Towards Large Language Models as Copilots for Theorem Proving in Lean

Peiyang Song[*2], **Kaiyu Yang**[1], **Anima Anandkumar**[1]
[1]Caltech, [2]UC Santa Barbara
p_song@ucsb.edu, {kaiyuy, anima}@caltech.edu

Abstract

Theorem proving is an important challenge for large language models (LLMs), as formal proofs can be checked rigorously by proof assistants such as Lean, leaving no room for hallucination. Existing LLM-based provers try to prove theorems in a fully autonomous mode without human intervention. In this mode, they struggle with novel and challenging theorems, for which human insights may be critical. In this paper, we explore LLMs as copilots that assist humans in proving theorems. We introduce Lean Copilot, a framework for running neural network inference in Lean. It enables programmers to build various LLM-based proof automation tools that integrate seamlessly into the workflow of Lean users. Using Lean Copilot, we build tools for suggesting proof steps and completing intermediate proof goals using LLMs. Experimental results demonstrate the effectiveness of our method in assisting humans compared to existing rule-based proof automation in Lean.

1 Introduction

Proof assistants, also known as interactive theorem provers [1–3], are software for mathematicians[2] to write formal proofs that can be checked rigorously by computers [4, 5]. Recently, there has been increasing interest in using them with machine learning, especially large language models (LLMs), to prove theorems automatically [6, 7]. It serves as a rigorous and challenging task for AI to master advanced mathematics, as the task requires generating formal proofs whose correctness can be verified. Furthermore, LLMs can make proof assistants easier to use by improving automation.

Existing LLM-based provers aim to prove theorems fully autonomously without human intervention [7–15]. They wrap the proof assistant into a gym-like [16] environment. The model interacts with the proof environment and is evaluated by the number of test theorems it can prove. The interaction happens on the backend server, without any human intervention. While an autonomous AI mathematician is desirable in the long term, current LLMs often fail to prove theorems that are truly novel or challenging, especially when they come from a different domain than the training data [17].

Instead of proving theorems by itself, AI can also assist human mathematicians in theorem proving. Humans can use mathematical intuition and knowledge to provide guidance or critical proof steps, whereas LLMs can automate parts of the proof that are more straightforward and tedious for humans. This approach is highly viable since it incrementally automates the current workflow of proof assistants. Furthermore, it may accelerate the research toward the long-term vision of AI mathematicians. Having LLMs as copilots makes proof assistants easier to use for humans. Therefore, it will improve the quality and coverage of formalized mathematics, which will provide more and better data for developing LLMs for theorem proving.

[*]Research conducted while Peiyang Song was an intern at Caltech.
[2]We talk about mathematics in writing, but our work also applies to formal verification.

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.

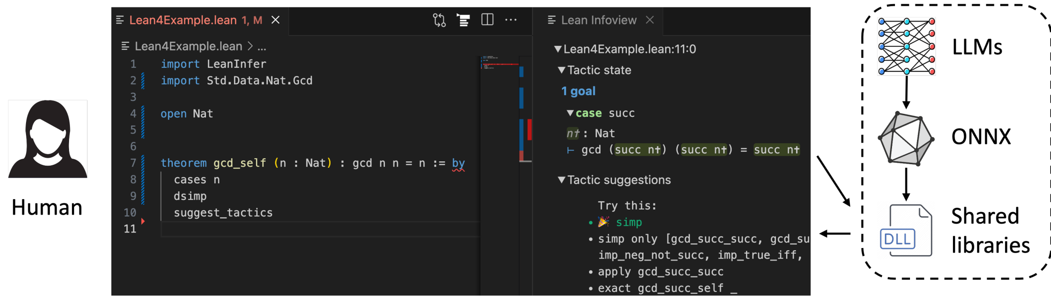

Figure 1: Large language models (LLMs) can assist humans in proving theorems. To prove the theorem `gcd_self` in Lean, the user enters two tactics manually (`cases n` and `dsimp`) and then calls `suggest_tactics`, which uses an LLM to generate four tactic suggestions, displayed in the infoview panel (*Right*). The LLM-generated tactic suggestion `simp` successfully proves the theorem.

Despite significant progress and open-source projects such as LeanDojo [15], existing LLM-based provers cannot readily assist humans. They are trained and evaluated following standard practices in machine learning but do not integrate into the workflow of proof assistants.

Lean Copilot. We focus on Lean,[3] a proof assistant popular among mathematicians [3]. As shown in Fig. 1, a proof in Lean consists of a sequence of proof steps called *tactics* (e.g., `cases n`, `dsimp`, and `suggest_tactics`). Starting from the theorem as the initial goal, tactics repeatedly transform the current goal into simpler sub-goals, until all goals are solved. Users write tactics interactively in an IDE powered by VS Code, which displays the goals in the infoview panel on the right. Proof automation tools in Lean can also take the form of tactics (e.g., `suggest_tactics` in Fig. 1). Like regular tactics, they can manipulate the proof goals and display information in the infoview panel.

We introduce *Lean Copilot*: a framework for developing LLM-based proof automation in Lean. It addresses a core technical challenge: *running neural network inference in Lean*. Fig. 1 illustrates how Lean Copilot works. It builds on LeanDojo's open-source LLM for tactic generation [15]. First, we convert the model into a platform-independent format, ONNX (Open Neural Network Exchange) [18]. Then, we run it as a shared library through Lean's foreign function interface (FFI). Users can install Lean Copilot easily as a Lean package. It works out of the box, without changing Lean's existing workflow or requiring additional setup steps in Python. Furthermore, the model is small and efficient enough to run on most hardware, including laptops without GPUs.

Lean Copilot enables programmers to build various LLM-based tools that work seamlessly in Lean. We use it to build two tools for assisting humans in theorem proving: (1) `suggest_tactics` (Fig. 1): a tactic that uses LLMs to suggest proof steps and (2) LLM-aesop: a proof search tactic that combines LLM-generated proof steps with `aesop` (Lean's existing rule-based proof search) [19]. We evaluate LLM-aesop on selected exercises from the "Mathematics in Lean" book [20]. LLM-aesop alone (without humans) can prove more theorems than the original `aesop`. Furthermore, it can better assist humans than `aesop`, requiring fewer tactics to be entered manually by humans.

In summary, we introduce a general framework, Lean Copilot, and two tools (`suggest_tactics` and LLM-aesop) for LLMs to assist humans in proving Lean theorems. They can be installed easily and run on most hardware, which paves the way for wide adoption in the Lean community. 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 mathematics. Our code will be released upon the publication of this paper.

2 Lean Copilot for Native Neural Network Inference in Lean

Besides an interactive theorem prover, Lean is also a general-purpose programming language [21]. For LLMs to assist humans in Lean, Lean Copilot provides a general framework for *running the inference of LLMs in Lean*, and Lean programmers can use it to build various LLM-based applications.

Since all popular deep learning frameworks are in Python, a natural solution would be hosting the model in Python (locally or remotely) and making requests to it from Lean [22]. However, this

[3] "Lean" in this paper refers to Lean 4, the latest major version of Lean.

approach suffers from the overhead of inter-process communication, and it requires users to perform additional setup steps that do not fit into Lean's conventional workflow. To overcome these issues, Lean Copilot runs LLMs natively in Lean through its foreign function interface (FFI).

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

We adopt the ReProver model from LeanDojo [15].[4] It is based on an encoder-decoder Transformer, ByT5 [23], that maps an input string to an output string. Lean Copilot makes the model runnable in Lean by wrapping it into a C++ function operating on strings, which can be called in Lean through FFI. As Fig. 1 shows, we convert the model into the ONNX format [18] and run it using ONNX Runtime [24]. We use multinomial sampling for decoding the output sequence, with hyperparameters such as temperature and the number of desired output sequences. We do not perform tokenization since ByT5 is a tokenizer-free model that works directly on UTF-8 bytes.

Future Extensions. Lean Copilot's FFI-based method is highly extendable. For example, to use a different model, one can convert the model into ONNX and set up its tokenizer. To further improve efficiency, one can optimize the ONNX model (e.g., using TVM [25]) or switch to a different runtime library such as llama.cpp [26] or GPT4All [27].

3 Building LLM-based Proof Automation with Lean Copilot

Lean Copilot provides a general mechanism for running LLM inference in Lean, which is useful for building various applications for proof automation and beyond. Next, we showcase two important use cases for LLMs to assist humans in theorem proving: tactic suggestion and proof search.

3.1 Generating Tactic Suggestions

When humans prove theorems in Lean, they inspect the current goal to decide the next tactic. Tactics do not come from a predefined list; they are more like programs in a domain-specific language (DSL). They can take arbitrary Lean terms as parameters, and simpler tactics can be combined into compound tactics. Furthermore, users can extend existing tactics by defining customized tactics. Due to these complexities, producing the right tactic can be challenging even for experienced Lean users.

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 goal into an LLM and displays the generated tactics in the infoview panel (Fig. 1 *Right*). It only displays information, without making any modifications to the goal. The user can choose whether to accept one of the suggestions (by clicking on it) or use them as inspirations to come up with a new tactic. Our frontend for displaying tactics is based on an existing tactic suggestion tool, `llmstep` [22]. If a suggestion can directly solve the current goal, it is marked by a party popper emoji (e.g., the `simp` tactic in Fig. 1).

The LLM tactic generator underlying `suggest_tactics` is the ReProver model open-sourced by LeanDojo [15]. It is relatively small (299M parameters) and can run efficiently on most hardware on which Lean runs, including laptops without GPUs. This is important for wide adoption among Lean users, as they may not have access to CUDA-enabled GPUs when writing proofs.

3.2 Searching for Proofs with LLM-aesop

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 different alternatives—a process called *proof search*. `Suggest_tactics` only generates tactics for the current step, without the capability to search for multi-tactic proofs. However, we combine it with `aesop` [19] to build an LLM-based proof search tool named LLM-aesop.

`Aesop` implements best-first search as a Lean tactic and allows users to configure how the search tree gets expanded. The search tree consists of 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

[4]For simplicity, we use the version of ReProver without retrieval.

applying tactics, and adds the resulting nodes as its children. The proof search succeeds when `aesop` has found a path from the root to 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. Therefore, `aesop`'s performance depends critically on whether the user has configured an effective rule set, which is often problem-dependent. `Aesop` lacks the flexibility to adaptively decide what tactics to try during proof search.

LLM-aesop augments `aesop`'s rule set with goal-dependent tactics generated by `suggest_tactics`. It allows the rule set to be customized for each goal, which makes `aesop` substantially more flexible. Furthermore, LLM-aesop is a drop-in replacement of `aesop`: Users can easily switch between LLM-aesop and the original `aesop` by activating/deactivating the LLM-generated tactics.

4 Experiments

We empirically validate the effectiveness of LLM-aesop compared to `aesop` in two settings: (1) proving theorems autonomously and (2) assisting humans in theorem proving. In addition, we compare LLM-aesop with `suggest_tactics` to demonstrate the benefits of proof search.

Dataset and Experimental Setup. We perform experiments on theorems from "Mathematics in Lean" [20]: a book for beginners to formalize and prove mathematical theorems in Lean. It has 233 theorem proving exercises, covering topics from sets and functions to topology, calculus, and measure theory. For evaluation, we randomly selected 50 theorems, and their proofs have 5.52 tactics on average. The complete list of selected theorems is in Appendix B.

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 an automated tool: LLM-aesop, `aesop`, or `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, and the number is zero if it can prove the original theorem fully autonomously without requiring human-entered tactics.

Results. Table 1 shows the experimental results. LLM-aesop can prove 64% (32 out of 50) theorems autonomously, which is significantly higher than `aesop` and `suggest_tactics`. When used to assist humans, LLM-aesop only requires an average of 1.02 manually-entered tactics, which also compares favorably to `aesop` (3.62) and `suggest_tactics` (2.72).

Table 1: Performance of `suggest_tactics`, `aesop` and LLM-aesop on proving 50 theorems selected from "Mathematics in Lean" [20]. LLM-aesop outperforms both baselines in proving theorems autonomously and in assisting human users, requiring fewer tactics entered by humans. More detailed results can be found in Appendix B.

Method	Avg. # human-entered tactics (↓)	% Theorems proved autonomously (↑)
`aesop`	3.62	12%
`suggest_tactics`	2.72	34%
`search_proofs`	**1.02**	**64%**

5 Conclusion

We have introduced Lean Copilot: a framework for running neural network inference in Lean through FFI. Using Lean Copilot, we have built LLM-based proof automation for generating tactic suggestions (`suggest_tactics`) and searching for proofs (LLM-aesop). Lean Copilot provides an extendable interface between LLMs and Lean. This work has explored how it enables LLMs to assist Lean users. In the future, we hope to see LLM-based proof automation help us formalize mathematics and ultimately enhance LLMs' capability in mathematical reasoning.

Appendix A Related Work

Neural Theorem Proving. Neural networks have been used to prove formal theorems by interacting with proof assistants. For example, they can select premises [28–30] or generate tactics [31–33]. Early works on neural theorem proving often use graph neural networks [6, 34–40], whereas more recent works focus on Transformer-based [41] language models [7–15, 17, 42]. 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 [43], linear arithmetic [44], and commutative rings [45]. 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` [19] 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 instead of generated by machine learning.

Many classical machine learning algorithms have been used for proof automation. Hammers [46–48] outsource the proof goal and selected premises to external automated theorem provers in first-order logic, and they often use machine learning for premise selection. TacticToe [49] and Tactician [50] predict tactics using the k-nearest neighbors algorithm (KNN) with handcrafted features. Piotrowski et al. [51] and Geesing [52] 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 LLMs in proof assistants [22, 53–55]. All of them run the model in Python (locally or remotely) and make requests to it from the proof assistant. In contrast, we run the model natively in Lean (details in Sec. 2).

AI to Assist Mathematical Reasoning. Collins et al. [56] has investigated using LLMs to assist human mathematicians by holding conversations in natural language. To our knowledge, we are the first to investigate this problem in the setting of formal theorem proving.

Appendix B Detailed Experimental Results

Table A shows our complete experimental results on 50 theorems selected from Mathematics in Lean [20]. The aggregated statistics are in Table 1.

Table A: Results on 50 theorems from Mathematics in Lean [20]. 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`		LLM-aesop	
		# Human tactics	Auto	# Human tactics	Auto	# Human tactics	Auto
C02_S01:8	3	3	No	**0**	**Yes**	**0**	**Yes**
C02_S01:38	4	3	No	2	No	**0**	**Yes**
C02_S02:14	1	**0**	**Yes**	1	No	**0**	**Yes**
C02_S02:17	2	**0**	**Yes**	**0**	**Yes**	**0**	**Yes**
C02_S02:24	3	3	No	2	No	**0**	**Yes**
C02_S02:58	3	**0**	**Yes**	**0**	**Yes**	**0**	**Yes**
C02_S03:7	3	3	No	**0**	**Yes**	**0**	**Yes**
C02_S03:31	2	1	No	2	No	1	No
C02_S04:13	5	4	No	3	No	**0**	**Yes**
C02_S04:20	19	9	No	2	No	1	No
C02_S04:109	5	5	No	**0**	**Yes**	**0**	**Yes**
C02_S05:15	19	18	No	**0**	**Yes**	**0**	**Yes**
C02_S05:43	19	**0**	**Yes**	**0**	**Yes**	**0**	**Yes**
C02_S05:108	2	**0**	**Yes**	**0**	**Yes**	**0**	**Yes**
C02_S05:116	3	2	No	**0**	**Yes**	**0**	**Yes**
C02_S05:127	4	4	No	**0**	**Yes**	**0**	**Yes**
C03_S01:36	6	2	No	5	No	1	No
C03_S01:58	4	1	No	3	No	**0**	**Yes**
C03_S01:134	4	1	No	3	No	**0**	**Yes**
C03_S02:28	5	5	No	5	No	4	No
C03_S02:79	4	3	No	**0**	**Yes**	2	No
C03_S03:34	4	4	No	2	No	**0**	**Yes**
C03_S03:56	3	3	No	2	No	1	No
C03_S03:66	3	**0**	**Yes**	**0**	**Yes**	**0**	**Yes**
C03_S03:105	3	1	No	2	No	**0**	**Yes**
C03_S04:7	6	6	No	6	No	6	No
C03_S04:85	7	7	No	7	No	**0**	**Yes**
C03_S05:18	4	3	No	4	No	3	No
C03_S05:76	1	1	No	1	No	**0**	**Yes**
C03_S05:90	9	9	No	7	No	2	No
C03_S05:129	12	7	No	**0**	**Yes**	**0**	**Yes**
C04_S01:11	3	1	No	3	No	**0**	**Yes**
C04_S01:80	9	8	No	8	No	8	No
C04_S01:121	19	11	No	19	No	**0**	**Yes**
C04_S01:145	5	4	No	5	No	4	No
C04_S02:26	3	2	No	1	No	**0**	**Yes**
C04_S02:35	8	1	No	**0**	**Yes**	**0**	**Yes**
C04_S02:49	2	1	No	**0**	**Yes**	**0**	**Yes**
C04_S02:61	7	6	No	5	No	**0**	**Yes**
C04_S02:99	4	1	No	2	No	**0**	**Yes**
C04_S02:179	8	7	No	7	No	3	No
C04_S02:221	7	6	No	6	No	2	No
C05_S02:27	6	6	No	5	No	5	No
C05_S03:75	3	1	No	2	No	**0**	**Yes**
C05_S03:82	3	1	No	1	No	1	No
C06_S01:19	1	1	No	**0**	**Yes**	**0**	**Yes**
C06_S03:191	2	2	No	**0**	**Yes**	**0**	**Yes**
C08_S01:63	3	3	No	3	No	1	No
C08_S02:92	7	6	No	6	No	5	No
C08_S03:55	4	3	No	4	No	2	No

References

[1] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq proof assistant reference manual: Version 6.1*. PhD thesis, Inria, 1997. 1

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

[3] 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. 1, 2

[4] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the Kepler conjecture. In *Forum of Mathematics, Pi*, volume 5, 2017. 1

[5] 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/`. 1

[6] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, 2019. 1, 5

[7] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020. 1, 5

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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. 1, 2, 3, 5

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016. 1

[17] 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. 1, 5

[18] Junjie Bai, Fang Lu, Ke Zhang, et al. ONNX: Open neural network exchange. `https://github.com/onnx/onnx`, 2023. 2, 3

[19] 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. 2, 3, 5

[20] Jeremy Avigad and Patrick Massot. Mathematics in Lean, 2020. 2, 4, 5, 6

[21] David Thrane Christiansen. Functional programming in Lean, 2023. 2

[22] Sean Welleck and Rahul Saha. llmstep: LLM proofstep suggestions in Lean. `https://github.com/wellecks/llmstep`, 2023. 2, 3, 5

[23] 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. 3

[24] Microsoft. ONNX Runtime: cross-platform, high performance ml inferencing and training accelerator. `https://github.com/microsoft/onnxruntime`, 2023. 3

[25] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. TVM: An automated end-to-end optimizing compiler for deep learning. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2018. 3

[26] GPT4All: open-source llm chatbots that you can run anywhere. `https://github.com/nomic-ai/gpt4all`, 2023. 3

[27] llama.cpp: Port of facebook's LLaMA model in C/C++. `https://github.com/ggerganov/llama.cpp`, 2023. 3

[28] 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. 5

[29] 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.

[30] 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. 5

[31] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A learning environment for theorem proving. In *International Conference on Learning Representations (ICLR)*, 2019. 5

[32] 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.

[33] 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. 5

[34] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning (ICML)*, 2019. 5

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

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

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

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

[39] 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.

[40] 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. 5

[41] 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. 5

[42] 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. 5

[43] 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. 5

[44] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *International Workshop on Types for Proofs and Programs*, 2007. 5

[45] 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. 5

[46] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016. 5

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

[48] Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. *Journal of Automated Reasoning*, 2018. 5

[49] 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. 5

[50] 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. 5

[51] 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. 5

[52] Alistair Geesing. *Premise Selection for Lean 4*. PhD thesis, Universiteit van Amsterdam, 2023. 5

[53] Sean Welleck and Rahul Saha. coq-synthesis: Coq plugin for proof generation and next tactic prediction. `https://github.com/agrarpan/coq-synthesis`, 2023. 5

[54] 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.

[55] Zhangir Azerbayev, Zach Battleman, Scott Morrison, and Wojciech Nawrocki. Sagredo: automated dialogue between GPT and Lean. `https://www.youtube.com/watch?v=CEwRMT0GpKo`, 2023. 5

[56] 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. 5