Enhancing LLM’s Mathematical and Logical Reasoning Abilities

CS 297 Report

Presented to:

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

Presented By:

Naga Rohan Kumar Bayya

December, 2024

ABSTRACT

Large Language Models (LLMs) excel at natural language tasks but often underperform in solving mathematical problems and logical reasoning due to their reliance on linguistic patterns rather than formal reasoning and symbolic manipulation. This project is the first phase of a two-semester effort to enhance the mathematical and logical capabilities of LLMs by addressing their inherent limitations, including insufficient training on structured mathematical data, limited multi-step reasoning abilities, and constrained context windows. In this phase, we focus on fine-tuning pre-trained LLMs using curated datasets such as MATH [1] and GSM8k [2], integrating symbolic reasoning tools like Mathics and LEAN, and employing process supervision and Chain-of-Thought (CoT) prompting to bridge the gap between language understanding and mathematical precision. The outcomes of this phase include improved model performance on complex math problems, basic theorem proving, and enhanced reasoning accuracy for solving word problems. These results lay the groundwork for the next semester, where the focus will shift to scaling these methods, refining hybrid models, and expanding their capabilities to tackle more advanced mathematical and logical reasoning tasks. This work represents a significant step toward developing robust hybrid models capable of handling diverse and challenging mathematical and logical domains.

Keywords: Large Language Models, LLMs, Mathics, Lean, Chain-of-Thought prompting, Fine Tuning

INTRODUCTION

Large Language Models (LLMs) have transformed natural language processing (NLP) by demonstrating proficiency in generating human-like text, understanding context, and performing a variety of language-related tasks. However, their performance on mathematical and logical reasoning problems remains suboptimal. Hendrycks, et al. proposed that this limitation arises because LLMs are predominantly trained on natural language corpora, which lack the structure and symbolic reasoning required for precise mathematical computations and multi-step logical deduction. Consequently, their responses to mathematical queries often prioritize linguistic plausibility over accuracy and mathematical reasoning, resulting in incorrect yet plausible-sounding answers. This project addresses these limitations by enhancing LLMs with specialized training and symbolic reasoning tools.

Mathematics and logic form the backbone of many critical applications, ranging from scientific research and engineering to decision-making systems and automated theorem proving. Enhancing the mathematical and logical reasoning capabilities of LLMs is therefore essential for expanding their utility in these domains. To address this challenge, this project investigates methods to improve LLM performance on mathematical problems and logical reasoning tasks. The approach combines fine-tuning on curated datasets, integrating symbolic reasoning systems, and leveraging advanced training methodologies such as process supervision and Chain-of-Thought (CoT) prompting [10].

This report documents the progress made during the first phase of a 2 semester long project in addressing the limitations of LLMs in handling mathematical problems. It covers the fine-tuning of LLMs using datasets such as MATH [1] and GSM8k [2], integration with symbolic tools like Mathics and LEAN to perform complex calculations and theorem proving in addition to creation of custom datasets to evaluate model performance. The deliverables include enhanced models capable of performing basic theorem proving, solving word problems, and conducting symbolic computations. This work represents a step toward hybrid AI systems that combine the generalization capabilities of LLMs with the rigor of formal reasoning.

The rest of this report is organized into a set of 4 deliverables. Deliverable 1 involves fine-tuning GPT-2 on the MATH and GSM8k datasets using the parameter-efficient DoRA technique to improve its ability to solve mathematical problems. Deliverable 2 focuses on integrating Mathics, an open-source symbolic mathematics engine, with LangChain to enable LLMs to perform complex calculations like integrals. Deliverable 3 utilized the LEAN proof assistant to prove the Infinite Primes Theorem, showcasing the power of formal verification tools for theorem proving. Finally, Deliverable 4 combined Mathics and LEAN in a unified langchain pipeline to solve mathematical word problems and prove theorems, demonstrating the potential of hybrid AI systems that merge LLMs with symbolic computation and formal reasoning tools. Last section summarizes the findings and concludes the work done .

DELIVERABLE-1: FINE-TUNING GPT-2 ON MATH & GSM8k DATASETS

This deliverable focuses on fine-tuning GPT-2, a pre-trained LLM, on the MATH dataset to improve its ability to solve complex mathematical problems. This task leverages the DoRA (Decomposed Low-Rank Adaptation) technique for parameter-efficient fine-tuning, forming a foundation for adapting an LLM to answer mathematical problems.

1. Parameter-Efficient Fine-Tuning (PEFT) Techniques

LLMs usually consist of billions of tunable weights. In order to train such huge models from scratch, a lot of computational resources and time are required. PEFT techniques have emerged as effective methods for adapting large pre-trained models to specialized tasks without retraining the entire model while achieving the same result. By focusing on a subset of parameters, these techniques reduce computational requirements, memory usage, and training time, making them suitable for tasks with limited resources. The idea stems from the fact that a pre-trained LM can be applied to down-stream tasks by changing a small subset of parameters [3]. Two widely used PEFT techniques are Low-Rank Adaptation (LoRA) and Decomposed Low-Rank Adaptation (DoRA) were explored in this project to fine-tune GPT-2 effectively.

1.1 Low-Rank Adaptation (LoRA)

LoRA is a PEFT method that adapts pre-trained models by introducing low-rank decomposition matrices to update the model's weights. Instead of modifying the entire parameter space, LoRA updates only low-rank representations, significantly reducing the number of trainable parameters. It introduces rank decomposition matrices for efficient parameter updates and supports fine-tuning on downstream tasks with minimal computational overhead.

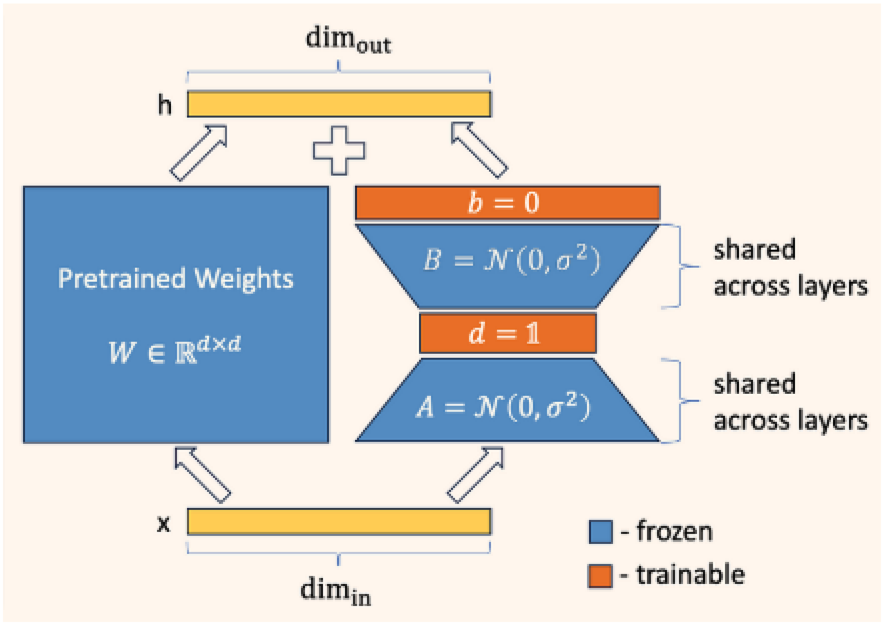


Fig 1: LoRA architecture

Drawing reference from Edward Hu et al., LoRA was applied on self-attention layers of a GPT-2 transformer during fine-tuning on the MATH and GSM8k datasets. This enabled efficient adaptation to mathematical reasoning tasks

1.2 Decomposed Low-Rank Adaptation (DoRA)

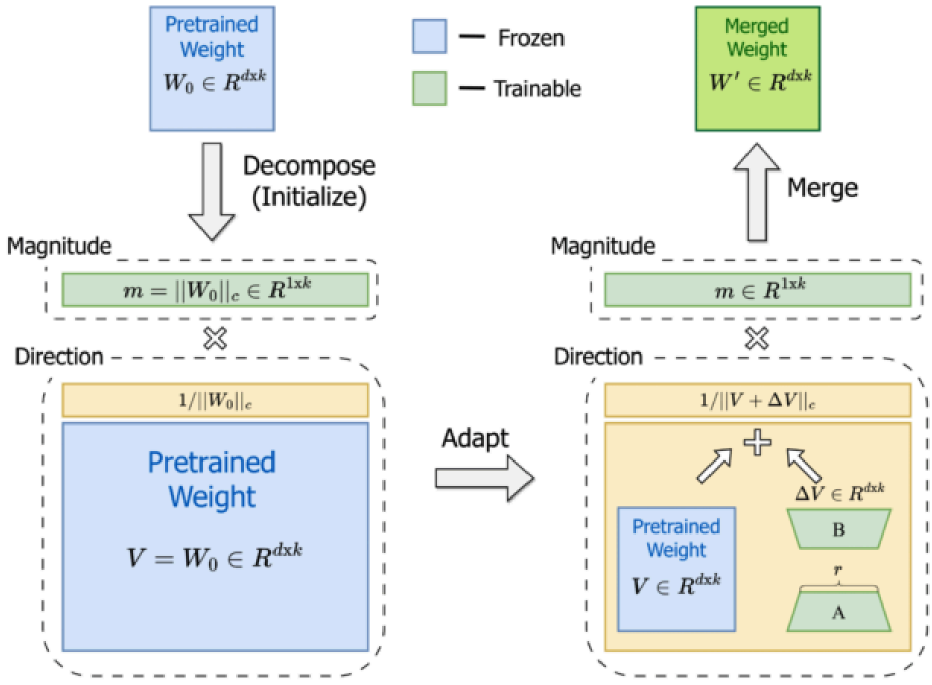


Fig 2: DoRA Architecture. Courtesy: NVIDIA's blog post

DoRA builds on the concepts of LoRA by introducing additional decomposition techniques to bridge the gap between LoRA and Full tuning arising from LoRA's limited learning capabilities [4]. DoRA achieves this through decomposition of original matrix W into Magnitude M and direction D matrices each of which can be tuned independently.

2. MATH Dataset

MATH dataset is used to evaluate and enhance the mathematical problem-solving abilities of machine learning models. It comprises 12,500 high-school-level math problems with varying difficulty levels ranging from 1 to 5. Of these, 7,500 problems are designated for training, and 5,000 for testing. The dataset covers topics such as Prealgebra, Algebra, Number Theory, Counting and Probability, Geometry, Intermediate Algebra, and Precalculus. Each problem is formatted in LaTeX with visual elements generated using the Asymptote vector graphics language. Solutions include step-by-step derivations and a boxed final answer, facilitating interpretability and learning using Chain of thought , as proposed by Jason Wei et al.

Complemented by the AMPS dataset, which contains over 100,000 problems for pretraining, the MATH dataset enables models to improve their accuracy and reasoning abilities when fine-tuned effectively.

A base LLM model has been fine-tuned on the MATH dataset using DoRA [12] . The process involved initializing the GPT-2 model and tokenizer using the Hugging Face library, preprocessing and tokenizing the dataset, and splitting it into 90% training and 10% validation subsets. Training configurations included a learning rate of 5e-5, a batch size of 8, and one epoch due to computational constraints. The fine-tuned model demonstrated improved mathematical reasoning compared to the base GPT-2, though limitations in GPU resources and dataset pre-training constrained its overall accuracy. It is published on hugging face as rkumar1999/gpt2-fine-tuned-math.

The image below shows the result of fine-tuning highlighting that the model has learnt mathematical equations and model in formal mathematical terms.

- **Finetuned Model:**

```
model_name = 'rkumar1999/gpt2-fine-tuned-math'
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
generate_response(model, tokenizer)
```

```
Question:
Find the value of  x  if  2x + 3 = 11 .

Generated Solution:
\left[\begin{array}{ll} \right] &= \dfrac{1}{2x+3x} & \cdot \frac{11}{11}&=\left(11x\right) \end{align*}\right\]$
Now let's look at the equation for the sum of x and y. We can see that we can solve for x by multiplying by the square root of
```

- **Base GPT-2 Model:**

```
model_name = 'gpt2'
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
generate_response(model, tokenizer)
```

```
Question:
Find the value of  x  if  2x + 3 = 11 .

Generated Solution:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
```

3. GSM8k Dataset

The GSM8k (Grade School Math 8K) is a dataset of 8.5K high quality linguistically diverse grade school math word problems. The dataset was created to support the task of question answering on basic mathematical problems that require multi-step reasoning. It offers unique challenges, requiring the model to not only understand problem statements but also perform sequential reasoning to arrive at correct solutions [2]

The dataset was prepared using custom preprocessing and tokenization methods to ensure compatibility with GPT-2's architecture. Fine-tuning parameters included a learning rate of 5e-5, a batch size of 4, and 10 epochs to achieve better optimization. DoRA was again utilized to adapt the model efficiently for this specialized task. The fine-tuned model displayed enhanced capabilities in solving grade-school-level word problems, producing more structured and logical solutions compared to the base GPT-2 model. However, like the MATH dataset, computational limitations impacted the model's performance on more complex tasks. The trained model is available on hugging face as rkumar1999/gpt2-fine-tuned-gsm8k.

The results showed that GSM8k dataset enabled the model to improve its handling of structured mathematical problems and extended these capabilities to word problems requiring multi-step reasoning.

- **Finetuned Model:**

```
device = th.device("cpu")
tokenizer = GPT2Tokenizer.from_pretrained("rkumar1999/gpt2-fine-tuned-gsm8k")
model = GPT2LMHeadModel.from_pretrained("rkumar1999/gpt2-fine-tuned-gsm8k")
model.to(device)

generate_response(model, tokenizer)
```

```
Triggered calculator, answer 1575
what is the value of 21*75 equals to?
21*75 = 21*75 = 21*75 = $<<21*75=1575>>15
```

- **Base GPT-2 Model:**

```
device = th.device("cpu")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")
model.to(device)

generate_response(model, tokenizer)
```

```
what is the value of 21*75 equals to?


I'm not sure. I'm not sure if it's a good idea to use a calculator, or if it's a good idea to use a calculator, or if it's a go
```

DELIVERABLE-2: Integrating Mathics Tool to Compute Integrals

This deliverable focuses on integrating Mathics, an open-source symbolic mathematics engine, into a LangChain pipeline to enable the LLM to compute complex mathematical expressions, such as integrals. The primary goal was to enhance the computational abilities of LLMs by combining their linguistic reasoning capabilities with Mathics' symbolic computation power.

1. Mathics Overview

Mathics is an open-source tool that provides functionality similar to Mathematica, supporting algebraic manipulation, calculus, and plotting. It allows users to perform symbolic computations, define functions, and visualize data. The flexibility of Mathics, with its command-line and web-based interfaces, makes it an ideal tool for integration with LLMs for solving complex mathematical problems [8].

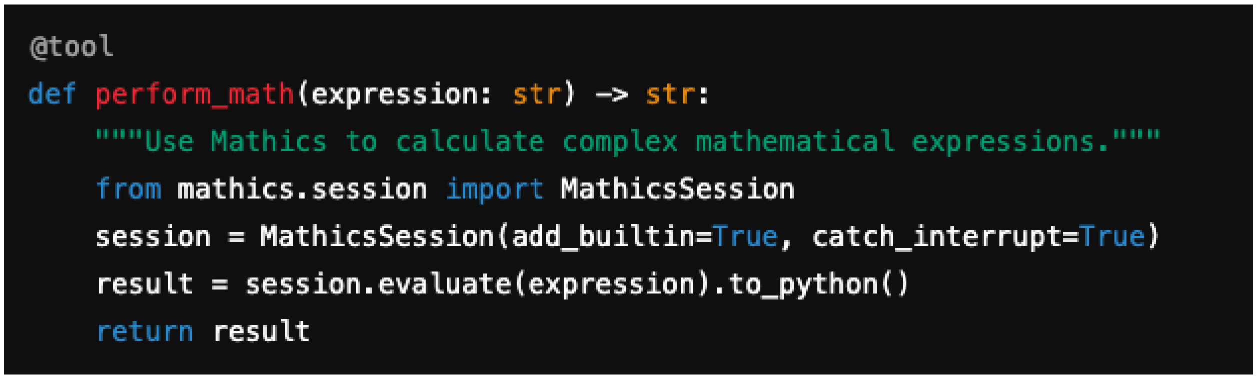
2. Integration with LLM

2.1 Framework

The integration was implemented using LangChain, a framework designed to enable LLMs to invoke external tools dynamically. LangChain’s create_tool_calling_agent function was used to create an agent that could seamlessly call Mathics for specific tasks [9].

2.2 Tool Implementation

A custom tool, perform_math, was designed to interact with Mathics. The tool used the MathicsSession class to handle mathematical queries, converting input expressions into a Python-compatible format. This allowed for smooth communication between the LLM and Mathics.



```python
@tool
def perform_math(expression: str) -> str:
    """Use Mathics to calculate complex mathematical expressions."""
    from mathics.session import MathicsSession
    session = MathicsSession(add_builtin=True, catch_interrupt=True)
    result = session.evaluate(expression).to_python()
    return result
```

A LangChain agent was built using the Mathics tool, allowing the LLM to invoke Mathics when mathematical computations were required. The agent was capable of handling user queries, determining whether to process them linguistically or pass them to Mathics for symbolic computation.

2.3 Results

The system was tested by solving integrals, such as computing the integral of sin ^2 (x) from 0 to π / 2. The Mathics-integrated model successfully computed the result (π / 4), demonstrating accurate and reliable performance.

[illegible]

```
tools = [perform_math]
augmented_agent= create_tool_calling_agent(
    llm=model,
    tools=tools,
    prompt=prompt)

agent_executor = AgentExecutor(agent=augmented_agent, tools=tools, verbose=True)
result = agent_executor.invoke({
    "input": ("Integrate sin^2(x) from 0 to pi/2 ")
})
```

```
> Entering new AgentExecutor chain...

Invoking: `perform_math` with `{'expression': 'Integrate[Sin[x]^2, {x, 0, Pi/2}]'}`


System`Times[1/4, System`Pi]The integral of sin^2(x) from 0 to pi/2 is equal to 1/4 * pi.

> Finished chain.
```

This deliverable highlights the potential of integrating LLMs with symbolic computation engines like Mathics. By combining the linguistic reasoning of LLMs with Mathics' mathematical capabilities, the system was able to handle complex computations that would otherwise be beyond the scope of the LLM alone. This integration sets the stage for more advanced hybrid systems capable of tackling diverse mathematical and logical challenges.

DELIVERABLE-3: Infinite Primes theorem using LEAN

The objective of this deliverable was to demonstrate the utility of formal proof verification systems in mathematical reasoning and to explore the integration of automated theorem proving tools with language models. It proves the Infinite Primes Theorem, a fundamental result in number theory, using the LEAN proof assistant. It aligns with the overall project goal by showcasing how formal reasoning tools can complement LLMs to enhance their logical capabilities and address complex mathematical challenges.

1. LEAN Overview

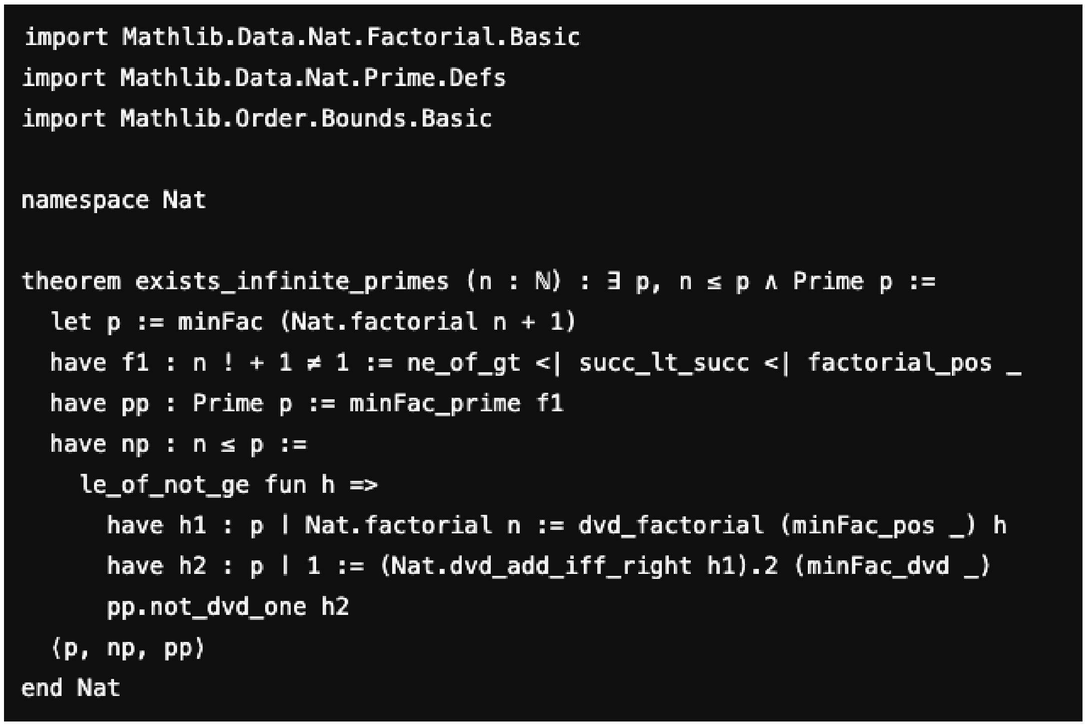
LEAN is a proof assistant and functional programming language based on the calculus of constructions with inductive types. It provides a robust environment for formalizing mathematics and verifying proofs. LEAN enables mathematicians and computer scientists to construct and validate mathematical proofs in a formal, rigorous manner, ensuring correctness [5].

2. LEAN Project Setup

The proof was implemented in LEAN 4. The setup involved installing LEAN 4 and Lake, its project management tool. Initializing a new LEAN project using lake new and compiling the project using lake build to check for syntax and type errors before proceeding with the implementation.

3. Proof Implementation

The Infinite Primes Theorem was formalized and proved in LEAN. The proof relies on classical number theory concepts, particularly the factorial of a number and the properties of prime numbers. The proof is as follows:



```
import Mathlib.Data.Nat.Factorial.Basic
import Mathlib.Data.Nat.Prime.Defs
import Mathlib.Order.Bounds.Basic

namespace Nat

theorem exists_infinite_primes (n : ℕ) : ∃ p, n ≤ p ∧ Prime p :=
  let p := minFac (Nat.factorial n + 1)
  have f1 : n ! + 1 ≠ 1 := ne_of_gt <| succ_lt_succ <| factorial_pos _
  have pp : Prime p := minFac_prime f1
  have np : n ≤ p :=
    le_of_not_ge fun h =>
      have h1 : p ∣ Nat.factorial n := dvd_factorial (minFac_pos _) h
      have h2 : p ∣ 1 := (Nat.dvd_add_iff_right h1).2 (minFac_dvd _)
      pp.not_dvd_one h2
  ⟨p, np, pp⟩
end Nat
```

Fig 3: This proof demonstrates that for any natural number n, there exists a prime p≥n by considering the smallest prime factor of n!+1. It uses the fact that n!+1 cannot be divisible by any prime ≤n, ensuring the infinitude of primes.

DELIVERABLE-4: Integrating Mathics and LEAN to Solve Mathematical Word Problems

Deliverable 4 aimed to create a unified pipeline by integrating Mathics, a symbolic computation tool, and LEAN, a formal proof assistant, to solve complex mathematical word problems. The primary objective was to demonstrate how combining these tools with LLMs can enhance their ability to tackle problems requiring symbolic manipulation and theorem proving.

1. Methodology

This integration was implemented using LangChain's framework, enabling seamless interaction between the LLM, Mathics, and LEAN. The approach involved leveraging Mathics for symbolic computation tasks, such as evaluating mathematical expressions, and using LEAN to formalize and prove theorems. The LLM served as a bridge, parsing user inputs and invoking the appropriate tool for solving specific tasks.

1.1 Integration of Mathics

The Mathics tool, previously integrated in Deliverable 2, was extended to support more complex expressions required for solving word problems. Mathics was invoked through a LangChain agent using its MathicsSession class to handle symbolic computations. For example, queries like "Find the integral of $\sin^2(x)$ from 0 to $\pi/2$" were passed to Mathics, which returned precise results.

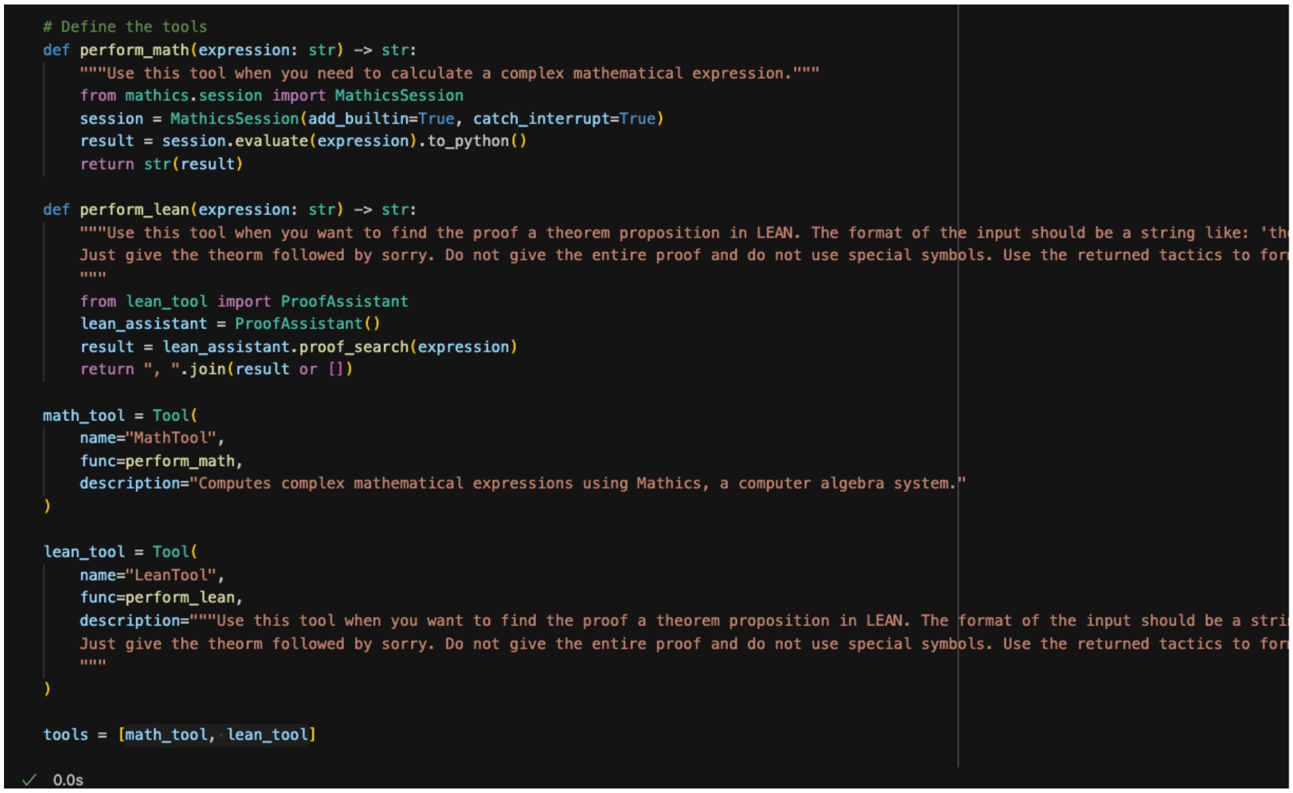
1.2 Integration of LEAN

A custom class, Proof Assistant, was designed to interact with LEAN. The class used a pre-trained sequence-to-sequence model [6] to generate proof tactics. The integration included the following key components:

- Proof Initialization: LEAN's REPL (Read-Eval-Print Loop) was used to define initial proof states with placeholders (sorry) for unresolved steps
- Tactic Generation: The model generated potential tactics for each proof state, which were recursively applied to explore solution paths
- Proof Validation: Each step was verified to ensure correctness, with LEAN providing feedback on unresolved goals or errors.

1.3 Pipeline Integration

The combined system allowed for dynamic switching between Mathics and LEAN based on the problem's requirements. For example, while Mathics handled numerical calculations, LEAN proved supporting mathematical theorems. This enabled the LLM to solve problems involving both symbolic computation and formal reasoning.



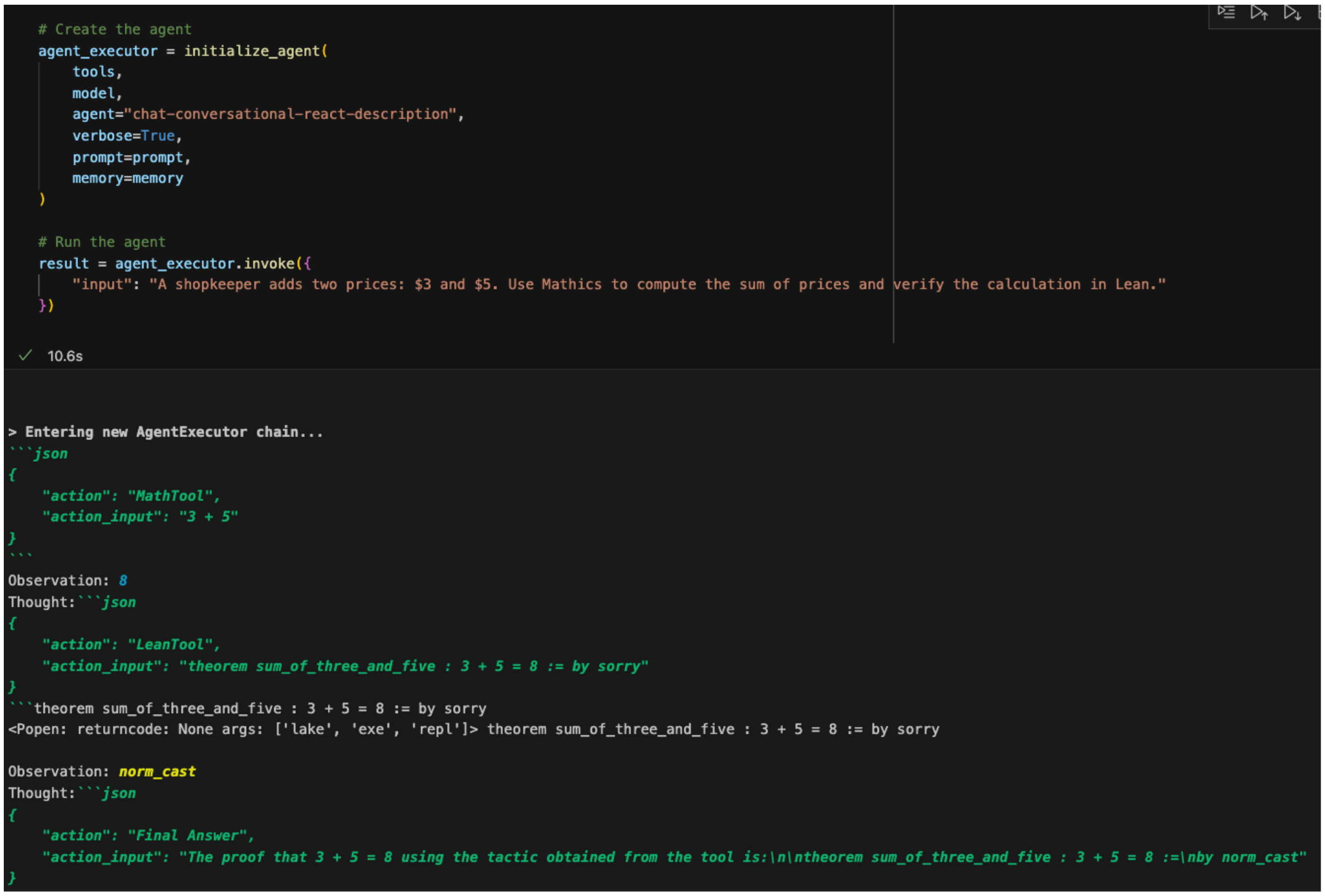
```
# Define the tools
def perform_math(expression: str) -> str:
    """Use this tool when you need to calculate a complex mathematical expression."""
    from mathics.session import MathicsSession
    session = MathicsSession(add_builtin=True, catch_interrupt=True)
    result = session.evaluate(expression).to_python()
    return str(result)

def perform_lean(expression: str) -> str:
    """Use this tool when you want to find the proof a theorem proposition in LEAN. The format of the input should be a string like: 'th
    Just give the theorm followed by sorry. Do not give the entire proof and do not use special symbols. Use the returned tactics to for
    """
    from lean_tool import ProofAssistant
    lean_assistant = ProofAssistant()
    result = lean_assistant.proof_search(expression)
    return ", ".join(result or [])

math_tool = Tool(
    name="MathTool",
    func=perform_math,
    description="Computes complex mathematical expressions using Mathics, a computer algebra system."
)

lean_tool = Tool(
    name="LeanTool",
    func=perform_lean,
    description="""Use this tool when you want to find the proof a theorem proposition in LEAN. The format of the input should be a stri
    Just give the theorm followed by sorry. Do not give the entire proof and do not use special symbols. Use the returned tactics to for
    """
)

tools = [math_tool, lean_tool]
```

✓ 0.0s

Fig 4: Using Tool class of langchain to invoke Mathics and LEAN

2. Results

The integration successfully demonstrated the complementary strengths of Mathics and LEAN in solving mathematical word problems. Mathics provided accurate solutions for symbolic expressions, while LEAN ensured logical rigor by verifying underlying mathematical principles. Together, these tools significantly enhanced the LLM's problem-solving capabilities, enabling it to tackle more complex and multi-faceted tasks. For instance, the pipeline accurately solved problems like evaluating integrals and proving supporting theorems to validate results.



```
# Create the agent
agent_executor = initialize_agent(
    tools,
    model,
    agent="chat-conversational-react-description",
    verbose=True,
    prompt=prompt,
    memory=memory
)

# Run the agent
result = agent_executor.invoke({
    "input": "A shopkeeper adds two prices: $3 and $5. Use Mathics to compute the sum of prices and verify the calculation in Lean."
})
```

✓ 10.6s

~~~
> Entering new AgentExecutor chain...
```json
{
    "action": "MathTool",
    "action_input": "3 + 5"
}
~~~
Observation: 8
Thought:```json
{
    "action": "LeanTool",
    "action_input": "theorem sum_of_three_and_five : 3 + 5 = 8 := by sorry"
}
```theorem sum_of_three_and_five : 3 + 5 = 8 := by sorry
<Popen: returncode: None args: ['lake', 'exe', 'repl']> theorem sum_of_three_and_five : 3 + 5 = 8 := by sorry

Observation: norm_cast
Thought:```json
{
    "action": "Final Answer",
    "action_input": "The proof that 3 + 5 = 8 using the tactic obtained from the tool is:\n\ntheorem sum_of_three_and_five : 3 + 5 = 8 :=\nby norm_cast"
}
```

Fig 5: Invoking the Langchain

Ⅶ. CONCLUSION

This project makes advancements in enhancing the mathematical and logical reasoning capabilities of LLMs by integrating them with specialized tools and datasets. LLMs, despite their remarkable performance in natural language understanding, face limitations in solving complex mathematical problems and logical tasks due to their training predominantly on natural language data. We address these challenges by fine-tuning LLMs on domain-specific datasets, employing parameter-efficient techniques, and integrating symbolic computation and proof assistant tools.

Through the deliverables, the project successfully showcased how fine-tuning LLMs on datasets like MATH and GSM8k can improve their ability to handle structured mathematical problems and word problems requiring multi-step reasoning. The use of DoRA, a parameter-efficient fine-tuning technique, allowed for effective adaptation of GPT-2 with limited computational resources. While these models demonstrated noticeable improvements, further pre-training on foundational datasets and access to greater computational power could unlock their full potential.

The integration of Mathics as a symbolic computation tool and LEAN as a proof assistant further expanded the scope of the project. Mathics empowered the LLM to perform accurate symbolic manipulations, such as evaluating integrals and solving equations, while LEAN enabled formal theorem proving, exemplified by the successful proof of the Infinite Primes Theorem. Combining these tools into a unified pipeline allowed for seamless transitions between symbolic computations and logical verifications, enabling the resolution of complex mathematical word problems.

Overall, it underscores the value of hybrid AI systems that combine the linguistic reasoning capabilities of LLMs with the rigor of symbolic computation and formal proof verification tools. This is the first phase of a two-semester project, laying the foundation for scaling these methods and addressing more advanced challenges in the next phase. Moving forward, further optimization of these systems, including pre-training on larger mathematical datasets and enhancing integration pipelines, could yield even more powerful and versatile AI systems capable of addressing broader and more complex challenges in mathematics and logic.

Ⅷ. REFERENCES

[1] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. arXiv preprint arXiv:2103.03874. https://doi.org/10.48550/arXiv.2103.03874

[2] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, Ł., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*. https://doi.org/10.48550/arXiv.2110.14168

[3] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*. https://doi.org/10.48550/arXiv.2106.09685

[4] Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., & Chen, M.-H. (2024). DoRA: Weight-Decomposed Low-Rank Adaptation. *arXiv preprint arXiv:2402.09335*. https://doi.org/10.48550/arXiv.2402.09335

[5] de Moura, L., Kong, S., Avigad, J., van Doorn, F., & von Raumer, J. (Year). The Lean Theorem Prover (System Description). *Microsoft Research* and *Carnegie Mellon University*.

[6] Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., & Anandkumar, A. (Year). LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. *Caltech*, *NVIDIA*, *MIT*, *UC Santa Barbara*, *UT Austin*. Retrieved from https://leandojo.org

[7] leanprover-community. (n.d.). repl: A simple REPL for Lean 4, returning information about errors and sorries. GitHub repository. Retrieved December 9, 2024, from https://github.com/leanprover-community/repl

[8] (2022). Mathics: A Free, Open-Source Alternative to Mathematica. Retrieved December 9, 2024, from https://mathics.org/

[9] LangChain. (2024). LangChain Documentation. Retrieved December 9, 2024, from https://python.langchain.com/docs/introduction/

[10] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., & Zhou, D. (2016). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Google Research, Brain Team. arXiv:2201.11903v6*

[11] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T.,

Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language Models are Few-Shot Learners. 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada

[12] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. NeurIPS. Retrieved December 9, 2024, from https://github.com/hendrycks/math