A Study on Improving Mathematical Reasoning using Multi-Agent Collaboration

Praveen Kumar R[1*]

[1*]Computer Science and Engineering, PES University, Electronic City, Bengaluru, 560100, Karnataka, India.

Corresponding author(s). E-mail(s): thejavamonk@gmail.com;

Abstract

Interactive question-answering with human tutors is a proven method for teaching mathematics. There is a potential for large language models (LLMs) to automate parts of this process, including interactive discussions. Though LLMs have demonstrated significant potential, they are limited by problems such as hallucination, accessing outdated knowledge, knowledge gaps, and insufficient data for training. LLMs can produce incorrect or irrelevant answers. LLM Agents with Tools can enhance LLM responses by incorporating verified external knowledge into the model's prompts, and tools can be used to verify the thoughts and solve this issue. In this paper, we designed experiments on mathematical reasoning using LLM Agent architectures integrated with tools on a sample of 700 questions from MATH and 1,319 questions from GSM8k test datasets. We determine the base performance of GPT-4o-mini using zero-shot and evaluate the efficacy of different LLM agent architectures like Reflection, Reflexion, and Multi-agent collaboration implemented using LangGraph in solving complex problems in the Math dataset. We demonstrate that multi-agent collaboration can improve response accuracy and reduce hallucinations.

Keywords: Agents, LangGraph, Langchain, Reflection, Reflexion, Multi-agent collaboration, GPT-4o-mini, MATH, GSM8K

1 Introduction

Large Language Models are orders of magnitude ahead of humans in many tasks in many areas [1], so they are the most promising candidates for building autonomous agents [19], especially in multi-agent collaboration. The tasks are very complex, and the agents have comparative advantages like information sharing and collective decision-making among agents with specialized skills. Considering the essential nature of mathematics and the profusion of descriptions of mathematical problems in human prose in all areas of science and engineering, it is interesting to understand to what extent large language models can solve them.

This work investigates using conversational agents to solve complex mathematics problems. These problems typically involve several steps, where an accurate solution to any step implies the correctness of the solution to the problem. Conversations, together with code execution, provide a convenient model to refine and debug each of these steps iteratively. We present a novel conversational framework built using LangGraph for chat-based LLMs. We leverage an LLM-based agent that, together with a user proxy agent, iteratively simulates a conversation until it solves the math problem. We further explore and build on effective prompting techniques. We evaluate the conversational framework on the MATH dataset, comprising challenging mathematics problems. Specifically, we selected 700 level-5 problems due to their complexity. They are also so tricky that even college students have the potential to struggle with half of them. We conducted multiple experiments with zero-shot prompting, the Program of Thoughts agent, self-reflection, and Reflexion to determine the baseline performance of the model. Lastly, we conducted studies using the multi-agent collaboration conversational model developed using LangGraph. Our result shows that our approach improves mathematical reasoning by 6% with good performance across problem types.

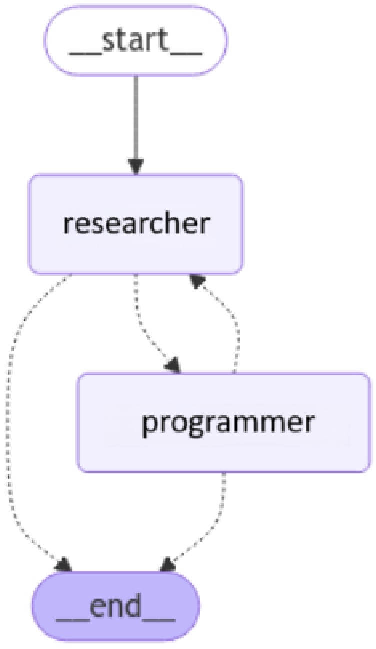

Fig. 1 Multi-Agent Collaboration

2 Related Work

2.1 MATHEMATICAL REASONING

LLMs, such as GPT-4o-mini and GPT-4, and open-source models like Llama 3, have demonstrated astounding capabilities in various natural language applications. These developments will have immense consequences for sectors from healthcare to finance. Yet, it is widely accepted that current LLMs have a significant flaw, i.e., they perform poorly on critical reasoning tasks like mathematical, commonsense, abductive, and multi-hop reasoning. This lack has driven the research community to investigate various approaches to enlarge their reasoning skills. This quest has resulted in the emergence of methods such as prompting techniques and augmenting LLMs with tools and Agent architectures.

2.2 PROMPT ENGINEERING

Directed Prompting variants, such as Chain Of Thought [17], work under the assumption that although LLMs have strong capabilities in predicting sequences of tokens, they are usually not designed for explicit reasoning. This explicit, guided sequence reverses most often implicit reasoning in LLMs, transforming the model to become capable of deductive outputs and solving more complex problems. [7], Chain of Thought (Self-Consistency) [16] uses an ensemble approach; this method works by asking the LLM to generate multiple answers for each question. So, we can say that the coherence among responses makes them more credible. Self-consistency is based on the premise that if an LLM produces similar responses to a single prompt, then those responses are more likely correct. This description of the method involves the LLM responding to a query multiple times and checking if each response is dependent on one another. Alignment across different dimensions can be quantified through content overlap, semantic similarity tests, or more advanced metrics (like BERT scores and n-gram overlaps).

The Program Of Thought [3] is a prompting strategy aimed at improving the reasoning capabilities of LLMs. Instead of verbalizing intermediate reasoning steps in text, the Program of Thought prompts the LLM to output a program that encodes the solution. This design decouples reasoning as code from computation. It leverages an external interpreter who executes the generated code, enjoying the benefits of computing with LLM.

The Tree Of Thoughts [22] is an approach to prompt design that allows you to iteratively explore different ways of solving a problem. In other words, consider more scenarios before concluding with the most probable solution. At the heart of the ToT process are thought trees, representing different lines of reasoning. Such diversity enables the LLM to walk through an ample space of hypotheses resembling the human approach. When dealing with an issue, we think about many moments until we decide, for example, on our anticipations related to something. A critical part of ToT is the methodical assessment of these reasoning branches. Self-Refinement [11], inspired by how humans can review their work and initially come back with a better draft. At first, the LLM creates a response. It then solicits feedback on its output from the same LLM and continues to improve based on this collected information.

Program-Aided Language models (PAL) [5] use a program-guided solving paradigm; it uses code generation as an intermediate step for problem-solving. PAL enables LLMs to use the strength and exactness of programming languages for advanced reasoning tasks by utilizing LLMs to convert natural language challenges into code that can be executed. Executing code results in a stronger and more accurate method for performing calculations.

2.3 TOOL AUGMENTED LLMs

There is considerable interest in tool-augmented approaches. Unlike humans, LLMs can only produce words according to probabilistic models derived from outputs of extensive data sets, often with shallow reasoning or understanding of complex elaborations of the relations between concepts. This limitation has motivated researchers to investigate methods for augmenting LLMs with external tools and resources. Integrating with tools allows them to utilize information and computation outside of their trained linguistic domain. This can help them solve complex equations and access and process real-time information. Connect to knowledge bases, databases, or real-time data feeds to consume up-to-date information and have their responses grounded in reality. Use programming environments to create and run code so that they can do more complex jobs and solve an issue that needs computational steps.

ART [12] combines manually chaining prompts with outside tools to get the best answer. It is an intersection of many popular prompt engineering strategies. The process includes a sequence of systematic ART steps, given some task + input, and the system identifies the most similar tasks from its library. These tasks will be examples in the prompt to help guide and direct LLMs in approaching current work. Toolformer [13] involves training language models to leverage tool usage, inserting the new calls via unique tokens directly into keying up model training data. This enables self-supervised learning to use tools. CHAMELEON [10] proposes a more structured framework, enabling sequential multi-tool usage. The sequence of tool execution is fixed according to the particular task. The output of each tool is then fed as input to the next tool in the sequence.

Existing works that assess the behavior of tool-augmented LLMs face challenges. CHAMELEON does not evaluate performance on mathematical datasets, and ART only focuses on algebra problems. We tackle this limitation in our research by presenting a comparative analysis of the tool-augmented LLM frameworks for mathematical reasoning, evaluating multiple planning methodologies like ReACT [21] and Multi-agent collaboration.

2.4 AGENT ARCHITECTURES

LLM Agents are constructed using LLMs as central planners, allowing for natural human-like decision-making. An Agent framework comprises the Profiling, Memory, Planning, and Action modules. The profiling module distinguishes the agent's role. The memory and planning modules put the agent in a changing world that allows it to remember past actions or plan future behavior. The action module translates the agent's concrete decisions to target outputs. Within these modules, the profiling module affects the memory and planning modules, which, in conjunction, affect an action module.

Reflexion [14] is a new way of modifying language agents by adapting them to linguistic feedback rather than changing their internal parameters. Reflexion agents do not learn from weight updates. Instead, these feedback signals are reflected in a human-understandable way and stored as an episodic memory. They remember this and do not make those errors in future tasks. ReACT [21] agent proposes extending an agent's action space and including language-based "thoughts" or "reasoning traces." LLMs can partially reason and solve problems step by step like humans. But their thought process is in isolation. They do not interface with reality, so they can slip or fall into falsehood or hallucinate. ReAct solves this problem by combining reasoning and actions with feedback from tools. Plan and Execute agent architecture is inspired by plan and solve prompting technique[15] and BabyAGI Project. It is a meta-framework that helps you divide complex tasks and shape them by dividing them into small, more manageable tasks. This methodology is called planning. At the end of the execution step, we evaluate the result with the re-plan agent to determine whether it can complete the task with the given information. Otherwise, a new plan is created by re-calling the agent with a re-plan prompt. This approach is motivated by eliminating the redundant calls to the large planner LLM for each tool calling. However, this approach executes the tools sequentially, which is a limitation.

ReWOO (Reasoning Without Observation) [20] tries to optimize token usage by separating the external observations from the reasoning process. This distinction permits an extra streamlined approach to problem-solving. Language Agent Tree Search (LATS) [23] is an LLM-specific search algorithm. It uses reflection (the LLM evaluates the moves it just made) and searches with Monte Carlo Tree Search. The goal of this method is to consistently increase the overall task performance of LLMs beyond ReACT, Reflexion, and Tree of Thoughts. LLM Compiler[8] is an agent architecture where we are trying to optimize the efficiency of agentic tasks. It achieves this by using eager tasks and reducing the LLM calls. Motivated by humans' reliance on tools like search engines and code interpreters, Gou et al. introduce CRITIC [6], a framework that allows an LLM to validate its outputs. This framework enables LLMs to use tools with some back-and-forth to elicit changes and improvements, resulting in comprehensive feedback from tool execution. CRITIC uses a verify-then-correct strategy in which an external tool verifies the quality of an initial output. A key feature of CRITIC is this iterative refinement, where feedback points to issues in its production, effectively raising the overall quality. MathChat [18] is a conversational framework for solving math problems. It uses LLM Agent and User Proxy Agent to execute tools and suggest additional instructions. This design fosters a collaborative dialog model that solves complex high school competition problems from the MATH dataset. MathChat outperformed previous probe-based methods using other tools by 6%. RECONCILE [2] is a round-table conference simulation framework to obtain an improved consensus among various LLM agents. At the beginning of every round, each agent discusses the other's answer to convince them to keep their response if correct or change it otherwise. It even beats GPT-4 and a bespoke DeepSeekMath model on MATH by 8%.

The Degeneration of Thought problem stifles self-reflection, a technique that iteratively improves its solution on an LLM with its feedback. After doing so, an LLM will be too confident in its answer, and early on in reflection, it may already have difficulty getting out new ideas even if the initial claim was wrong. We propose a different framework called the Multi-Agent Debate (MAD) [9], where various agents can compete to put forward their claims, which makes it similar to playing "tit-for-tat" with each other. A judge oversees the debate process to come up with the final solution.

3 Proposed Methodology

3.1 DATASET

The GSM8K [4] dataset comprises 8,792 human-written elementary school math word problems spanning various languages. It's divided into training (7,473 problems) and testing (1,319 problems) sets. Each problem requires 2 to 8 steps to solve using basic arithmetic operations. Unlike many datasets that use mathematical expressions, GSM8K provides solutions in plain language. This makes the data easily understandable for humans and allows for broader application of our research methods. https://github.com/openai/grade-school-math

The MATH dataset [7] contains 5000 math problems evenly divided into seven subjects (Pre-calculus, Pre-algebra, Algebra, Geometry, Intermediate Algebra, Probability, and Number Theory) and five difficulty levels (1 = easiest and 5 = hardest). The MATH dataset was chosen because of its unique characteristics. One notable observation is that, unlike many datasets where larger LLMs yield better accuracy, increasing the size of LLMs does not necessarily lead to improved performance on MATH. Moreover, the dataset contains complex challenges beyond simple arithmetic or high school mathematics problems.

3.2 MODEL

This research leverages the capabilities of the GPT-4o-mini language model, accessed through OpenAI's API and Langchain.

3.3 LANGGRAPH

LangGraph is a flexible framework for building complex multi-agent systems. It allows us to create structured workflows that involve multiple AI agents working together, collaborating, and accomplishing tasks by utilizing their specializations. Some critical features of LangGraph are Multi-Agent orchestration, which enables collaboration among agents with diverse capabilities; Integration capabilities that allow easy integration with external tools, APIs, or databases; and Graph-based workflow, which crafts a graph of tasks and agent conversations. Nodes represent agents or processes, while edges specify how information flows. Reflection and feedback enable agents to self-reflect, optimize, or give feedback to improve workflows continuously and Human In the Loop facility to integrate human feedback in agentic workflows.

3.4 E2B CODE INTERPRETER

An e2b code interpreter is a part of the e2b platform, and it has tools and frameworks to interact with code in a live development environment. The e2b code interpreter is a feature that enables developers and agents to execute, modify, and analyze code in real-time, executing code in an environment where the state can persist between runs, which is helpful for experiment cycles or debugging. https://e2b.dev/

4 Experimental Setup

The section below details the experiments performed using the test set of GSM8K and MATH datasets.

Experiment 1: Baseline Performance
In this experiment, we assess GPT-4o-mini's overall performance in answering the questions from the test set of GSM8K and 700 level-5 questions from MATH datasets by prompting each question exactly once and accepting a single answer. We used a default prompt, which is derived from the few-shot prompt that is used in the MATH dataset. The prompt "Solve the problem carefully. Enclose the final answer in \\boxed{}. {Problem}", the prompting method was aimed to prompt the model towards a solution.

Experiment 2: Effect of Program-Of-Thought
In this experiment, we assessed the performance of a LangGraph agent integrated with an e2b code interpreter to execute the code in a sandboxed environment. The agent is using GPT-4o-mini and the Program-of-Thought prompt.

Experiment 3: Effect of Self-Reflection
In this experiment, we use Self-Reflection, which consists of a generator and a reflector agent. The generator tries to solve the mathematical reasoning question directly on request. The reflector is prompted to role-play as a teacher and offer constructive criticism for the initial response. The loop proceeds n times, and the final generated output is generated. (We capped the value of n to 3)

Experiment 4: Effect of Reflexion

In this experiment, we use Reflexion, an architecture designed to learn through verbal feedback and self-reflection. Within reflexion, The generator tries to solve the mathematical reasoning question by writing a program, and the reflector agent explicitly critiques each response and grounds its criticism in external data obtained by executing the program in an e2b Python code interpreter. It also provides details of missing aspects of the generated response as comments. This makes the content of the reflections more constructive and steers the generator in responding to the feedback.

Experiment 5: Effect of Multi-Agent Collaboration

A single agent is limited to poor performance on complex tasks requiring many tools across many domains. To mitigate this limitation, a multi-agent architecture has been adopted. Creating specialized agents designed for a particular task. This significantly enhances the overall efficiency and effectiveness by delegating complex tasks into multiple sub-tasks and directing them to the particular expert agent.

This experiment uses collaboration between agents, mainly Researcher and Programmer agents. The Researcher and The Programmer collaborate and converse to solve the problem. The conversation proceeds n times, and the generator generates the final output using the conversational data in memory.

5 Results and Discussion

On the GSM8k dataset, we find that the accuracy increases when agents use Python to solve the problems, and Multi-agent collaboration improves accuracy by 2% compared to the Reflexion Agent. This increase in accuracy can be attributed to the simplistic nature of the GSM8k word problems, as they mainly involve simple math.

Experiment	**Accuracy**
Baseline Performance	69%
Effect of Self-Reflection	84%
Effect of Program-Of-Thought	90%
Reflexion	91%
Multi-Agent Collaboration	93%

On the Math dataset, we evaluated six classes of level-5 problems; the agent using a Program of Thought is more effective than vanilla prompting, with about a 10% improvement in overall accuracy. The increase in accuracy is particularly pronounced in subjects requiring extensive formal numerical calculations (Counting & Probability and Number Theory) and more abstract subjects (Intermediate Algebra and Precalculus). However, the Program of Thought Agent only achieves marginal improvement or degrades accuracy for algebra and prealgebra. In comparison, Multi-agent collaboration improves total accuracy by another 6% and shows consistent performance across all categories.

Experiment	Accuracy
Baseline Performance	28.6%
Effect of Program-Of-Thought	37.6%
Multi-Agent Collaboration	44.7%

We can see that the agent performance improves significantly in solving math problems as it receives feedback by executing the generated code solution using a Python interpreter.

6 Conclusion

We conducted extensive experiments to evaluate the impact of using Python as a medium for mathematical reasoning in LLMs. Our findings reveal that using a Python interpreter as a tool and feedback from the executed code significantly outperforms the agent's performance compared to zero-shot prompting, achieving a 6% improvement on the MATH dataset. We found that the effect of tool-augmented models is more pronounced on straightforward word problems like those in GSM8k than on complex mathematical problems in the MATH dataset.

7 Future Work

Future directions include:

- Integrating tools like Wolfram Alpha, Z3, and SAT solvers for advanced logical reasoning and OMCS knowledge bases for commonsense understanding
- Studies with more advanced agent architectures.
- To unlock our system's full potential, we need to develop advanced planners capable of dynamically tailoring tool selection to each problem's unique requirements.

Acknowledgments

I would like to express my gratitude to Dr. D. Narayan, "Designation", "Affiliation", for his/her continuous guidance, assistance, and encouragement throughout the development of this Project.

I am grateful to the Project Coordinator, Rohan Gurubhaiye', Academic Counsellor, for organizing, managing and helping with the entire process.

I take this opportunity to thank Dr. Sandesh B J, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing me various opportunities and enlightenment at every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

References

[1] Sébastien Bubeck et al. "Sparks of artificial general intelligence: Early experiments with gpt-4". In: *arXiv preprint arXiv:2303.12712* (2023).

[2] Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. "Reconcile: Round-table conference improves reasoning via consensus among diverse llms". In: *arXiv preprint arXiv:2309.13007* (2023).

[3] Wenhu Chen et al. "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks". In: *arXiv preprint arXiv:2211.12588* (2022).

[4] Karl Cobbe et al. "Training verifiers to solve math word problems". In: *arXiv preprint arXiv:2110.14168* (2021).

[5] Luyu Gao et al. "Pal: Program-aided language models". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 10764–10799.

[6] Zhibin Gou et al. "Critic: Large language models can self-correct with tool-interactive critiquing". In: *arXiv preprint arXiv:2305.11738* (2023).

[7] Dan Hendrycks et al. "Measuring mathematical problem solving with the math dataset". In: *arXiv preprint arXiv:2103.03874* (2021).

[8] Sehoon Kim et al. "An LLM compiler for parallel function calling". In: *arXiv preprint arXiv:2312.04511* (2023).

[9] Tian Liang et al. "Encouraging divergent thinking in large language models through multi-agent debate". In: *arXiv preprint arXiv:2305.19118* (2023).

[10] Pan Lu et al. "Chameleon: Plug-and-play compositional reasoning with large language models". In: *Advances in Neural Information Processing Systems* 36 (2024).

[11] Aman Madaan et al. "Self-refine: Iterative refinement with self-feedback". In: *Advances in Neural Information Processing Systems* 36 (2024).

[12] Bhargavi Paranjape et al. "Art: Automatic multi-step reasoning and tool-use for large language models". In: *arXiv preprint arXiv:2303.09014* (2023).

[13] Timo Schick et al. "Toolformer: Language models can teach themselves to use tools". In: *Advances in Neural Information Processing Systems* 36 (2024).

[14] Noah Shinn et al. "Reflexion: Language agents with verbal reinforcement learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[15] Lei Wang et al. "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models". In: *arXiv preprint arXiv:2305.04091* (2023).

[16] Xuezhi Wang et al. "Self-consistency improves chain of thought reasoning in language models". In: *arXiv preprint arXiv:2203.11171* (2022).

[17] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[18] Yiran Wu et al. “MathChat: Converse to Tackle Challenging Math Problems with LLM Agents”. In: *ICLR 2024 Workshop on Large Language Model (LLM) Agents*. 2024.

[19] Zhiheng Xi et al. “The rise and potential of large language model based agents: A survey”. In: *arXiv preprint arXiv:2309.07864* (2023).

[20] Binfeng Xu et al. “Rewoo: Decoupling reasoning from observations for efficient augmented language models”. In: *arXiv preprint arXiv:2305.18323* (2023).

[21] Shunyu Yao et al. “React: Synergizing reasoning and acting in language models”. In: *arXiv preprint arXiv:2210.03629* (2022).

[22] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[23] Andy Zhou et al. “Language agent tree search unifies reasoning acting and planning in language models”. In: *arXiv preprint arXiv:2310.04406* (2023).