_lean/tree/full-scale-experiment`.

For each theorem, the link in the table below directs to the file that contains ground-truth proofs for individual theorems. The corresponding exercises are in the same folder of the repository, under the same name excluding the "solution" phrase. Our experiment details are documented in the original "problem" files, while the "solution" files with ground-truth proofs are left intact. In the "problem" files, we leave in comments SUGGEST_TACTICS, AESOP, and SEARCH_PROOF, at the position where they automated the rest of the proof. That is, if one of the three automation tactics appears particularly in the next line after the proof, that means the automation tactic has not helped with the proof at all, and all steps are written by users.

AESOP is a white-box rule-based tool, so uncommenting the line AESOP at the given position will guarantee to finish the rest of the proof, which means our evaluation at each theorem can be exactly reproduced. SUGGEST_TACTICS and SEARCH_PROOF uses neural networks under the hood, thus inherently probabilistic. It is possible that when running those two tactics again, a different tactic/proof would be generated, possibly leading to slightly different evaluation results. Therefore, to ensure transparency, in addition to the comments showing the exact positions where those two tools are able to automate the rest of the proof, we also leave in comments the tactic/proof that they generated during our run of this experiment.

Table 2: Results on all theorems from Mathematics in Lean (Avigad and Massot, 2020) that have tactic-based proofs. The "# Tactics" column shows the number of tactics in the ground truth proof. The "# Human tactics" columns are the number of human-entered tactics required for the automated tool to finish the proof. The "Auto" columns show whether the tool can prove the theorem without humans, i.e., requiring zero human-entered tactics.

| Theorem | # Tactics | AESOP | | SUGGEST_TACTICS | | SEARCH_PROOF | |
|---|---|---|---|---|---|---|---|
| | | # Human tactics | Auto | # Human tactics | Auto | # Human tactics | Auto |
| C02_S01:3 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S01:8 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S01:13 | 2 | 2 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S01:17 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S01:22 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S01:28 | 4 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S02:8 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:11 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:14 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:17 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:21 | 1 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S02:24 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S02:29 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:33 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:42 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:45 | 1 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S02:48 | 1 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S02:58 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:63 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S02:66 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S03:7 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S03:12 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S03:24 | 3 | 2 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S03:30 | 2 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S04:13 | 5 | 5 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S04:20 | 19 | 8 | No | 2 | No | 8 | No |
| C02_S04:41 | 5 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S04:48 | 9 | 5 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S04:89 | 8 | 7 | No | 6 | No | 4 | No |
| C02_S04:109 | 5 | 5 | No | **0** | **Yes** | **0** | **Yes** |

*Continued on next page*

Table 2 – *Continued from previous page*

| Theorem | # Tactics | AESOP | | SUGGEST_TACTICS | | SEARCH_PROOF | |
|---|---|---|---|---|---|---|---|
| | | # Human tactics | Auto | # Human tactics | Auto | # Human tactics | Auto |
| C02_S05:8 | 5 | 5 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S05:15 | 19 | 18 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S05:36 | 5 | 5 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S05:43 | 19 | 17 | No | **0** | **Yes** | **0** | **Yes** |
| C02_S05:64 | 5 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S05:71 | 5 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S05:108 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S05:112 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C02_S05:127 | 4 | 4 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S01:6 | 2 | 2 | No | 1 | No | **0** | **Yes** |
| C03_S01:30 | 6 | 5 | No | 1 | No | 1 | No |
| C03_S01:50 | 3 | 2 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S01:58 | 4 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S01:73 | 2 | 2 | No | 2 | No | 2 | No |
| C03_S01:80 | 3 | 3 | No | 3 | No | 3 | No |
| C03_S01:85 | 3 | 3 | No | 3 | No | 3 | No |
| C03_S01:96 | 4 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S01:127 | 2 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S01:134 | 4 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S02:28 | 5 | 5 | No | 4 | No | 4 | No |
| C03_S02:35 | 4 | 4 | No | 3 | No | 3 | No |
| C03_S02:51 | 3 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S02:62 | 3 | 3 | No | 2 | No | 2 | No |
| C03_S02:79 | 4 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S03:23 | 4 | 4 | No | 3 | No | 3 | No |
| C03_S03:29 | 3 | 3 | No | 2 | No | 2 | No |
| C03_S03:34 | 4 | 4 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S03:40 | 4 | 3 | No | 1 | No | **0** | **Yes** |
| C03_S03:46 | 8 | 8 | No | 7 | No | 7 | No |
| C03_S03:56 | 3 | 3 | No | 3 | No | **0** | **Yes** |
| C03_S03:66 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S03:71 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S03:75 | 4 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S03:81 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S03:85 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S03:94 | 10 | 4 | No | 4 | No | 4 | No |
| C03_S03:105 | 3 | 1 | No | 1 | No | **0** | **Yes** |
| C03_S04:7 | 6 | 6 | No | 6 | No | 6 | No |
| C03_S04:15 | 13 | 13 | No | 12 | No | **0** | **Yes** |
| C03_S04:34 | 9 | 7 | No | 8 | No | 7 | No |

*Continued on next page*

Table 2 – *Continued from previous page*

| Theorem | # Tactics | AESOP | | SUGGEST_TACTICS | | SEARCH_PROOF | |
|---|---|---|---|---|---|---|---|
| | | # Human tactics | Auto | # Human tactics | Auto | # Human tactics | Auto |
| C03_S04:49 | 3 | 2 | No | 2 | No | 2 | No |
| C03_S04:58 | 14 | 14 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S04:80 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S04:85 | 7 | 7 | No | 7 | No | **0** | **Yes** |
| C03_S04:60 | 4 | 4 | No | 3 | No | **0** | **Yes** |
| C03_S05:12 | 4 | 4 | No | 3 | No | **0** | **Yes** |
| C03_S05:18 | 4 | 4 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S05:24 | 5 | 5 | No | 5 | No | **0** | **Yes** |
| C03_S05:31 | 20 | 16 | No | 16 | No | 10 | No |
| C03_S05:52 | 18 | 15 | No | 10 | No | 1 | No |
| C03_S05:76 | 1 | 1 | No | 1 | No | **0** | **Yes** |
| C03_S05:79 | 9 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S05:90 | 9 | 9 | No | 7 | No | 4 | No |
| C03_S05:105 | 9 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C03_S05:116 | 9 | 9 | No | 7 | No | 5 | No |
| C03_S05:129 | 12 | 3 | No | **0** | **Yes** | **0** | **Yes** |
| C03_S06:16 | 10 | 10 | No | 10 | No | 10 | No |
| C03_S06:36 | 14 | 14 | No | 14 | No | 14 | No |
| C03_S06:56 | 4 | 4 | No | 4 | No | 4 | No |
| C03_S06:68 | 11 | 11 | No | 11 | No | 11 | No |
| C04_S01:10 | 3 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C04_S01:15 | 7 | 2 | No | 2 | No | **0** | **Yes** |
| C04_S01:28 | 5 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S01:35 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S01:40 | 14 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S01:56 | 21 | **0** | **Yes** | 18 | No | **0** | **Yes** |
| C04_S01:79 | 7 | 7 | No | 5 | No | 5 | No |
| C04_S01:97 | 4 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S01:103 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S01:118 | 19 | 12 | No | 18 | No | **0** | **Yes** |
| C04_S01:142 | 6 | 4 | No | 4 | No | 4 | No |
| C04_S01:142 | 6 | 4 | No | 4 | No | 4 | No |
| C04_S02:16 | 8 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:26 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:31 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:35 | 8 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:45 | 2 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C04_S02:49 | 1 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C04_S02:52 | 1 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:55 | 4 | 2 | No | 2 | No | **0** | **Yes** |

*Continued on next page*

Table 2 – *Continued from previous page*

| Theorem | # Tactics | AESOP | | SUGGEST_TACTICS | | SEARCH_PROOF | |
|---|---|---|---|---|---|---|---|
| | | # Human tactics | Auto | # Human tactics | Auto | # Human tactics | Auto |
| C04_S02:61 | 7 | 2 | No | 2 | No | **0** | **Yes** |
| C04_S02:70 | 7 | 1 | No | 5 | No | **0** | **Yes** |
| C04_S02:84 | 5 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:91 | 2 | 1 | No | 2 | No | 1 | No |
| C04_S02:95 | 2 | 1 | No | 2 | No | 1 | No |
| C04_S02:99 | 4 | 1 | No | 2 | No | **0** | **Yes** |
| C04_S02:107 | 6 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:115 | 4 | 1 | No | 1 | No | **0** | **Yes** |
| C04_S02:121 | 11 | 8 | No | 8 | No | 6 | No |
| C04_S02:134 | 2 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C04_S02:138 | 2 | **0** | **Yes** | 1 | No | 1 | No |
| C04_S02:148 | 3 | 1 | No | 3 | No | **0** | **Yes** |
| C04_S02:157 | 4 | 1 | No | 2 | No | **0** | **Yes** |
| C04_S02:167 | 10 | 5 | No | 4 | No | 4 | No |
| C04_S02:179 | 8 | 7 | No | 8 | No | **0** | **Yes** |
| C04_S02:209 | 7 | 6 | No | 6 | No | 2 | No |
| C04_S02:221 | 7 | 4 | No | 3 | No | 3 | No |
| C04_S02:239 | 10 | 8 | No | 9 | No | 8 | No |
| C04_S03:25 | 10 | 10 | No | 10 | No | 10 | No |
| C05_S01:9 | 22 | 16 | No | 19 | No | 16 | No |
| C05_S01:34 | 26 | 17 | No | 25 | No | 17 | No |
| C05_S01:76 | 10 | 9 | No | 8 | No | 6 | No |
| C05_S01:89 | 12 | 12 | No | 12 | No | 12 | No |
| C05_S02:8 | 9 | 7 | No | 7 | No | 7 | No |
| C05_S02:26 | 6 | 6 | No | 5 | No | 5 | No |
| C05_S02:67 | 4 | 3 | No | 1 | No | **0** | **Yes** |
| C05_S02:73 | 3 | 3 | No | 3 | No | **0** | **Yes** |
| C05_S02:78 | 4 | 1 | No | 1 | No | **0** | **Yes** |
| C05_S02:84 | 4 | 3 | No | 3 | No | 3 | No |
| C05_S03:33 | 19 | 16 | No | 18 | No | 15 | No |
| C05_S03:59 | 3 | **0** | **Yes** | **0** | **Yes** | **0** | **Yes** |
| C05_S03:69 | 3 | **0** | **Yes** | 1 | No | **0** | **Yes** |
| C05_S03:81 | 3 | 1 | No | 2 | No | 1 | No |
| C05_S03:88 | 11 | 8 | No | 8 | No | 8 | No |
| C05_S03:102 | 26 | 25 | No | 22 | No | 22 | No |
| C05_S03:160 | 3 | 3 | No | 3 | No | **0** | **Yes** |
| C05_S03:165 | 27 | 27 | No | 27 | No | 26 | No |
| C06_S01:19 | 1 | 1 | No | 1 | No | **0** | **Yes** |
| C06_S01:25 | 1 | 1 | No | 1 | No | 1 | No |
| C06_S03:196 | 2 | 2 | No | **0** | **Yes** | **0** | **Yes** |

*Continued on next page*

Table 2 – *Continued from previous page*

| Theorem | # Tactics | AESOP | | SUGGEST_TACTICS | | SEARCH_PROOF | |
|---|---|---|---|---|---|---|---|
| | | # Human tactics | Auto | # Human tactics | Auto | # Human tactics | Auto |
| C06_S03:200 | 2 | 1 | No | 1 | No | 1 | No |
| C06_S03:204 | 2 | 1 | No | 1 | No | 1 | No |
| C06_S03:208 | 2 | 2 | No | 1 | No | **0** | **Yes** |
| C07_S01:181 | 1 | 1 | No | 1 | No | **0** | **Yes** |
| C07_S01:185 | 1 | 1 | No | 1 | No | 1 | No |
| C07_S01:189 | 1 | 1 | No | 1 | No | 1 | No |
| C07_S01:206 | 3 | 3 | No | 3 | No | 3 | No |
| C07_S03:94 | 3 | 3 | No | 3 | No | 3 | No |
| C07_S03:110 | 10 | 10 | No | 10 | No | 8 | No |
| C08_S01:8 | 13 | 4 | No | 6 | No | 3 | No |
| C08_S01:32 | 3 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C08_S01:37 | 3 | 1 | No | **0** | **Yes** | **0** | **Yes** |
| C08_S02:33 | 1 | 1 | No | 1 | No | 1 | No |
| C09_S01:63 | 3 | 3 | No | 3 | No | 1 | No |
| C09_S02:92 | 7 | 6 | No | 6 | No | **0** | **Yes** |
| C09_S03:55 | 4 | 3 | No | 2 | No | **0** | **Yes** |
| C09_S03:112 | 1 | 1 | No | 1 | No | 1 | No |
| C09_S03:118 | 19 | 19 | No | 19 | No | 18 | No |

## Appendix D. Impacts & Future Works

In this paper, we have presented Lean Copilot, a general open-source framework for running LLM inference in Lean. This solves a historical challenge in Lean of using the power of *neural* networks to aid development in the *symbolic* system of the Lean proof assistant. Therefore, Lean Copilot opens up a wide range of opportunities for neuro-symbolic reasoning in Lean. In this section, we highlight some immediate impacts Lean Copilot may have in the field, in the near future.

1. **LLM-Powered Proof Automation for Lean Users**: Lean users can now use our proof automation tools easily in Lean, to get tactic suggestions, proof completions, and premise selections, which accelerate the theorem proving process in Lean. We are glad to have already heard many such reports from the Lean community, and greatly appreciate people's feedback.

2. **Building More Proof Automation**: In this work, we build proof automation tools for three highly important applications, yet there are certainly more such tools that can be built to aid theorem proving in Lean. Example include autoformalization, proof repair, LLM-guided proof search algorithms, and more. We are happy to have already seen such efforts being proposed and added to Lean Copilot, including proof progress prediction (Huang et al., 2025b) and lifelong learning (Kumarappan et al., 2024).

3. **More Immediate Synchronization within the Field**: With Lean Copilot being a general framework that supports user-brought models, once a new state-of-the-art machine learning model appears on the *neural* side of the field, it can be immediately added via Lean Copilot to

the *symbolic* system of Lean. On the other hand, Lean Copilot can also help AI developers test their models in the real Lean enviroment. In other words, Lean Copilot as a neuro-symbolic framework can serve as a bridge between the *neural* and the *symbolic* sides of the field, enabling more effective synchronization.

4. **Dynamic Copilots**: While Lean Copilot supports efficient inference of LLMs, there are times when (some light) learning is desired. Especially in fast-developing formalization projects, new premises can appear frequently. If the knowledge base of the LLM behind the proof automation tools is static, it would require the LLM to be updated once in a while. Otherwise it soon becomes less helpful. There can be many ways to enable LLMs to learn dynamically from a growing Lean project, such as in-context learning (Dong et al., 2024; Min et al., 2022) (which then requires satisfactory long-context abilities) or efficient ML (Zhao et al., 2024; Gretsch et al., 2024) (so that not only inference but also training can happen natively in Lean). Advances in those approaches can help make Lean Copilot dynamic and consistently helpful.