Iterative Prompting of LLMs for Automated Theorem Proving

Guillermo Rodriguez[1]

KIT Department of Informatics Institute of Information Security and Dependability (KASTEL)

Abstract. The rise of Large Language Models (LLMs) has introduced a new approach for guiding Interactive Theorem Provers (ITPs)—tools originally designed to help users construct formal proofs. This paper proposes to explore the potential of integrating LLMs with ITPs to address high school inequality problems. We examine LLMs' ability to complete formal proofs through different feedback models. We evaluate this strategy using GPT-4o benchmarked against POLYA. Overall, the project aims to deliver an understanding of various LLMs' capabilities in logic tasks with different feedback strategies.

Keywords: LLM · Prompting · Feedback · GPT-4o

1 Task and Motivation

This project aims to explore the intersection between two technologies, that naturally seem to complement each other.

Interactive Theorem Provers (ITPs) are computer programs that allow the user to conduct formal proofs. Cambridge University released the Higher Order Logic system in the 1980s, which in the subsequent years became widely used by the scientific community.[1]. Currently, most standard ITPs are based on backwards reasoning. In a backwards reasoning system, a user enters a theorem as input to be proven as a goal. Then the program, orchestrated by the user, can either:

- Apply automated theorem provers to solve that goal.
- Divide the main goal into distinct subgoals, for which the process described (either solving the subgoals or further dividing them) can be repeated.

Regarding why ITPs have proven to be valuable in the context of proof theory, Huang, Dhariwal, Song, *et al.* [2] argue that they offer programmable environments to conduct proofs, where the user can customize the program behavior to tackle distinct problem domains. Another key advantage of this type of program, is the automatic verification of the logical steps taken [2].

Nonetheless, ITPs still have some disadvantages over traditional informal proofs. Many mathematical proofs do not have an implementation on ITPs yet

[3]. Furthermore, developing new proofs or implementing existing ones can potentially require some expertise. A good example of this is the Kepler Conjecture, which took over 20 years to be formalized [4].

Large Language Models (LLMs) have led to significant advances in numerous scientific fields, including proof theory. When coupled with Interactive Theorem Provers (ITPs), LLMs can function as an oracle, guiding the decision-making process in proofs. This pairing has the potential to reduce or potentially eliminate the need for human intervention.

However, LLMs have notable drawbacks, particularly in terms of reliability [5]. They can produce incorrect or biased results, which poses a significant challenge for their application in formal proofs. This is where the environment of ITPs becomes invaluable. ITPs offer a rigorous verification process that ensures the correctness of logical steps, thereby compensating for the reliability issues of LLMs.

The combination of LLMs and ITPs presents substantial opportunities. By integrating the intuitive, heuristic capabilities of LLMs with the deterministic, rule-based processes of ITPs, we can potentially alter how proofs are approached. This integration not only enhances the reliability of the proofs but also leverages the strengths of both technologies, making the process more efficient and robust.

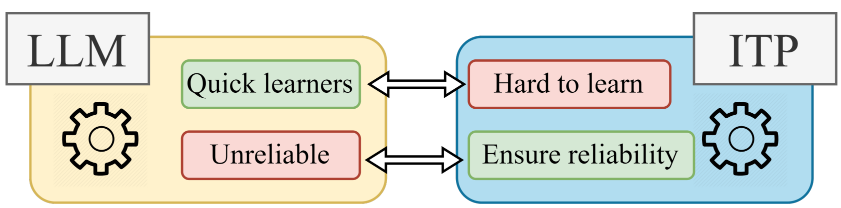

Fig. 1: Diagram of strengths and weaknesses between LLM and TPS

As we observe, the strengths and weaknesses of these two technologies complement each other well (see Figure 1). Moreover, if a reliable interaction can be designed, we could potentially reach a point where the progress in the field of proof theory could be fully automatized.

However, despite the high potential, we still need to figure out how to design these interactions optimally. As argued by Daniel Selsam [6], designing the interaction of these technologies as a non-deterministic program seems promising. A non-deterministic program generates a search tree, to which a search algorithm can be applied. For this project, we propose to apply this principle and also explore how to integrate domain-specific knowledge to improve overall reliability and performance. This leads to our research question:

Can providing iterative feedback enhance the performance of Large Language Models in generating formal proofs targeted to an Interactive Theorem Prover?

2 Background

To further discuss the project, additional background is needed. First, in Section 2.1, an overview of the progress of LLMs in the field of logic is presented. Next, concrete examples of interactions between LLMs and ITPs are discussed in Section 2.2. Finally, the gap in knowledge is summarized in Section **??**, and how this project intends to address it is explained.

2.1 Large Language Models

To understand the impact that LLMs could have in the field of Proof Theory, we first explore one of the early pioneers of LLMs, analyzing its potential and limitations. We then examine a study that proposes various techniques to overcome specific challenges regarding LLMs. Finally, we assess a breakthrough.

Early Promising Results of LLMs In 2020 shortly after the release of OpenAI's GPT-3, a study testing its capabilities was published. It serves not only to point into the realm of possibilities LLMs could bring to the table, but also highlights the core issues of such models [7].

It combined its augmentation on the number of parameters with "In-context learning", which consists of providing the model in the form of natural language an explanation of the task to be solved, or some previous examples and then relying on the model's prediction capabilities to fulfill the task. Compared against models relying on fine-tuning, i.e. being trained on a dataset specific to the task to be fulfilled, in Natural Language Processing (NLP tasks).

GPT-3 showed similar or better results (achieving state-of-the-art performance) than the fine-tuned models if used in a few-shot setting. Few-shot prompting consists of giving the already trained model some examples of the tasks to be solved for it to recognize the patterns and achieve better performance. One of the insights of this paper was how an LLM could profit from a few-shot setting.

Regarding its hindrances, one issue explained by the authors is the model lacking reliability in its answers. Regarding some types of questions or surpassing a specific length input, this model began to lose coherence and contradict itself. Another problem was the model retaining biases from the data it was trained.

The concern of reliability is not exclusive to GPT-3 but is a common challenge among various Large Language Models (LLMs) as well [8]. This leads to the following question: How can we enhance the reliability of LLMs?

On the Improvement of Reliability of LLMs An article published in the blog *OpenAI's Cookbook* by Sanders [5] addresses LLMs' reliability issue.

First, the article explains that the capabilities of a model are influenced by the context, i.e. that a better task-adequate prompting can improve a lot the results of the model. Subsequently, some techniques that lead to more efficient prompting are then presented (as depicted in Table 1).

Chain of thought	LLMs can profit a lot from resolving a task step by step and retrieving the answer as a sequence of steps instead of retrieving the answer directly. More specifically in the case of GPT 3, if asked to "Think step by step" in the prompt against just retrieving the answer directly on a collection of word math problems (MultiArith), the result showed to change from en 18% to 79% of correctness [9].
Split bigger tasks into smaller tasks	The article also argues that segmenting a task into smaller ones gives a better result. For example, if asked to provide a summary of a foreign in the same language GPT3 sometimes retrieves the summary in English. Instead, if asked to first recognize the language the language and then asked to offer a summary, this mistake does not occur.
Few-shot prompting	Few-shot prompting, i.e. providing a few examples to an LLM on how to solve a specific task and then asking it to perform it, can substantially improve the performance. In the benchmark GSM8K with school math problems, GPT3 achieved 57% resolution through chain of thought and few-shot prompting instead of the 18% achieved by the vanilla version. In the case of few-shot prompting, it takes about 8 examples to saturate performance in the case of GPT3 [10].
Fine-tunning	The technique of fine-tuning consists of training the model on datasets that contain relevant data for a specific task. This method applied to LLMs can arguably maximize the performance, but it may take a thousand examples [11].

Table 1: Overview on strategies proposed to enhance LLMs reliability

Overall, the article offers a wide comprehension of different techniques to improve the reliability of the outputs of an LLM. It even points to publications that have made use of these techniques and have proven their value. the question remaining is what would happen if we combine them together?

Minerva: The Watershed Moment The study by Lewkowycz, Andreassen, Dohan, *et al.* [12] marked a significant milestone in the synergy between Automated Theorem Proving (ATP) and Large Language Models (LLMs), introducing *Minerva*, an LLM that significantly exceeded the anticipated performance levels in math and logic benchmarks at that time.

Minerva, an adaptation of Google's PaLM model, incorporated techniques mentioned in Section 2.1 and employed majority voting, a method where multiple responses from an LLM to the same query are sampled, and the most frequent answer is selected as the final one [13].

In order to discuss its results, we consider Minerva's performance on the MATH benchmark, comprising 12,500 complex mathematic problems. Prior predictions estimated that LLMs would solve about 12.7% of this benchmark within a year of its release [14]. In contrast, by November 2022, Minerva had completed 50.3% of the benchmark, a feat not expected until 2025. Google's PaLM model, the foundation for Minerva, achieved approximately 8.8% completion. However, Minerva's limitations included the lack of automatic verification for answers and minimal control over specific tasks.

Despite these challenges, Minerva's remarkable performance demonstrated the potential of LLMs to significantly contribute to the future of Proof theory, mathematics, and logic.

2.2 Examples of Interaction between LLMs and ITPs

As discussed by Li, Sun, Murphy, *et al.* [15], the intersection of LLMs and ITPs is being explored in multiple ways, primarily in proof step generation, autoformalization, and proof search. We will present the main approach on which this project is based.

Proofstep generation - *ChatBot Approach* Arguably, one of the most intuitive ways to combine Large Language Models (LLMs) and Interactive Theorem Provers (ITPs) is presented by Yang, Swope, Gu, *et al.* [16]. This approach is structured as follows:

1. A user inputs a theorem in the formal language of the ITP.
2. The LLM generates the next proof step.
3. The ITP verifies the generated proof step. If the step is incorrect, the ITP provides an error message, and the LLM makes corrections.
4. If the proof step is correct, the ITP applies it and updates the proof, then returns the new proof state along with the next proof goal.

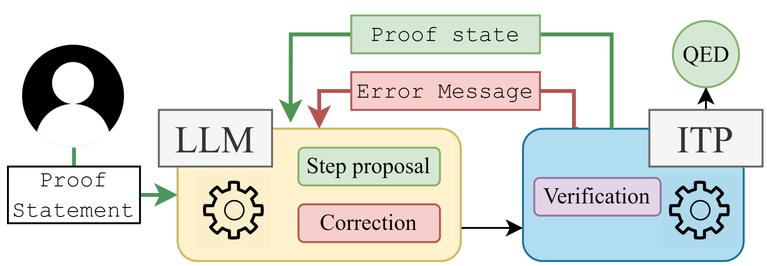

Fig. 2: Representation of the Chatbot Approach

This cycle repeats until the proof is complete (see Figure 2). The integration of LLMs with ITPs via the ChatBot approach not only streamlines the interaction but also enhances the effectiveness of theorem proving by leveraging the strengths of both technologies.

The LeanDojo tool is based on Lean4, which uses a more procedural, tactic-driven proving style. However, what if we were to take this approach and apply it to an interactive theorem prover (ITP) that uses a more declarative proving style? Declarative proof systems are more similar to traditional mathematical writing, focusing on developing a proof through logical assertions and the respective reasoning. This contrasts with the step-by-step procedural approach found in Lean4, where the focus is on the tactics used instead of the assertions.

Exploring this shift to a declarative style could be beneficial for large language models (LLMs), as they might perform better when working with formats that closely resemble natural language. By aligning the proof style with the LLM's natural language processing strengths, we might observe improvements in proof generation and completion.

Furthermore, another question is also presented: How does the way we reflect the proof errors to the LLM affect the ability to correct itself? Since declarative systems often produce clearer, more human-readable error messages, we could experiment with different methods of error explanation to see which ones guide the LLM most effectively.

3 Methodology

In order to present the experiment conducted for this project, we need to explain the methodology that builds it. Section 3.1 covers the main libraries we used and their background. Next, how these elements interact with each other is presented in Section 3.2. Finally, the experiment protocol is detailed in Section 3.3

3.1 Main Components

Isabelle/HOL is a software tool that serves as a proof assistant, or interactive theorem prover (ITP), designed to facilitate the formalization and verification of mathematical proofs. Its development began in 1986 as a collaboration between the University of Cambridge and the Technical University of Munich [17]. Isabelle/HOL incorporates various logical constructs, including quantifiers over operators and functions. Contrary to other ITPs, Isabelle/HOL uses a declarative proving style. In Isabelle/HOL, writing proofs is more similar to natural language than in other provers, as proofs are interleaved with conjectures and their corresponding justifications. This declarative proving style may be particularly effective when used with a large language model (LLM) [18]. Additionally, we utilize the `isabelle_client` library [19], which provides a way to interact with the ITP through `Python`.

OpenAI: For this project, we selected OpenAI's Python library `openai-python` [20] to design the pipeline for our experiment. This allows us to combine the verification capabilities of `isabelle_client` with the available LLMs through `openai-python` in Python. For the experiments, we chose the `GPT-4o` model, as it is the latest release.

Polya: We will develop our pipeline to solve the examples presented by Avigad, Lewis, and Roux [21]. These examples contain exercises of varying complexity belonging to various problem domains, including linear and nonlinear inequalities and properties of exponential and logarithmic functions. These exercises align well with our project because they present a balance between complexity and accessibility. Additionally, these examples have a well-defined scope. Even more, by having access to the correct answers, we gain a reliable control-check mechanism. Additionally, having access to the correct answers allows to observe how the answers deviate from a possible solution.

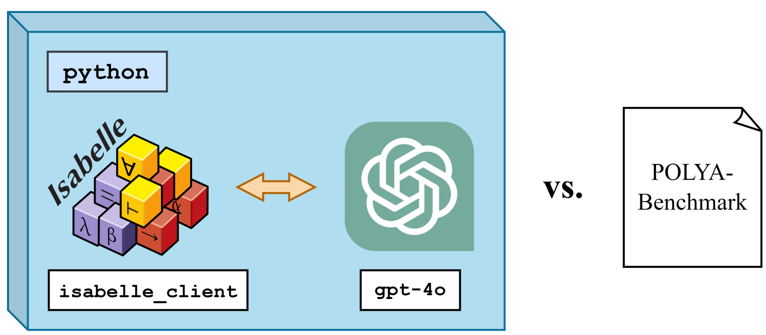

Fig. 3: Schema of main elements used to develop the pipeline.

3.2 Pipeline

In this section, we provide an overview of how the components introduced in Section 3.1 interact with each other. The pipeline design is illustrated in Figure 4 in the form of pseudocode. It is important to note that the system description presented here is a simplified summary of the original design. Following this, we discuss the parallelization mechanism and the feedback models in detail. The complete code is available through the following link:

https://github.com/guillerplazas/IPLATP.

```
SYSTEM_DESCRIPTION="Complete the given Isabelle/HOL proof"

def feedback(solution, errors) -> prompt:... # How we prompt mistake corrections

def process_problem(theory, n_prob, succ_count):
    current_prompt = theory
    for att in range(8): # Maximal attempts for each exercise
        sol = ask_llm(current_prompt)
        errors = check(solution)
        current_prompt = feedback(sol, errors) if errors!=[] else:
            succ_count[0] += 1
            return True
    return False

def worker(client, theory_queue, success_count):
    while True:
        break if theory_queue.empty() else:
            process_problem(theory_queue.get(), client, success_count)
            theory_queue.task_done()

def benchmark(thr_files):
    clients = [start_isabelle_client() for _ in range(3)]
    thr_queue = Queue(enumerate(thr_files, start=1))
    success_count = [0]
    [tasks = [worker(client, thr_queue, success_count)] for client in clients]
    print(f"SCORE: {success_count[0]} out of {len(thr_files)}")
```

Fig. 4: Pseudocode of the pipeline

Parallelization To improve the efficiency of our experimental pipeline, we used `asyncio`'s parallelization capabilities. This allowed us to handle multiple proof tasks concurrently rather than sequentially. Specifically, we employed the following elements:

- An **asynchronous queue** to store the benchmark problems that need to be solved.
- A **worker function** that retrieves proofs from the queue and processes them independently.
- A **pool of workers**: each interacts with an instance of the Isabelle client (isabelle_client) to verify the proofs generated by the large language model (LLM).

By running the verification in parallel, the computation time was significantly reduced facilitating experimentation and analysis.

Feedback To address one of the key questions in this project, it is essential to establish which feedback systems will be compared. By "feedback system," we refer to the prompt provided to the LLM when it generates an incorrect answer. To limit the scope of possibilities, we focus on three distinct feedback methods (see Table 2).

None	Just asking the LLM to provide another answer with no feedback. Serves as the baseline.
Standard Feedback	Providing the proposed answer and error codes that the `isabelle_client` has retrieved with the line number.
Annotated Program	Processing the error codes of the isabelle_client with the solution so that we obtain an annotated program, where if there exists an error it is directly annotated on the error line.

Table 2: Overview of the different feedback strategies

LLM proposes wrong answer:

```
1 lemma "(0 :: real) < 1 + y^2"
2 proof -
3   (*Use the fact that for any real number y, y^2 is always non-negative.*)
4   have "y^2 ≥ 0"  by (rule power2_ge_0)
5   (*Adding a non-negative number y^2 to 1 results in at least 1.*)
6   then have "1 + y^2 ≥ 1"  by (simp add: add_nonneg_nonneg)
7   (* Since 1 is greater than 0, 1 + y^2 is also greater than 0. *)
8   thus ?thesis  by simp
9 qed
```

Feedback is given:

No Feedback

```
Please correct the following proof:...
```

Standard Feedback

```
Please correct the following proof:...

It contains the following errors:
line 4 Undefined fact: "power2_ge_0"
line 8 Failed to finish proof: goal (1 subgoal):  1. 0 < 1 + y^2}
```

Annotated Feedback

```
Correct the following proof. The errors are annotated as comments:
lemma "(0 :: real) < 1 + y^2"
proof -
  have "y^2 ≥ 0" by (rule power2_ge_0) (*ERROR Undefined fact: "power2_ge_0"*)
  then have "1 + y^2 ≥ 1" by (simp add: add_nonneg_nonneg)
  thus ?thesis by simp  (* ERROR: Failed to finish proof*)
qed
```

Fig. 5: Example of the feedback types given a wrong answer

Testing with no feedback serves as a baseline to compare against using a feedback model. The standard feedback modality represents a simple approach to indicating errors, requiring minimal effort to generate the corresponding prompt. Finally, the annotated program feedback offers a more refined method, where the proposed answer and the detected errors are combined. Figure 5 provides an example for each of these cases.

3.3 Experiment Protocol

The CPU used to conduct the experiments was an AMD Ryzen 9 6900HX CPU. The procedure involved testing each of the three feedback mechanisms against the complete Polya benchmark six times to ensure statistical reliability. In our setup, the program was allowed up to eight attempts to solve each problem before it was declared unsuccessful. In case of success, we annotate at which attempt a correct answer is achieved. After the experiments, we computed the mean and standard deviation of the performance metrics to quantitatively assess the effectiveness of each feedback model. We do this for the values $n = 1, 3, 6, 8$, being n the number of feedback rounds allowed in case of error. All the necessary code and resources for replicating the experiments are provided in the following repository: https://github.com/guillerplazas/IPLATP.

4 Results

In this section, we present the results from the project's experiment (see Section 3). The results can be found in Table 3. We analyze the performance of each feedback model and compare them to the baseline case, where no feedback is provided. Additionally, we discuss the number of attempts required for a successful result with each one of the mechanisms.

Feedback	**n=1**	**n=3**	**n=6**	**n=8**
None		15.2 ± 9.1%	22,8 ± 6.1%	23.9 ± 5.3%
Standard	6.7±4.2%	22.2 ± 7.8%	31.1 ± 5.8%	32.8 ± 7.1%
Annotated Prog.		20.0 ± 4.7%	34.4 ± 4.6%	40.0 ± 2.2%

Table 3: Results Summary of Experiment. The values of $n = 1, 3, 6, 8$ represent the number of feedback rounds allowed in case of an error. The table displays the mean success rates and their standard deviations for each feedback mechanism across these values of n. Each experiment was repeated 6 times.

Overall capacity As we can observe, the way errors are communicated to the LLM significantly impacts overall performance when multiple rounds of feedback are provided. The results indicate that more detailed feedback leads to better scores. The annotated feedback model, which incorporates both the errors and the original proof into a single prompt, appears to guide the LLM more effectively in revising its answers. Additionally, it is also suggested that the performance of the annotated feedback model could potentially benefit from having more attempts. Due to time and budget constraints, exploring this possibility is left for future work.

Saturation of performance with no feedback We also observe that when no feedback is provided, the LLM does not show substantial improvement when asked the same question multiple times. Its performance quickly reaches a plateau, or saturates, without guidance on how to correct its mistakes. This means that indicating information about the errors allows the LLM to adjust and improve, highlighting the importance of error-specific feedback.

Similar performance on few attempts When the number of attempts is limited (e.g., $n = 3$ or $n = 6$), we observe that the performance is quite similar across both feedback models—standard and annotated feedback. This suggests two possible interpretations: either the dataset used in our experiment is too small (only 6 runs of each model), or the LLM may require more attempts to fully benefit from a more sophisticated feedback system. Extending the number of runs or experimenting with more refined feedback models is left for future research.

5 Conclussion

In this project, the combination of Large Language Models (LLMs) with Interactive Theorem Provers (ITPs) has been explored, specifically GPT-4o with Isabelle/HOL. Our goal was to determine how different feedback mechanisms (e.g. prompting strategies when an error is found) impact the LLM's ability to complete formal proofs, using the Polya benchmark as a testing ground.

Our experiments show that the way we prompt an LLM to correct an error indeed impacts its performance. Specifically, providing detailed, annotated feedback led to better proof completions compared to offering no feedback or just standard error messages. The annotated feedback model, which integrates error messages directly into the proof script, allowed the LLM to adjust and improve more effectively over multiple attempts.

Without feedback, the LLM's performance quickly reaches a plateau. Additionally, while the standard and annotated feedback models showed similar performance with a limited number of attempts, the annotated model showed greater potential for improvement over extended iterations.

These findings suggest that LLMs working with declarative proving styles (such as Isabelle/HOL) benefit from having effective feedback mechanisms to correct errors.

Future Work Regarding the results obtained and the observations made during this project, there are some directions for further investigation. Firstly, more feedback systems could be developed. We have tested three feedback models, but with more time and effort, better-performing models could likely be developed. Secondly, enhancing the overall pipeline with reliability-improving techniques could be beneficial. Incorporating a search algorithm, implementing few-shot prompting, and making use of other reliability-enhancing techniques could further improve the system's effectiveness.

References

[1] F. Marić, "A Survey of Interactive Theorem Proving," *Zbornik radova*, Jul. 2015.

[2] D. Huang, P. Dhariwal, D. Song, and I. Sutskever, "GamePad: A Learning Environment for Theorem Proving," in *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=r1xwKoR9Y7`.

[3] G. Lample, M.-A. Lachaux, T. Lavril, *et al.*, "HyperTree Proof Search for Neural Theorem Proving," in *Thirty-sixth Conference on Neural Information Processing Systems*, Nouvelle Orléans (LA), United States, Nov. 2022. [Online]. Available: `https://hal.science/hal-04360432`.

[4] T. Hales, M. Adams, G. Bauer, *et al.*, "A FORMAL PROOF OF THE KEPLER CONJECTURE," en, *Forum of Mathematics, Pi*, vol. 5, e2, 2017, ISSN: 2050-5086. DOI: `10.1017/fmp.2017.1`. [Online]. Available: `https://www.cambridge.org/core/product/identifier/S2050508617000014/type/journal_article` (visited on 12/09/2023).

[5] T. Sanders, *Techniques to improve reliability — OpenAI Cookbook*, en, Sep. 2022. [Online]. Available: `https://cookbook.openai.com/articles/techniques_to_improve_reliability` (visited on 12/27/2023).

[6] Daniel Selsam, *The IMO Grand Challenge (AITP 2020)*, Aussois France, Sep. 2020. [Online]. Available: `https://www.youtube.com/watch?v=GtAo8wqWHHg` (visited on 12/11/2023).

[7] T. Brown, B. Mann, N. Ryder, *et al.*, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[8] X. Shi, J. Liu, Y. Liu, Q. Cheng, and W. Lu, *Know where to go: Make llm a relevant, responsible, and trustworthy searcher*, 2023. arXiv: `2310.12443 [cs.IR]`.

[9] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large Language Models are Zero-Shot Reasoners*, _eprint: 2205.11916, 2022.

[10] J. Wei and D. Zhou, *Language Models Perform Reasoning via Chain of Thought*, en, May 2022. [Online]. Available: `https://blog.research.google/2022/05/language-models-perform-reasoning-via.html` (visited on 12/27/2023).

[11] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman, *Star: Bootstrapping reasoning with reasoning*, 2022. arXiv: 2203.14465 [cs.LG].

[12] A. Lewkowycz, A. J. Andreassen, D. Dohan, *et al.*, "Solving Quantitative Reasoning Problems with Language Models," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=IFXTZERXdM7.

[13] X. Wang, J. Wei, D. Schuurmans, *et al.*, "Self-Consistency Improves Chain of Thought Reasoning in Language Models," en, Mar. 2022.

[14] J. Steinhardt, *AI Forecasting: One Year In*, en, Jul. 2022. [Online]. Available: https://bounded-regret.ghost.io/ai-forecasting-one-year-in/ (visited on 12/18/2023).

[15] Z. Li, J. Sun, L. Murphy, *et al.*, "A survey on deep learning for theorem proving," *ArXiv*, vol. abs/2404.09939, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269148388.

[16] K. Yang, A. Swope, A. Gu, *et al.*, "LeanDojo: Theorem proving with retrieval-augmented language models," in *Neural Information Processing Systems (NeurIPS)*, 2023.

[17] M. Wenzel, *Isabelle — Sketis*, en-US. [Online]. Available: https://sketis.net/isabelle (visited on 07/15/2023).

[18] A. Q. Jiang, S. Welleck, J. P. Zhou, *et al.*, *Draft, sketch, and prove: Guiding formal theorem provers with informal proofs*, 2023. arXiv: 2210.12283 [cs.AI].

[19] B. Shminke, *Isabelle-client*, original-date: 2021-02-20T16:35:00Z, Jul. 2024. [Online]. Available: https://github.com/inpefess/isabelle-client (visited on 09/16/2024).

[20] *Openai/openai-python*, original-date: 2020-10-25T23:23:54Z, Oct. 2024. [Online]. Available: https://github.com/openai/openai-python.

[21] J. Avigad, R. Y. Lewis, and C. Roux, "A Heuristic Prover for Real Inequalities," *Journal of Automated Reasoning*, vol. 56, no. 3, pp. 367–386, Mar. 2016, ISSN: 1573-0670. DOI: 10.1007/s10817-015-9356-y. [Online]. Available: https://doi.org/10.1007/s10817-015-9356-y.

A Examples of Correct Solutions

In this section, we present examples of LLM and ITP interactions that led to correct results. Figures 6 and 7 correspond to the standard and annotated feedback models, respectively. For readability, some code has been adjusted—such as avoiding line breaks—but the content remains unchanged.

LLM's parsed answer:

```
1  lemma "(∀ x y. x ≤ y ⟶ f x ≤ f y) ⟹ (u::real) < v ⟹ 1 < v ⟹ (x::real) ≤ y ⟹ f x + u < v^2 + f y"
2  proof -
3    assume monotone: "∀ x y. x ≤ y ⟶ f x ≤ f y"
4    assume u_less_v: "u < v"
5    assume v_greater_1: "1 < v"
6    assume x_le_y: "x ≤ y"
7    from monotone and x_le_y have "f x ≤ f y" by auto
8    have "u < v^2"
9    proof -
10     have "v^2 > v * 1" using v_greater_1 by (simp add: power2_gt_1)
11     thus "u < v^2" using u_less_v by (simp add: mult.commute)
12   qed
13   then have "f x + u < f y + v^2" using `f x ≤ f y` by auto
14   thus ?thesis by (simp add: add.commute)
15 qed
```

Round 1 Feedback:

```
1  Please correct the following proof:
2
3  lemma "(∀ x y. x ≤ y ⟶ f x ≤ f y) ⟹ (u::real) < v ⟹ 1 < v ⟹ (x::real) ≤ y ⟹ f x + u < v^2 + f y"
4  proof -
5    assume monotone: "∀ x y. x ≤ y ⟶ f x ≤ f y"
6    assume u_less_v: "u < v"
7    assume v_greater_1: "1 < v"
8    assume x_le_y: "x ≤ y"
9    from monotone and x_le_y have "f x ≤ f y" by auto
10   have "u < v^2"
11   proof -
12     have "v^2 > v * 1" using v_greater_1 by (simp add: power2_gt_1)
13     thus "u < v^2" using u_less_v by (simp add: mult.commute)
14   qed
15   then have "f x + u < f y + v^2" using `f x ≤ f y` by auto
16   thus ?thesis by (simp add: add.commute)
17 qed
18
19 It returns the following errors: line 12 Undefined fact: "power2_gt_1"
```

LLM's correct answer:

```
1  lemma "(∀ x y. x ≤ y ⟶ f x ≤ f y) ⟹ (u::real) < v ⟹ 1 < v ⟹ (x::real) ≤ y ⟹ f x + u < v^2 + f y"
2  proof -
3    assume monotone: "∀ x y. x ≤ y ⟶ f x ≤ f y"
4    assume u_less_v: "u < v"
5    assume v_greater_1: "1 < v"
6    assume x_le_y: "x ≤ y"
7    from monotone and x_le_y have "f x ≤ f y" by auto
8    have "u < v^2"
9    proof -
10     have "v^2 = v * v" by (simp add: power2_eq_square)
11     have "v * 1 < v * v" using v_greater_1 by (simp add: mult_strict_right_mono)
12     thus "u < v^2" using u_less_v by (simp add: power2_eq_square)
13   qed
14   then have "f x + u < f y + v^2" using `f x ≤ f y` by (auto simp add: add_strict_mono)
15   thus ?thesis by simp
16 qed
```

Fig. 6: Successful pipeline example using the standard feedback model. The feedback first provides the solution, followed by any identified errors.

LLM's parsed answer:

```
lemma "(∀ x. f x ≤ 2) ⇒ (0::real) < w ⇒ u < v ⇒ u + w * (f x - 1) < v + w"
proof -
 assume H1: "∀ x. f x ≤ 2"
 assume H2: "(0::real) < w"
 assume H3: "u < v"
 show "u + w * (f x - 1) < v + w"
 proof -
  (* From H1, we have the specific bound on f x: *) have "f x ≤ 2" using H1 by auto
  (* This implies f x - 1 <= 1: *) hence "f x - 1 ≤ 1" by simp
  (* Multiply by w (positive), so inequality is preserved:*) have "w*(f x -1) ≤ w*1" using H2 by (simp add: mult_left_mono)
  (* Simplifying the right-hand side: *) hence "w * (f x - 1) ≤ w" by simp
  (* Adding u to both sides of the inequality: *) have "u + w * (f x - 1) ≤ u + w" by (simp add: add_mono)
  (* As u < v from H3, so u + w < v + w since w > 0: *) have "u + w < v + w" using H2 H3 by (simp add: add_strict_mono)
  (* Combine the inequalities: *) show "u + w * (f x - 1) < v + w" using `u + w * (f x - 1) ≤ u + w` `u + w < v + w` by simp
 qed
qed
```

Round 1 Feedback:

```
Correct the proof. The errors are annotated as comments:

lemma "(∀ x. f x ≤ 2) ⇒ (0::real) < w ⇒ u < v ⇒ u + w * (f x - 1) < v + w"
proof -
 assume H1: "∀ x. f x ≤ 2"
 assume H2: "(0::real) < w"
 assume H3: "u < v"
 show "u + w * (f x - 1) < v + w"
 proof -
  have "f x ≤ 2" using H1 by auto
  hence "f x - 1 ≤ 1" by simp
  have "w*(f x - 1) ≤ w*1" using H2 by (simp add: mult_left_mono) (* ERROR: Failed to prove 1 subgoal:  0 < w ⟹ f x ≤ 2 *)
  hence "w * (f x - 1) ≤ w" by simp
  have "u + w * (f x - 1) ≤ u + w" by (simp add: add_mono) (* ERROR: Failed to prove 1 subgoal:  w * (f x - 1) ≤ w *)
  have "u + w < v + w" using H2 H3 by (simp add: add_strict_mono)
  show "u + w * (f x - 1) < v + w" using `u + w * (f x - 1) ≤ u + w` `u + w < v + w` by simp
 qed
qed
```

LLM's correct answer:

```
lemma "(∀ x. f x ≤ 2) ⇒ (0::real) < w ⇒ u < v ⇒ u + w * (f x - 1) < v + w"
proof -
 assume H1: "∀ x. f x ≤ 2"
 assume H2: "(0::real) < w"
 assume H3: "u < v"
 show "u + w * (f x - 1) < v + w"
 proof -
  have Fx: "f x ≤ 2" using H1 by auto
  hence "f x - 1 ≤ 1" by simp
  (* Multiply by w ... *) have "w * (f x -1) ≤ w * 1" using `f x - 1 ≤ 1` H2 by (simp add: mult_left_mono)
  (* Simplifying the right-hand side: *) hence "w * (f x - 1) ≤ w" by simp
  (* Adding u to both sides: *) have "u + w * (f x - 1) ≤ u + w" using `w * (f x - 1) ≤ w` by auto
 (* As u < v H3, so u + w < v + w since w > 0: *) have "u + w < v + w" using H2 H3 by (simp add: add_strict_mono)
  (* Combine the inequalities: *) have "u + w * (f x - 1) < v + w" using `u + w * (f x - 1) ≤ u + w` `u + w < v + w` by linarith
  show ?thesis by fact
 qed
qed
```

Fig. 7: Successful pipeline example using the annotated feedback model. The feedback first provides the solution, with the errors marked as comments.