COMP4801 Final Year Project
Final Report

Empowering Scientific Discoveries with Artificial Intelligence:

A Synergy between Formal Theorem Proving and Large Language Models

Name: Xijia Tao
UID: 3035767762
Supervisor: Dr. Lingpeng Kong
Department of Computer Science, HKU

Abstract

In recent advancements within the realm of deep learning (DL), substantial progress has been witnessed, particularly in the development and deployment of large language models (LLMs). These models have demonstrated exceptional efficacy across a spectrum of linguistic tasks, including translation, summarization, and question-answering [1], showcasing their profound capabilities in language comprehension and generation. However, the application of LLMs in the domain of formal theorem proving (FTP) — a discipline characterized by intricate reasoning and mathematical logic to verify the veracity of statements or to establish proofs — has been relatively underexplored. Despite the traditionally labor-intensive nature of FTP, its advancement holds immense value due to the extensive range of mathematical formalisms it encompasses and its considerable potential for real-world applications [2].

During the 2023-24 academic year, our investigation embarked on an expansive survey of the existing methodologies in this niche, against the backdrop of recent LLM advancements and the integration of newly-contributed datasets. This endeavor led to the development of fine-tuned expert models specifically designed for FTP tasks. Our research rigorously evaluated these models against the miniF2F benchmark, a leading standard in the field, with the intent to address two pivotal research questions: the comparative adeptness of expert versus general models in FTP, and the efficiency of different prompting techniques in enhancing model performance.

As a contribution towards fostering community engagement and application, we seamlessly integrated our expert models into the Lean 3 formal system. This integration is materialized through the development of a VS Code extension, aimed at enhancing the accessibility and utility of our models for FTP purposes.

Acknowledgments

I am immensely grateful to my esteemed supervisor Dr. Lingpeng Kong for his invaluable guidance and support throughout this project. Our collaborative ideation meeting, which included a distinguished PhD student, Mr. Xueliang Zhao, specializing in AI for Mathematics, was instrumental in acquainting us with the latest pertinent research within this domain. The discussions notably enhanced our understanding of the datasets pertinent to fine-tuning our project. His insights into the pivotal challenges in this field have been particularly enlightening. While the project execution was largely independent, the foundation laid during our interactions was a constant source of direction and inspiration.

List of Figures

List of Tables

Abbreviations

- DL: deep learning
- ML: machine learning

- AI: artificial intelligence
- LLM: large language model
- FTP: formal theorem proving
- MoE: mixture of experts
- RL: reinforcement learning
- SFT: supervised fine-tuning
- lr: learning rate
- LoRA: low-rank adaptation
- AGI: artificial general intelligence
- RQ: research question
- SOTA: state of the art
- 7B: 7 billion (of parameters)
- RAG: retrieval-augmented generation

Table of Contents

1 Introduction

In this section, we introduce the background of FTP and then discuss some earlier work by the ML community in the field. Next, we provide a brief overview of the recent progress in LLMs. Then, our motivation behind this FYP is elaborated. Finally, an outline of this report is included for reference.

1.1 Background

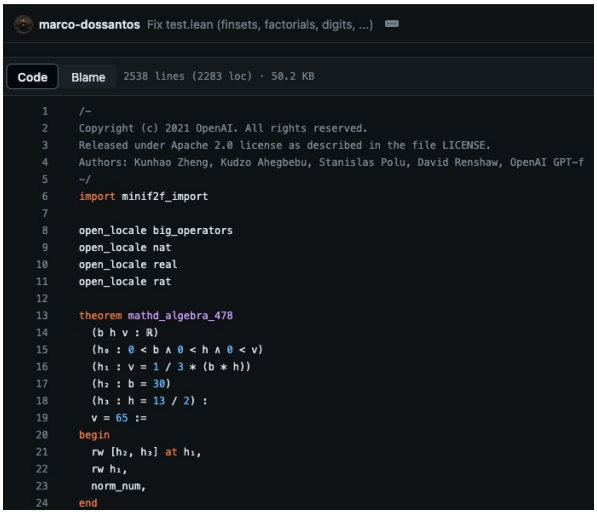
1.1.1 Formal Theorem Proving

In the past decade, many mathematicians have established the correctness of their proofs for theorems in a formal system. This not only automatically checks for any error in a written proof, but also allows others to trust in a proof's conclusion without going through it step by step. Such systems are also employed in other fields, such as software verification [2]. Currently, FTP is mainly performed using specialized proof assistants like Lean, Coq, Isabelle, and HOL Light [9]. These systems provide a framework for mechanically checking proofs, but they often require users to write detailed and explicit formal proofs, which can be laborious and error-prone. In addition, the learning curve associated with these proof assistants can be steep, posing a barrier to newcomers in the field.

The miniF2F benchmark [10] comprises a few hundred problem statements sourced from high-school-level mathematics competitions and the International Mathematical Olympiad, as well as material from high school and undergraduate mathematics courses. It is considered one of the most significant benchmarks in the field of theorem proving. Two example test data points written in the Lean formal language are illustrated in Figure 1.

1.1.2 Large Language Models

LLMs' development was largely motivated by OpenAI when they first invented ChatGPT. Now, many organizations have trained and open-sourced their own LLMs. Among them, Llama and its variants are one of the most popular choices for developers and users. Among the variants, Code Llamma is the official model that continues training



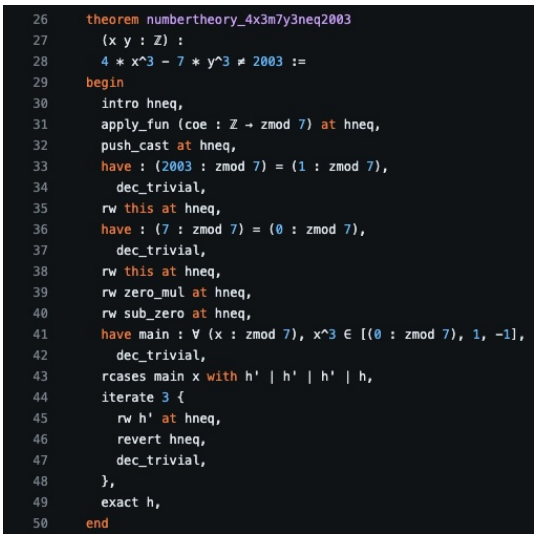


Figure 1: Two example theorems and their corresponding proofs from the test split of the miniF2F benchmark, with screenshots taken from their GitHub repository.

from the Llama checkpoint using program-related datasets. This training pipeline is illustrated in Figure 2.

Llamas are decoder-only transformers [11] [12]. Although it is not yet clear if ChatGPT and GPT-4 use the vanilla decoder-only architecture, many LLMs, including those fine-tuned from Llamas, are based on the same general architecture as previous GPTs (e.g., GPT-2 and GPT-3). More recently, Mixture of Experts (MoE) models have gained widespread attention. They are represented by Mixtral 7X8B, with a performance comparable or better than Llama 2 70B and ChatGPT on various benchmarks [13]. For such architecture, the number of parameters activated during inference is much less than the total number of parameters. It allows different experts to be employed for different inputs, thus maintaining comparable performance to models of large size. However, there remain subtleties in fine-tuning for such MoE models due to their special architectures.

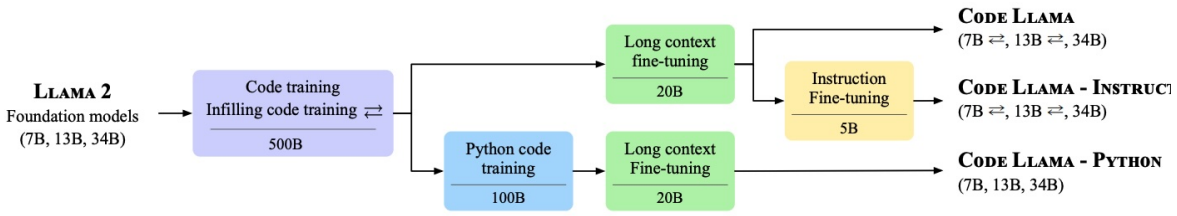


Figure 2: The Code Llama specialization pipeline, adapted from the paper [3]. The number of training tokens is labeled below each fine-tuning stage. The ⇄ symbol marks models with infilling capability.

Model	Size	HumanEval			MBPP		
		pass@1	pass@10	pass@100	pass@1	pass@10	pass@100
code-cushman-001	12B	33.5%	-	-	45.9%	-	-
GPT-3.5 (ChatGPT)	-	48.1%	-	-	52.2%	-	-
GPT-4	-	**67.0%**	-	-	-	-	-
PaLM	540B	26.2%	-	-	36.8%	-	-
PaLM-Coder	540B	35.9%	-	88.4%	47.0%	-	-
PaLM 2-S	-	37.6%	-	88.4%	50.0%	-	-
StarCoder Base	15.5B	30.4%	-	-	49.0%	-	-
StarCoder Python	15.5B	33.6%	-	-	52.7%	-	-
StarCoder Prompted	15.5B	40.8%	-	-	49.5%	-	-
Llama 2	7B	12.2%	25.2%	44.4%	20.8%	41.8%	65.5%
	13B	20.1%	34.8%	61.2%	27.6%	48.1%	69.5%
	34B	22.6%	47.0%	79.5%	33.8%	56.9%	77.6%
	70B	30.5%	59.4%	87.0%	45.4%	66.2%	83.1%
Code Llama	7B	33.5%	59.6%	85.9%	41.4%	66.7%	82.5%
	13B	36.0%	69.4%	89.8%	47.0%	71.7%	87.1%
	34B	48.8%	76.8%	93.0%	55.0%	76.2%	86.6%
Code Llama - Instruct	7B	34.8%	64.3%	88.1%	44.4%	65.4%	76.8%
	13B	42.7%	71.6%	91.6%	49.4%	71.2%	84.1%
	34B	41.5%	77.2%	93.5%	57.0%	74.6%	85.4%
Unnatural Code Llama	34B	**62.2%**	**85.2%**	**95.4%**	**61.2%**	**76.6%**	86.7%
Code Llama - Python	7B	38.4%	70.3%	90.6%	47.6%	70.3%	84.8%
	13B	43.3%	77.4%	94.1%	49.0%	74.0%	87.6%
	34B	53.7%	82.8%	94.7%	56.2%	76.4%	**88.2%**

Figure 3: Code Llama pass@ scores on HumanEval and MBPP, which are highly recognized benchmarks for code, adapted from the paper [3].

Model	Modality	MMLU	HellaSwag	WinoG	PIQA	Arc-e	Arc-c	NQ	TriviaQA	HumanEval	MBPP	MATH	GSM8K
LLaMA 2 7B	Pretrained	44.4%	77.1%	69.5%	77.9%	68.7%	43.2%	24.7%	63.8%	11.6%	26.1%	3.9%	16.0%
LLaMA 2 13B	Pretrained	55.6%	**80.7%**	72.9%	80.8%	75.2%	48.8%	**29.0%**	**69.6%**	18.9%	35.4%	6.0%	34.3%
Code-Llama 7B	Finetuned	36.9%	62.9%	62.3%	72.8%	59.4%	34.5%	11.0%	34.9%	**31.1%**	**52.5%**	5.2%	20.8%
Mistral 7B	Pretrained	**60.1%**	**81.3%**	**75.3%**	**83.0%**	**80.0%**	**55.5%**	**28.8%**	**69.9%**	**30.5%**	47.5%	**13.1%**	**52.2%**

Table 2: Comparison of Mistral 7B with Llama. Mistral 7B outperforms Llama 2 13B on all metrics, and approaches the code performance of Code-Llama 7B without sacrificing performance on non-code benchmarks.

Figure 4: Comparison of Mistral 7B with Llama on various metrics, adapted from the paper [4]. Without losing accuracy on non-code benchmarks, Mistral 7B achieved comparable performance to Code Llama 7B on code-related benchmarks.

1.2 Motivation

Our motivation for investigating the use of LLMs for FTP is primarily the same as all the prior work in this field. On the one hand, LLMs currently lack sufficient reasoning ability, which many believe is the key to artificial general intelligence (AGI). An improved performance on FTP benchmarks implies an improved reasoning ability, which can potentially be extended to tasks other than theorem proving. It thus can be considered as a step towards AGI. On the other hand, FTP demands math expertise and is time-consuming for both human experts and traditional search-based methods. LLMs can arise as a tool for mathematicians formulating their proofs. With automated proof generation, it is even possible to prove unproven conjectures. Among all the unproven conjectures, those which demand intensive calculations are the most suitable ones for computers. An example is the four-color theorem, which was proven with the aid of the Coq formal system.

The combination of LLMs and formal theorem provers holds great potential for advancing scientific discovery. However, current approaches for this problem have shown limited performance on relevant benchmarks. For the miniF2F benchmark, SOTA methods have achieved only around 50% accuracy under many attempts per test theorem (e.g., 100 attempts), indicating ample room for improvement. With stronger LLMs available

every month, we see a great opportunity for standing on the shoulders of giants and introducing novel methods to improve SOTA.

1.3 Outline

In the following sections of the report, the objectives of our study are first outlined (Sec 2). Then, a review of recent and more classical ML approaches towards FTP is introduced (Sec 3). Next, we describe the methodology, from experiment setup to training and evaluation methods (Sec 4). Subsequently, we showcase and discuss the results with respect to the two research questions proposed (Sec 5, 6). Finally, we highlight the contribution of our study and conclude by addressing its limitations (Sec 7, 8, 9).

2 Objectives

The main objective of this project is to improve the performance of current LLMs in the task of theorem proving. We took a direct fine-tuning approach with carefully selected datasets, resulting in expert models that can be used out of the box. We have open-sourced all information relevant to our experiments - experiment codes, fine-tuned models, a VS Code extension that is useful when writing proof in the editor, etc., according to appropriate licenses.

2.1 RQ1 - Expert vs. Generalist Models

Which one is more suitable for FTP - generically capable models, exemplified by GPT-4, or smaller expert models, represented by Code-Llama fine-tuned on task data?

As shown in Figure 3, on the HumanEval benchmark [14], Code-Llama 34B achieves a 48.8% pass rate, outperforming ChatGPT by a small margin [3]. GPT-4 achieves 67% on the same benchmark, which is significantly higher. However, when sampling Code-Llama's response 100 times, it can reach a remarkable 93% pass rate. Therefore, we argue that Code-Llama possesses abundant knowledge in coding and a strong reasoning ability. FTP can also be viewed as a coding task. Therefore, it is sensible to compare Code-Llama and GPT-4's performance on FTP benchmarks. Extrapolating from their performance on HumanEval, we hypothesize that GPT-4 is better at FTP. However, after fine-tuning on domain-related datasets, Code-Llama might in turn outperform GPT-4.

On the one hand, for 7B LLMs, both Code Llama and Mistral achieved around 30% pass rate, as indicated in Figure 4. This implies that they are comparable in coding and reasoning relevant to the FTP task. Hence, we selected these two LLMs in order to examine which LLM performed better after fine-tuning. This could be an indicator of other relevant abilities not demonstrated in existing benchmarks. These fine-tuned 7B models are referred to as experts. On the other hand, GPT-3.5 and GPT-4 appearing in our experiments are referred to as generalists.

2.2 RQ2 - Impact of Prompting Methods

How are the popular prompting methods performing in the theorem-proving scenario? / How to elicit the full reasoning power of frozen LLMs when conducting FTP?

It is computationally expensive to train an LLM with billions of parameters. Hence, it is ideal to intervene in the inference process instead of training to improve FTP performance.

At inference time, there are many recently proposed methods to augment the input for maximizing the accuracy of the model's answers. In our experiments, we focused on four such techniques, namely Self-Refine, Reflexion, Corex and RAG, apart from standard prompting which is used as default. Each of the four techniques targets different abilities of LLMs. For instance, Reflexion adds environmental feedback to augment the knowledge of a model, and Self-Refine elicits reasoning and self-correction abilities.

In particular, our initial idea on multi-agent collaboration to improve the model's output accuracy coincides with a recent paper named Corex [7]. In reinforcement learning, a multi-agent framework has been developed for various learning algorithms, such as DDPG and PPO. When viewing LLMs as agents, it is natural to formulate a multi-agent or multi-LLM paradigm for either collaboration or competition. There are some related work last year, e.g., MetaGPT [15] and AgentCoder [16]. For theorem proving, multiple LLMs working together towards solving a single problem might be more effective than a single LLM. Hence, we adapted the Corex method for evaluation in our experiments.

3 Literature Review

In this section, a curated list of recent and classical works on using AI for FTP is discussed.

3.1 Related Work Before ChatGPT-like LLMs

Before the widespread use of foundation models, attempts by AI researchers in the field of FTP have centered around training a specialized model. In this section, we outline some representative work that involves training from scratch.

GPT-f [17] In 2020, GPT-f introduced the use of transformer networks for FTP. It leveraged pre-training and then iterative learning of two objectives that are specifically related to FTP, i.e., proofstep and outcome objective. This marked an important step in the application of DL techniques to automate aspects of mathematical proof.

Proof Artifact Co-Training [18] More recently, in the formal system Lean, extracting data from kernel-level problem statements and theorems derived during proof generation was found to benefit a model's performance. Specifically, it leverages these abundant self-supervised data for training for multiple curated tasks, alongside the proofstep objective proposed in the GPT-f paper. They verified the effectiveness of their method on a held-out suite of test theorems, outperforming the previous methods in the pass rate of generated proofs.

Expert iteration [2] In 2022, researchers proposed to leverage proofs generated by a model itself as training data. By interleaving proof search with learning, the method outperforms proof search only under the same compute budget. As a follow-up work from GPT-f, it proposed to replace the outcome objective with proofsize objective. The later achieves better performance and prefers goals leading to short proofs during proof search. Since the cost of reinforcement learning (RL) is significantly smaller than that of proof search, this approach combines the best of both worlds in a way that heuristics learned from RL and rigorous reasoning performed in search are applied at the same time. Their model, in the same architecture as the previous work GPT-f, can be iteratively improved with proof search trajectories as its training data. Their

method has achieved state-of-the-art (SOTA) on the *miniF2F* benchmark [10] which is widely acknowledged in the field of FTP to this date.

HyperTree Proof Search [19] In the same year as [2], a paper introduced a new search algorithm, inspired by AlphaZero [20]. It is an adaptation based on the Monte Carlo tree search for finding proofs in unbalanced hypergraphs. With a detailed ablation study, the paper suggested that online training works better than the expert iteration approach proposed in the previous work. When generalized from domains far from the training distribution, the method remains performant and outwits the previous SOTA GPT-f in the Metamath formal system. In another system Lean, it has also improved SOTA on the miniF2F benchmark.

3.2 Related Work After ChatGPT

Subgoal-based Demonstration Learning [5] The work proposed to leverage existing LLMs to generate subgoal-based proofs from informal proofs. This is done via iterative interaction with ChatGPT and verification with the Isabelle prover to ensure that the constructed subgoals are easily provable by the LLM. The resulting subgoal-based proofs and their corresponding formal sketches are used as demonstration examples. In addition, the authors point out that the selection and order of in-context examples matter for the performance of demonstration learning. To achieve this, a diffusion model was trained to generate the most effective combination of examples for translating a subgoal-based proof to its corresponding formal sketch. A demonstration of their method is shown in Figure 5.

Lyra [21] Based on LLMs, a new framework employing two correction mechanisms has been proposed. On the one hand, in the post-processing stage, tool correction leverages prior knowledge of predefined prover tools to replace incorrectly used tools. This has

Statement: Suppose n is a natural number such that n leaves a remainder of 3 when divided by 7. Show that (2n+1) leaves a remainder of 0 when divided by 7.

Subgoal-based Proof:
Step 1: Express n as $7k + 3$ for some $k \in \mathbb{N}$
Step 2: Substitute the expression for n into the expression for $2n + 1$ and simplify.
Step 3: Show that the resulting expression is divisible by 7.

Formal Sketch:

```
theorem mathd_numbertheory_370:
  fixes n :: nat
  assumes h0 : "n mod 7 = (3::nat)"
  shows "(2*n+1) mod 7 = (0::nat)"
proof -
  (* Step 1: Express n as 7k + 3 for some k ∈ ℕ. *)
  have "n = 7 * (n div 7) + (n mod 7)"
    using div_mult_mod_eq[of "n" "7"] sledgehammer
  also have "... = 7 * (n div 7) + 3"
    using h0 sledgehammer
  finally have c0: "n = 7 * (n div 7) + 3".
  (* Step 2: Substitute the expression for n into the expression for 2n + 1 and simplify. *)
  have "2*n+1 = 2*(7*(n div 7) + 3) + 1"
    using c0 sledgehammer
  also have "... = 14*(n div 7) + 7"
    sledgehammer
  also have "... = 7*(2*(n div 7) + 1)"
    sledgehammer
  finally have c1: "2*n+1 = 7*(2*(n div 7) + 1)" .
  (* Step 3: Show that the resulting expression is divisible by 7.*)
  show "(2*n+1) mod 7 = 0"
    using c1 sledgehammer
qed
```

Figure 5: A demonstration of various stages involved in subgoal-based demonstration learning, adapted from their paper [5]. To prove the statement, a subgoal-based proof is first generated. Then an LLM is used for translating the natural language proof to a formal sketch, which is a skeleton of a formal proof with holes that can be filled with existing automated prover tools.

been shown to mitigate the hallucination problem. On the other hand, conjecture correction involves interaction between an LLM and a prover to refine proof conjectures with error messages. With the two mechanisms, Lyra has improved the previous SOTA on miniF2F.

ReProver [8] Retrieval-augmented generation has developed as a technique to aid LLMs in integrating solid facts into their response. Recently, researchers open-sourced fine-grained annotation of premises in proofs. This contributes data with strong supervising signals for the task of premise selection, which is a key bottleneck in FTP. With this dataset, a retrieval-augmented prover was proposed. It consists of a retriever and a language model for tactic generation. Instead of larger language models like ChatGPT and LLaMA or models that have undergone domain-specific pretraining like Minerva [22], the researchers chose to continue training from a small (299-600M param-

eters) and generic. This serves as an orthogonal direction to the previously introduced works.

4 Methodology

In this section, we introduce our methodology for improving LLMs for FTP, while investigating RQ1 and RQ2. This includes environment setup, training datasets, fine-tuning settings, and the evaluation pipeline.

4.0.1 Environment Setup

We selected Lean as the formal environment. It is widely used and has an active community contributing proofs to theorems. In addition, its formalization language is high-level with layers of abstraction, also incorporating multiple prover tools. This makes writing proof much easier than writing a proof using other systems like Metamath. The formal environment Lean currently maintains two major versions, numbered 3 and 4. In our experiments, we selected the older Lean 3 because there are more high-quality datasets we found for it in comparison with Lean 4.

4.0.2 Fine-tuning Datasets

For injecting formal math knowledge into Code-Llama, we mainly focused on a dataset contributed recently, named MUSTARDSAUCE [23]. The same paper was published in ICLR 2024 as a spotlight paper. To address a research gap, this study introduces a framework for generating high-quality and diverse theorem and proof data. It consists of three stages: concept sampling, language model prompting, and proof validation.

Using the proposed framework, the researchers created a benchmark dataset, which comprises 5,866 valid data points. Each data point includes an informal statement, an informal proof, and a translated formal proof that has been successfully validated by the proof assistant. Examples of the MUSTARDSAUCE data are shown in Figure 6.

In addition, we identified a dataset contributed in a paper last year [24]. The contributed MMA dataset is a parallel corpus of formal-informal statement pairs. The informal data are diverse and flexible, making it suitable for LLMs to acquire generalization ability to different math domains. With more than 200K data points in two formal languages, Lean and Isabelle, LLMs can learn much about FTP without overfitting. This makes multilingual fine-tuning of LLMs possible. Examples of the MMA data are shown in Figure 7.

4.0.3 Relevant LLMs

As the base models for further training, we selected the base version (i.e., not instruction-tuned for human interaction) of Codellama 7B and Mistral 7B for their outstanding coding and reasoning ability among the models of the same parameter scale.

As baseline methods for comparing with our fine-tuned models, we selected the same two base LLMs, Codellama and Mistral. In addition, we evaluated OpenAI's GPT-3.5 and GPT-4 as stronger baselines than the 7B base models. In the following discussion, we refer to the fine-tuned 7B LLMs as *experts* and the unfine-tuned OpenAI LLMs as *generalists*.

```
{
    "problem_name": "correct_by_msg__MIDD_word_problem_Pre-algebra_974_round1",
    "informal_statement": "Consider two numbers 9 and 2. Calculate the result of the expression 9^2 -
2^3.",
    "informal_proof": "In this problem, we have to follow the order of operations, also known as BIDMAS
or PEMDAS, which stands for Brackets, Indices, Division and Multiplication (from left to right),
Addition and Subtraction (from left to right). In our case, we only have Indices (powers) and
Subtraction. \n\nSo, we first calculate the powers:\n9^2 = 9 * 9 = 81 \n2^3 = 2 * 2 * 2 = 8 \n\nThen we
substitute these results back into the expression to get:\n81 - 8 = 73\n\nTherefore, the result of the
expression 9^2 - 2^3 is 73.",
    "formal_proof": "import data.real.basic\n\n-- proof\nexample : (9:ℝ)^2 - 2^3 = 73 := \nbegin\n
norm_num,\nend",
    "formal_proof_verbose": "# Modified Formal proof in Lean 3: \n\n```lean\nimport
data.real.basic\n\n-- proof\nexample : (9:ℝ)^2 - 2^3 = 73 := \nbegin\n  norm_num,\nend\n```\nThe
original formal proof was more complicated than necessary. We don't need to do any rewriting in this
case; the `norm_num` tactic is able to directly compute the result."
}
```

```
{
    "problem_name": "correct_by_msg__MIDD_theorem_proving_Basic geometry and measurement_544_round1",
    "informal_statement": "Let's consider a rectangular box with dimensions \\(a\\), \\(b\\), and \\
(c\\), where \\(a\\), \\(b\\), and \\(c\\) are positive integers. Given that the total volume of the
box is \\(V = a \\cdot b \\cdot c\\), if we increase the length \\(a\\) by \\(3\\), the width \\(b\\)
by \\(2\\), and the height \\(c\\) by \\(1\\), resulting in a new volume \\(V'\\), express \\(V'\\) as
an algebraic expression.",
    "informal_proof": "The volume of a rectangular box is given by the product of its length, width,
and height. If we increase the length \\(a\\) by \\(3\\), the width \\(b\\) by \\(2\\), and the height
\\(c\\) by \\(1\\), the new dimensions of the box will be \\(a+3\\), \\(b+2\\), and \\(c+1\\).
Therefore, the new volume \\(V'\\) of the box will be \\((a+3) \\cdot (b+2) \\cdot (c+1)\\).",
    "formal_proof": "def V := λ a b c : ℕ, a * b * c\n\ndef V' := λ a b c : ℕ, (a + 3) * (b + 2) * (c +
1)\n\ntheorem new_volume: ∀ a b c : ℕ, V' a b c = (a + 3) * (b + 2) * (c + 1) :=\nbegin\n    intros,\n
   unfold V V',\nend",
    "formal_proof_verbose": "The errors in the provided Lean 3 proof are due to unknown identifiers.
The identifier 'ℝ' for real numbers is not recognized by Lean 3 because it uses 'real' instead.
Moreover, the 'ring' tactic is also not recognized because it is used for simplification of algebraic
expressions involving addition, subtraction and multiplication. However, in this case, the
simplification is not required. Here is the corrected Lean 3 proof:\n\n```lean\ndef V := λ a b c : ℕ, a
* b * c\n\ndef V' := λ a b c : ℕ, (a + 3) * (b + 2) * (c + 1)\n\ntheorem new_volume: ∀ a b c : ℕ, V' a
b c = (a + 3) * (b + 2) * (c + 1) :=\nbegin\n    intros,\n    unfold V V',\nend\n```\n\nThis formal
proof in Lean 3 correctly defines the volume of a rectangular box, and subsequently proves that after
increasing the dimensions of the box, the new volume is equal to the product of the increased
dimensions."
}
```


Figure 6: Two examples from the MUSTARDSAUCE dataset. The first data point takes the form of a math word problem, requiring a model to first identify the answer to the problem and subsequently prove the correctness of the proposed answer. The second data point is in a more usual form of theorem-proving problems, not explicitly requiring the identification of an answer.

```
{
  "input": "Statement in natural language:\nFor a given number 'n', 'n' is less than 'xn' of 'a1' and
'n'.\nTranslate the statement in natural language to Lean:",
  "output": "theorem n_lt_xn (n) : n < xn a1 n :="
}
```

```
{
  "input": "Statement in natural language:\nFor any two elements 'a' and 'b' of a group with zero, if
'a' and 'b' commute, then 'a' to the power of any integer 'm' and 'b' to the power of any integer 'n'
also commute.\nTranslate the statement in natural language to Lean:",
 "output": "theorem Commute.zpow_zpow\u2080 {a b : G\u2080} (h : Commute a b) (m n : \u2124) : Commute
(a ^ m) (b ^ n) :="
}
```


Figure 7: Two example data points from the MMA dataset. `input` indicates a theorem statement in natural language, while `output` indicates the corresponding statement in a formal language, either Lean or Isabelle.

4.0.4 Fine-tuning Settings

Due to the computational constraint imposed by our limited GPU resources, fine-tuning LLMs requires a parameter-efficient approach. Currently, we only had access to RTX 3090 cards with 24GB of GPU memory. Hence, LoRA was necessary to continue training the 7B LLM. We applied low-rank adaptation, or LoRA [6], which is commonly used for efficiently adapting an LLM to a specific domain with a performance comparable to full fine-tuning. The working principle of LoRA is illustrated in Figure 8, adapted a figure from their original paper.

We followed the implementation of LoRA in a Python library called PEFT. For LoRA settings, we set r=8, alpha=16, dropout=0.05. We only targeted the Q and V matrices from the attention layers. Together with training hyperparameters, these configurations are illustrated in Figure 9. To fit the model parameters, gradients and optimizer states in a GPU with small RAM, e.g., 24GB, we pushed the quantization of LLMs to the extreme, with each 32-bit floating-point number converted to a 4-bit integer in model weights. This was achieved using the 4-bit quantization method from the bitsandbytes

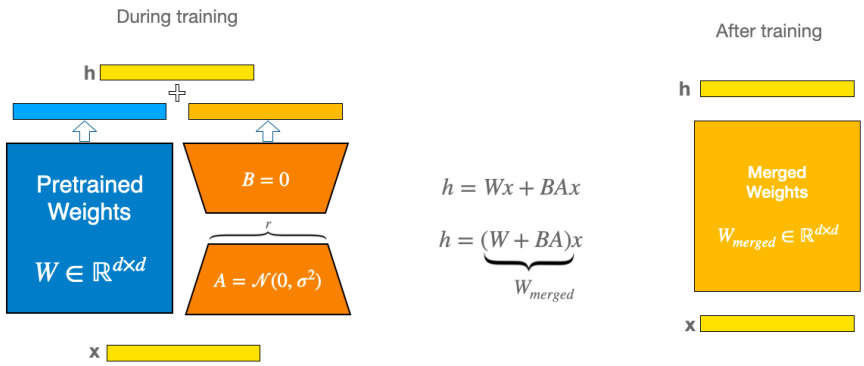


Figure 8: Working principle of LoRA, a figure adapted from the paper [6].

library. To achieve parallelism in multiple GPUs, we employed ZeRO Stage 3 from DeepSpeed. A cosine learning rate (lr) scheduler with a 5e-4 initial lr and a 0.1 ratio of linear warmup was employed. Batch size and gradient accumulation steps were set to 2 and 8, respectively, for each of the 4 GPUs. Code-Llama was trained for 2 epochs with bf16 precision.

The training loss was computed on both the input (an informal statement and instruction to translate) and the output (the formal statement and proof).

```
TrainingArguments(
    per_device_train_batch_size=2,
    num_train_epochs=2,
    learning_rate=5e-4,
    bf16=True,
    lr_scheduler_type='cosine',
    warmup_ratio=0.1,
    dataloader_num_workers=4,
    metric_for_best_model="eval_loss",
    gradient_accumulation_steps=8,
    local_rank=0,
    deepspeed="config/zero3.json",
)
```

```
LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=[
        "q_proj",
        "v_proj",
    ],
    bias="none",
    task_type="CAUSAL_LM",
)
```

Figure 9: Training hyperparameters (left); LoRA configuration (right).

4.1 Evaluation Pipeline

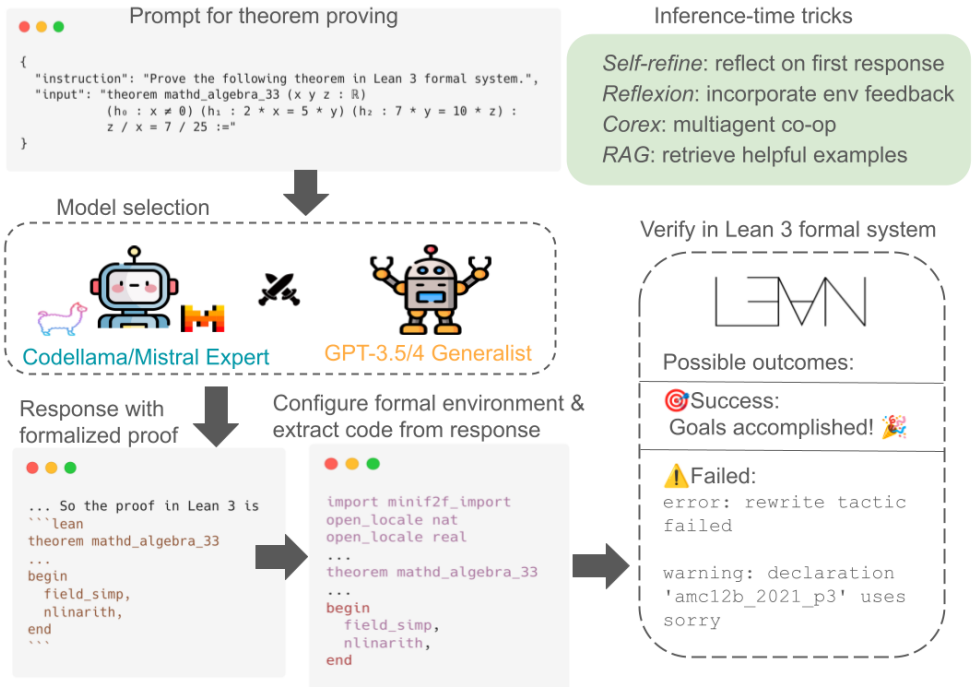


Figure 10: The FTP inference pipeline with LLMs in Lean 3 formal system.

For evaluation, we selected the miniF2F benchmark [10]. miniF2F is a dataset of formal mathematics problems aimed at evaluating neural theorem proving systems. It includes 488 problem statements from various sources, such as Olympiad-level exams and mathematics courses. The benchmark focuses on systems like Metamath, Lean, Isabelle, and HOL Light. Some examples are provided in Figure 1. The fine-tuned models and GPT models were evaluated in a pipeline detailed below and depicted in Figure 10.

1. For each theorem statement in the miniF2F benchmark, an input consisting of an instruction and the theorem statement is passed to an LLM for inference. The LLM could be either base/expert 7B models or OpenAI generalist models.

2. After obtaining the response, a proof is extracted using our self-designed algorithm

detailed in Section 4.2.

3. Based on the required import Lean packages, the Lean environment is properly set up.

4. The proof is then passed to the Lean formal environment for verification. It gives error messages if some part of the proof failed, otherwise the proof is deemed successful.

5. By repeating this process for all the theorems in miniF2F, a pass rate, the percentage of proofs without any error message, i.e., e can be calculated for a given model. In our experiments, we focus on pass@1, where 1 means sampling a model only 1 time for a proof to each theorem.

Orthogonal to this procedure, different prompting methods can be used on the LLMs. The prompting methods are elaborated in Section 4.3.

4.2 Proof Extraction

To extract a formalized proof from a model's response, an algorithm was designed with a theorem and a response as inputs and an extracted proof as the output. The pseudocode of the algorithm is visualized in Figure 11.

Inputs:

1. `theorem_statement`: This is the actual statement of the theorem, written in the Lean3 theorem-proving language syntax, which you are attempting to prove.

2. `text`: This is the natural language response or a mix of natural language and Lean3 code that the model generated in response to the theorem statement you

provided.

Output:

1. `extracted_proof`: This output will be the pure Lean3 code that represents the proof of the theorem, which can be passed to a Lean3 verifier tool to check its validity.

Variables and Flow:

1. `prompt_method`: A string that specifies the method used to enhance or refine the proof extraction process, such as 'self-refine', 'reflexion', etc.
2. `imports`: A string of default Lean3 import commands that should be prepended to the extracted proof, ensuring that all necessary modules are available for the proof verification to work.
3. `match_last`: A Boolean flag indicating whether to match the last proof pattern found in the `text` (`True`) or to use the first one found (`False`).
4. `patterns`: A list of strings where each string is a regex pattern designed to capture different ways the proof might be included in the `text`.

Procedure:

1. The algorithm starts by initializing a list named `proof_candidates`.
2. It then uses regular expressions (`regex.findall`) to search through the 'text' and find all matches for each pattern listed in 'patterns'.

3. If `match_last` is set to True, the order of the list containing the found patterns (`proof_candidates`) is reversed, to prioritize the last match as the desired proof.

4. The pseudocode enters a loop over each `proof` found in `proof_candidates`. This loop will try to find a valid proof according to the given conditions.

5. Inside the loop, it checks if the `theorem_statement` is present in the `proof` candidate. If it isn't, the loop continues to the next candidate.

6. If a valid proof candidate that contains the `theorem_statement` is found, the pseudocode will:

   - Check if the word 'import' is not present. If it's not, it prepends the `imports` to the `proof`.
   - Set `extracted_proof` to this valid `proof`.
   - Break the loop, as a valid proof has been found.

Failsafe: If no valid proof is found (no `proof` containing the `theorem_statement`), the `extracted_proof` is set to None, indicating that the extraction process failed.

4.3 Prompting Methods

To investigate RQ2, we considered and experimented with 5 different prompting methods. These methods in action are illustrated in Figure 12.

- Standard: A prompt with barely any engineering. The theorem statement to be proved is placed before the prompt, while the proof is expected to follow the prompt immediately.

```
Inputs:
    theorem_statement = a formal Lean3 theorem
    text = model's response to a given theorem_statement
Output:
    extracted_proof = Lean3 code to be passed to the verifier

prompt_method = method name {'self-refine'|'reflexion'|...}
imports = default Lean3 imports {'import minif2f ...'}
match_last = True
# curated match patterns in order of priority
patterns = [
    '```lean((?:.|\n)*?)```',
    '```Lean((?:.|\n)*?)```',
    '```\ntheorem((?:.|\n)*?)```'
]
proof_candidates = [regex.findall(text, pattern) for pattern in
patterns]
# match the last matched proof otherwise use the first one
if match_last:
    proof_candidates = [l[::-1] for l in proof_candidates]
for proof in proof_candidates:
    # ensure that the matched proof is for the given theorem
    if theorem_statement in proof:
        extracted_proof = proof
        if 'import' not in proof:
            extracted_proof = imports + proof
        break
else:
    # extraction failed
    extracted_proof = None
```

Figure 11: The algorithm for extracting a proof from a model's response to prove a certain theorem.

- Self-refine [25]: An iterative approach that improves initial outputs from LLMs on top of standard prompting without the need for additional training or supervision. It uses the same LLM as the generator, refiner, and feedback provider.

- Reflexion [26]: A framework for reinforcing language agents, such as LLMs, using linguistic feedback instead of weight updates. The agents reflect on task feedback signals and store reflective text in an episodic memory buffer, improving decision-making in subsequent trials. Reflexion incorporates various types and sources of feedback.

- RAG (Retrieval-Augmented Generation) [27]: A method combines parametric and non-parametric memory to improve language generation. In RAG, the parametric memory is a pre-trained sequence-to-sequence model, while the non-parametric

memory is a dense vector index of a text database, e.g., Wikipedia.

- Corex [7]: Corex is a suite of strategies that transforms LLMs into autonomous agents for complex task-solving. It introduces collaboration paradigms such as Debate, Review, and Retrieve modes to enhance the factuality, faithfulness, and reliability of LLM reasoning. By orchestrating multiple LLMs to work together, the approach promotes collaboration among LLMs, reduces hallucinations, and provides better solutions.

```
Answer this question with a formal proof in Lean:\n\n```lean\n
```

```
Answer this question with a formal proof in Lean:\n\n```lean
<model-response-1>
There is an error in the code above because of lack of understanding of the question. What is the
error? To find the error, go through semantically complete blocks of the code, and check if everything
looks good.
<model-response-2>
Okay! Here is the rewrite:
<model-response-2>
```

```
Answer this question with a formal proof in Lean:\n\n```lean
<model-response-1>
For the previous proof, the tactic state and/or info from the Lean 3 formal system is:
<lean3-env-feedback>
Reflect on the feedback and rewrite the answer in Lean:\n\n```lean
<model-response-2>
```

```
Answer and proof the following question with code in the Lean 3 formal system, given some related
examples within <context> </context>:

<context>
{context}
</context>

Question: ```lean\n{inp-theorem}
```

Figure 12: From top to bottom: standard, self-refine, reflexion and RAG. Except for the RAG method, the input test theorem is prepended before the prompt template shown.

```
init_ftp_system_prompt = {
        "Tom" : "You are Tom, an expert renowned for exceptional math skills and knowledge of formal
theorem proving in Lean 3 formal system. Your friends often approach you for help with formal theorem
proving due to your ability to explain complex concepts in an easy-to-understand manner. Your task here
is to draw upon your deep understanding of mathematical concepts to answer the subsequent question.
Please explain your proof in Lean 3 step by step, demonstrating your thought process clearly.",
        "Jerry" : "You are Jerry, a postgraduate math student renowned for exceptional math skills and
knowledge of formal theorem proving in Lean 3 formal system. Your teachers are always impressed with
your ability in rigorously proving a theorem. Your task here is to draw upon your deep understanding of
mathematical concepts to answer the subsequent question. Please explain your proof in Lean 3 step by
step, demonstrating your thought process clearly.",
}

ftp_debate = {
        "Tom" : """You are engaged in a friendly discussion with your friend Jerry regarding a math
theorem to prove in Lean 3. Jerry shared his solution: {}. As you review it, your task is to carefully
consider Jerry's insights, incorporate any valuable elements into your own analysis, and refine your
proof accordingly. Remember, this is a collaborative endeavor aimed at reaching the best possible proof
together. Please articulate your response thoughtfully.""",
        "Jerry" : """You are in the middle of a friendly discussion with your friend Tom about a
complex math theorem to prove in Lean 3. Tom has just presented his solution: {}. Your role now is to
thoughtfully evaluate Tom's approach, incorporate any beneficial aspects into your own, and adjust your
proof to enhance its quality. Keep in mind, this is a collaborative endeavor where both of you are
working towards reaching the best proof. Please craft your response carefully."""
}
```

Figure 13: The prompt template for the Corex method, followed from the paper [7].

5 Results

The change in training and evaluation loss during training is illustrated in Figure 14.

The SOTA as reported in *paperswithcode* is 35.2% under pass@1

Our experiments assessed the capabilities of various LLMs in formal theorem proving (FTP) using the miniF2F benchmark within the Lean 3 formal system. The performance of base and expert 7B LLMs was compared against generalist models like GPT-3.5 and GPT-4 under different prompting techniques. The expert models were Mistral and Codellama that were fine-tuned with relevant datasets, pivoting them towards domain specialization.

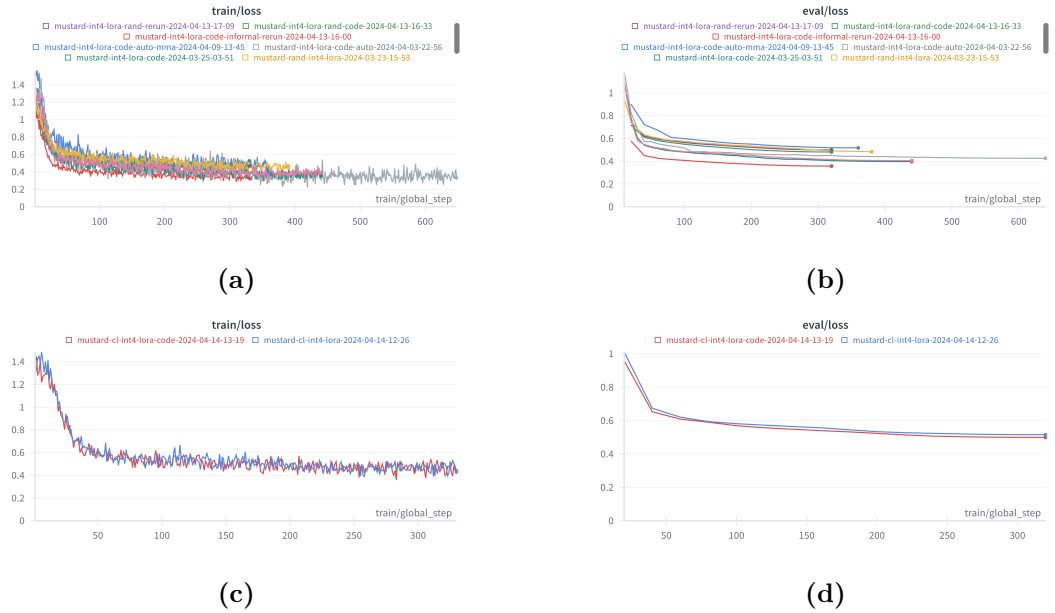


Figure 14: Training dynamics for various experiment settings. 14a: Training loss for various experiment settings with Mistral 7B. 14b: Evaluation loss for various experiment settings with Mistral 7B, evaluated every 10 steps during training on a random left-out split of the train data. 14c: Training loss for various experiment settings with Code Llamma 7B. 14d: Evaluation loss for various experiment settings with Code Llamma 7B, evaluated every 10 steps during training on a random left-out split of the train data.

6 Discussion

According to the results, the insights into the two research questions are introduced in this section.

6.1 RQ1 - Expert vs. Generalist Models

Under the standard prompting method, GPT-4 showcased the highest pass rate at 13.9%, significantly outperforming GPT-3.5's rate of 6.97%. The expert models displayed an improvement when transitioning from base to expert status, with Mistral's

Method/Model	Mistral (Base/Expert)	Codellama (Base/Expert)	GPT-3.5 (Generalist)	GPT-4 (Generalist)
Standard	0/6.15	0/6.97	6.97	13.9
Self-Refine	0/6.15	0/6.15	-	-
Reflexion	0/5.74	0.41/6.15	-	-
Corex	0/0	0/0	-	-
RAG	29.9/29.1	18.4/25.8	7.38	-

Table 1: The table shows the pass rates of 1-pass generated proofs for theorems in the miniF2F benchmark, verified in the Lean 3 formal system. The highest pass rate under standard prompting and the highest pass rate under all settings are highlighted.

pass rate increasing from 0% to 6.15% and Codellama's from 0% to 6.97%. We observed that the base un-finetuned LLMs cannot prove a single theorem in the benchmark correctly. Upon further investigation into their generated proofs, the base models often implicitly reject the answer by including the `sorry` keyword in their proof. In Lean, `sorry` is designed as placeholders to allow users to prove the other parts of the proof while leaving the previous part not yet unproven. Therefore, we argue that the base models know the syntax of Lean (as it uses `sorry`), but to a very limited extent. They cannot pick up the useful tactics (i.e., possible actions at the current progress of a proof) from their knowledge and organize them to achieve all the subgoals required for validating the correctness of a theorem.

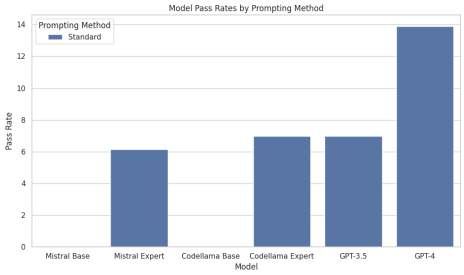
As visualized in Figure 15a, fine-tuning models on domain-specific datasets narrow their gap to larger and generally capable models. With larger high-quality datasets, smaller LLMs can have the potential to match GPT-4-like models in specific domains like FTP.

6.2 RQ2 - Impact of Prompting Methods

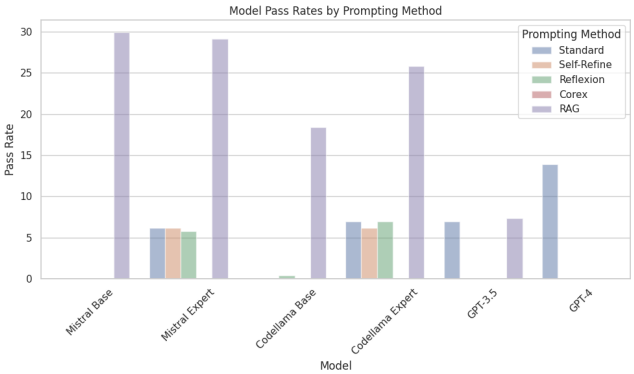
When exploring alternative prompting methods, *self-refine* did not lead to improvement, with constant performance under 3 settings and a slightly reduced performance under the expert Codellama. *Reflexion* minimally advanced the performance of the base Codellama model from 0% to 0.41%, but reduced the performance a little for other experiment settings on 7B LLMs. *Corex* highlighted the importance of multi-agent collaboration toward reasoning. However, a good performance under our 7B model experiment settings was not observed. With *Corex*, all the pass rates diminished to 0%. As a plausible explanation, all 7B models have not been instruction-tuned in our experiments, meaning they are not readily available for communication with humans. Instead, they are capable of generating a piece of sensible text conditioned on the user input. Therefore, it is difficult to follow the instructions that are key for *Corex* since the models are not built to communicate with each other (i.e., other LLMs).

Although the first three methods did not generate an improved result, *RAG* has significantly boosted the efficacy of all 7B models. The highest pass rate was achieved by the Mistral base model, reaching a peak of 29.9%. In addition, Codellama's expert model attained a 25.8% pass rate. These pass rates are two times as high as the pass rate attained by GPT-4 under standard prompting, highlighting the potential of retrieval-augmented strategies in fostering model performance. For GPT-3.5, however, RAG only achieved a small slight improvement of less than 1%. We leave the detailed investigation as future work.

In Table 1, the data fields marked with - are associated with GPT models. These experiments were not possible due to the financial constraint caused by the price of OpenAI API queries., As a key comparison, the SOTA on the miniF2F benchmark exhibited a 35.2% pass rate under pass@1 criteria (i.e., the average pass rate of model-generated proofs with only 1 generation per theorem statement). We highlight that the



(a) This bar plots show the pass rates of various LLMs under standard prompting. The expert models achieve competitive performance with GPT-3.5.



(b) This bar plots show the pass rates under all experiment settings grouped by each LLM, indicating the effectiveness of various prompting methods.

gap between our best result (29.9%) with SOTA is small, especially in comparison with the performance difference between different prompting methods on the same LLM. We believe that the paradigm of fine-tuning small LLMs on domain datasets in this research project would be effective, with the potential to match or even reach a new SOTA as the fine-tuning dataset scales.

As visualized in Figure 15b, the influence of varying prompting techniques on the expert 7B models was significant. While *Self-Refine* and *Reflexion* made limited progress, the *RAG* method proved its merit in boosting performance. Conversely, the 'Corex' method was the least effective, underscoring a potential disadvantage or incompatibility with the models in this context.

7 Contribution

7.1 VS Code Extension

To benefit the potential users of our study, namely mathematicians and AI researchers working in FTP, an extension for Visual Studio Code was developed. It is now directly installable from the editor's Marketplace. As a demonstration, relevant screenshots showcasing the functionalties of the extension are in Figure 16.

We first merged the LoRAs with base LLMs into single weight files. Then, we used FastChat to host the local LLMs, either the fine-tuned expert or base version. This leads to a local server with a local address, compatible with the OpenAI API library. Using the OpenAI API library, we can replace the API endpoint address from e.g., the default `https://api.openai.com/v1/` to e.g., `https://localhost:8001/v1`. In this way, a local LLM can be used for generation when prompting from an external source.

Next, we followed tutorials on how to build a VS Code extension and grasped the basics of Node.js and TypeScript. Two main functionalities were carefully designed.

7.1.1 Proof Auto-completion

First, with either a local LLM or calling to OpenAI GPT models, when the editor currently opens a `.lean` file, the extension is automatically activated. It reads the `.lean` file to obtain the theorem statement, ended with the symbols `:=` that also symbolize the beginning of a proof. Once the `.lean` file ends with `:=`, our extension calls an LLM to generate the completion, in our case of FTP, the proof to the theorem statement. After obtaining the response, the extension parses and extracts the proof and then shows the suggestion block with the label `LeanTheoremHint`. Upon the appearance of this

suggestion block, a user can hit the `Tab` key to enter the extracted proof in their editor automatically.

7.1.2 Proof Suggestion

Second, a user can access the command palette of VS code and directly ask for suggestions for an opened `.lean` file with two curated commands in our extension. The suggestions will appear in the form of a message box in the lower right corner of the editor instead of directly appearing in the window for that `.lean` file. The implementation is similar to the first functionality of proof auto-completion. However, it gives more flexibility to users since they can first examine a model's response and then copy and paste the useful part. In addition, it does not impose the constraint that the current `.lean` file has to end with `:=` that indicate the beginning of a proof.

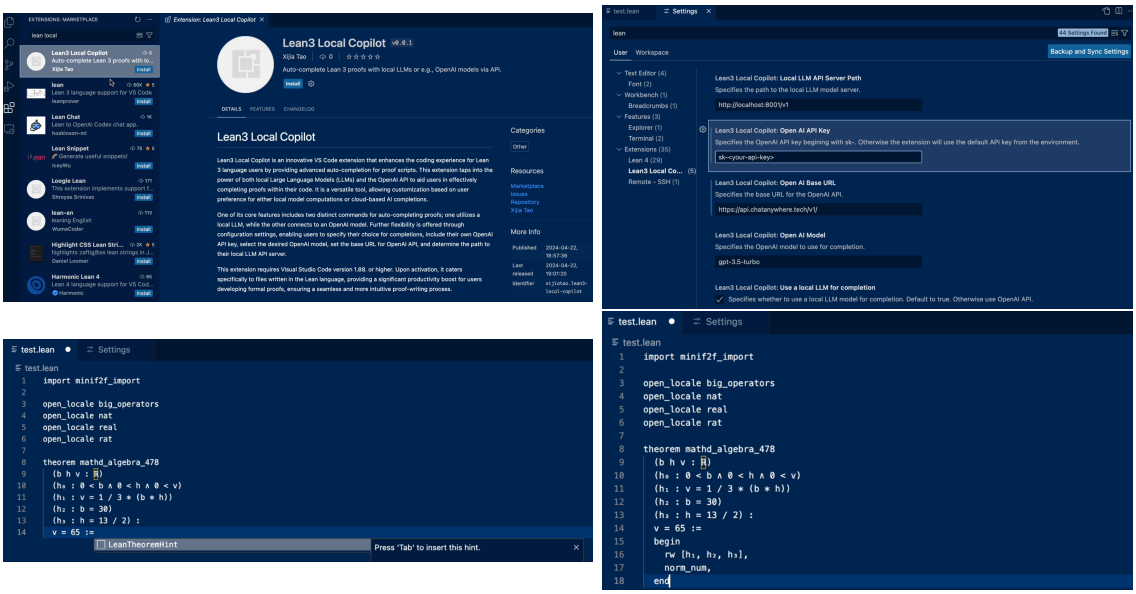


Figure 16: Screenshots showcasing the major functionalities of the developed VS Code extension named *Lean3 Local Copilot*. The choice of the name containing *local*, even if it works with normal OpenAI API endpoints, is to avoid a name clash with the more well-known Lean Copilot, as a contribution from the LeanDojo work [8].

7.2 Training Data Ablation

We conducted an ablation study on the composition and format of fine-tuning dataset. Specifically, the effects of the format of Lean proofs, contaminating the dataset with incorrect proofs, incorporating informal proofs into training, and the task of autoformalization were investigated.

7.2.1 Code Formatting

Many LLMs like ChatGPT format their response in markdown. For example, their response can include headings beginning with `#`, or code blocks surrounded by ```. We investigated that whether formatting the Lean codes in the training dataset in the same format as markdown would lead to an improved performance. The result was negative: it did not lead to an improvement, but a slight degradation from 6.17% to 4.92% for fine-tuned Mistral 7B models under standard prompting on the miniF2F benchmark.

7.2.2 Dataset Contamination

At first, we used the valid subset of MUSTARDSAUCE dataset to fine-tune the 7B models. Since the dataset also contains a random split that incorporates both valid and invalid data points (i.e., theorem statements with either correct or incorrect proofs), we decided to examine the effect of fine-tuning LLMs on this contaminated dataset split. The result was an unsurprising degradation from 6.17% to 4.10% for Mistral under standard prompting, highlighting the importance of a high-quality dataset.

7.2.3 Augment Data with Informal Proofs

In the original experiment settings, the dataset used for fine-tuning consist of only pairs of an informal theorem statement and a formal proof. However, it might also be useful to teach an LLM about the informal proof in natural language, leading to better guidance for becoming an FTP expert. Therefore, we fine-tuned another version of both 7B LLMs on trios of informal statements, informal proofs and formal proofs. The result is surprisingly negative: the pass rate degrade from 6.17% to 3.28% for Mistral under standard prompting. We suspect the possible reasons to be:

- incapability of base models: since 7B is a small parameter count for nowadays LLMs, the base models selected for further training might not be able to digest too much information at one time. That is, the context length increases with the addition of informal proofs. This interferes with the token probability distribution and leads to a worse performance.
- failure to extract proofs: the algorithm proposed in Section 4.2 might not take all possible proof formats into consideration, hence missing the correct proofs. The algorithm also take a heuristic approach in deciding which proof to evaluate when there are multiple proofs present in a model's response. In our experiments, we defaulted to either the first or last proof extracted.
- limitations of the fine-tuning method: since LoRA with 4-bit quantization was applied during fine-tuning, the fine-tuning performance might be lower than full-parameter fine-tuning. In addition, the relatively small dataset size and large learning rate might cause overfitting.

7.2.4 Autoformalization

Apart from MUSTARDSAUCE, another dataset that we investigated is MMA [24], which includes data pairs of an informal statement and its translated formal statement. The dataset does not include either informal or formal proofs like MUSTARD does. But the work highlighted the usefulness of training an LLM to do the translation from informal to formal language alone (aka autoformalization), in achieving better performance on the theorem proving task.

With this insight, we did an ablation on the autoformalization task. Both MMA and MUSTARDSAUCE are suitable for training use in this case. Therefore, we conducted two experiments with (1) MUSTARDSAUCE alone, and (2) both datasets. However, the pass rates are both 0% for Mistral under standard prompting. We suggest the reasons for this might overlap with those for data augmentation with informal proofs, incapability of 7B base models in particular. In addition, it suggest the inadequacy of only training on theorem statements, instead of the proofs. The intrinsic reasoning ability of LLMs might not be sufficient to generate proofs if only the basic syntax of the Lean formal language is known through memorizing theorems.

7.3 Summary

Our study makes several contributions to the field of computer-aided FTP.

- Firstly, we conduct a comprehensive literature review, examining the historical development and current state of computer-aided FTP.
- Secondly, we open-sourced two usable LLMs, namely Mistral and Codellama expert models fine-tuned on meticulously curated datasets specifically relevant to

FTP. These model weights can be found on HuggingFace.

- Thirdly, we perform an ablation study, systematically removing various techniques employed during training to assess their impact on model performance.
- Additionally, we provide a GitHub repository containing the codebase for evaluating and training models, ensuring the reproducibility of our results.
- Furthermore, we offer a convenient VS Code extension designed to assist FTP within the Lean 3 formal system.
- Lastly, we present valuable observations and insights addressing two research questions of significance in the field.

8 Limitations

The project utilized LoRA and INT4 quantization methods for fine-tuning. While these are resource-efficient techniques, they potentially limit the expressiveness and learning capability of the model as compared to full-precision fine-tuning, which might affect the precision of theorem proving.

A small fine-tuned dataset was employed, which may not fully represent the complexity and diversity of problems encountered in FTP. This limitation could affect the generalizability of the model, as it might not perform as well with more varied or complex input.

The experimentation that involves training was limited to two 7B parameter LLMs, due to GPU constraints. Expanding to larger models or incorporating a broader range of 7B models could reveal differences in performance and offer a more comprehensive

understanding of the two proposed RQs and how scaling affects theorem-proving capabilities.

The models were base versions and not instruction-tuned. With prompting methods requiring multi-agent collaboration or specific instructive prompts, these LLMS might underperform under the investigation for RQ2. Comparatively, instruction-tuned versions are designed to better comprehend and execute instructions, which could potentially enhance performance in these contexts.

Due to fiscal constraints, the project was unable to include more experiments with OpenAI models, e.g., RAG with GPT-4 was skipped due to the long context length of the method in nature and the prohibitive cost of a GPT-4 query. These missing experiments might bring new insights to the effect of prompting methods for RQ2. They might also demonstrate better performance, exacerbating the gap between experts and generalists.

9 Conclusion

In conclusion, our year-long study has explored the under-tapped potential of LLMs in the specialized domain of formal theorem FTP. Our integration of cutting-edge LLMs with the contributed datasets resulted in the fine-tuning of expert models, creating powerful tools tailored for the intricacies of formal reasoning and mathematical logic.

Through rigorous evaluation on the miniF2F benchmark, we have made strides in understanding the capabilities of these expert models in relation to general models and have investigated the impact of varied prompting techniques upon their effectiveness. The deployment of these models in the Lean 3 formal system, and their accessibility via a dedicated VS Code extension, stands as a testament to our commitment to community

collaboration and practical application.

Despite the promise shown by the expert models, our study is not without its limitations. The fine-tuning approach, scale of the dataset, diversity of the models examined, and the inherent characteristics of the base models used for fine-tuning have all confined the scope of our conclusions.

For future work, we advocate for the adoption of more encompassing and expressive fine-tuning methodologies, expansion of the dataset to capture a broader spectrum of FTP challenges, and application of a wider array of LLMs, including those that are instruction-tuned. This will not only refine our understanding of the role of LLMs in FTP but also enhance their practical utility in real-world applications. In doing so, we aspire to bridge the gap between AI and formal mathematics, carving a path toward more intuitive and collaborative theorem proving.

References

[1] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.

[2] S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever, "Formal mathematics statement curriculum learning," *arXiv preprint arXiv:2202.01344*, 2022.

[3] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[4] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[5] X. Zhao, W. Li, and L. Kong, "Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving," *arXiv preprint arXiv:2305.16366*, 2023.

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[7] Q. Sun, Z. Yin, X. Li, Z. Wu, X. Qiu, and L. Kong, "Corex: Pushing the boundaries of complex reasoning through multi-model collaboration," *arXiv preprint arXiv:2310.00280*, 2023.

[8] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar, "Leandojo: Theorem proving with retrieval-augmented language models," *arXiv preprint arXiv:2306.15626*, 2023.

[9] J. Harrison, J. Urban, and F. Wiedijk, "History of interactive theorem proving," in *Computational Logic*, 2014.

[10] K. Zheng, J. M. Han, and S. Polu, "Minif2f: a cross-system benchmark for formal olympiad-level mathematics," *arXiv preprint arXiv:2109.00110*, 2021.

[11] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[12] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[13] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," 2024.

[14] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[15] S. Hong, X. Zheng, J. Chen, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, *et al.*, "Metagpt: Meta programming for multi-agent collaborative framework," *arXiv preprint arXiv:2308.00352*, 2023.

[16] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui, "Agentcoder: Multi-agent-based code generation with iterative testing and optimisation," *arXiv preprint arXiv:2312.13010*, 2023.

[17] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *arXiv preprint arXiv:2009.03393*, 2020.

[18] J. M. Han, J. Rute, Y. Wu, E. W. Ayers, and S. Polu, "Proof artifact co-training for theorem proving with language models," *arXiv preprint arXiv:2102.06203*, 2021.

[19] G. Lample, T. Lacroix, M.-A. Lachaux, A. Rodriguez, A. Hayat, T. Lavril, G. Ebner, and X. Martinet, "Hypertree proof search for neural theorem proving," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26337–26349, 2022.

[20] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[21] C. Zheng, H. Wang, E. Xie, Z. Liu, J. Sun, H. Xin, J. Shen, Z. Li, and Y. Li, "Lyra: Orchestrating dual correction in automated theorem proving," *arXiv preprint arXiv:2309.15806*, 2023.

[22] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, and V. Misra, "Solving quantitative reasoning problems with language models," 2022.

[23] Y. Huang, X. Lin, Z. Liu, Q. Cao, H. Xin, H. Wang, Z. Li, L. Song, and X. Liang, "Mustard: Mastering uniform synthesis of theorem and proof data," *arXiv preprint arXiv:2402.08957*, 2024.

[24] A. Q. Jiang, W. Li, and M. Jamnik, "Multilingual mathematical autoformalization," 2023.

[25] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[26] N. Shinn, B. Labash, and A. Gopinath, "Reflexion: an autonomous agent with dynamic memory and self-reflection," *arXiv preprint arXiv:2303.11366*, 2023.

[27] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.