Zero-Shot Mathematical Problem Solving with Large Language Models via Multi-Agent Conversation Programming

Vivian Keating

GotIt! Education
vivian@gotitapp.co

Abstract

This research explores the application of conversation programming and multi-agent collaboration techniques to enhance the zero-shot, mathematical problem solving of Large Language Models (LLMs), with a specific focus on GPT-4. We compare various math solver strategies: a straightforward system prompt (Raw), a single-agent with Python access, and a multi-agent solution (Aurek), vs a single agent solution explored in prior work. Here two agents, 'Aurek' and 'Besh', work together through conversation programming and the social dynamics between them is central to achieving accurate final results. In the context of using the GPT-4 and without providing additional mathematical knowledge, we achieve a state-of-the-art 62.69% accuracy for zero-shot solution generation on a representative subset of the MATH dataset.

Introduction

Large language models (LLMs) and GPT-4 specifically are powerful for solving mathematical problems (Lightman et al. 2023; OpenAI 2023; Bubeck et al. 2023), but suffer from both arithmetical and conceptual errors. Recent advances have successfully augmented GPT-4's capabilities through prompt engineering, Python code execution (Wu et al. 2023b; Gou et al. 2023; Gao et al. 2023; Zhou et al. 2023), and chain-of-thought reasoning (Kojima et al. 2023; Wei et al. 2023).

However, in the new paradigm of *conversation programming* as proposed by (Wu et al. 2023a) in their work on AutoGen, we view Python code execution through the lens of a conversation with a REPL. This paradigm is also leveraged to create a multi-agent strategy, in this case two agents, Aurek and a debate partner Besh, to help encourage divergent thinking (Liang et al. 2023). We find that optimal performance of these methods depends heavily on the 'social dynamics' between the agents in our system.

Conversational Problem Solving

An example of a dialog between the two agents is presented here:

Problem Compute

$$\sum_{n=1}^{1000} \frac{1}{n^2 + n}.$$

Aurek: Hey Besh, I'm trying to solve the sum of $\frac{1}{n^2+n}$ from n=1 to 1000. I remember that these kinds of problems sometimes have a trick to them, like partial fraction decomposition or something. I'm thinking of starting by factoring the denominator. What do you think?

Besh: Yeah, Aurek. You are on the right track. That sounds good! We can try to factor the denominator $n^2 + n$. This gives us $n(n + 1)$. The next step would be to express the $\frac{1}{n(n+1)}$ as the difference of two fractions where one is in terms of n and the other is in terms of (n+1). Why don't you give that a shot?

Code Execution...

Aurek: I did the partial fraction decomposition and found that $\frac{1}{n(n+1)} = \frac{1}{n} - \frac{1}{n+1}$. This should help with the sum because the terms are going to cancel out a lot when I add them up, right?

Besh: Precisely, Aurek! That's the beauty of the Telescoping Series. Given that $\frac{1}{n(n+1)} = \frac{1}{n} - \frac{1}{n+1}$, when you do the sum, most of the terms will indeed cancel out. The first few terms, for example, would look like: $1 - \frac{1}{2} + \frac{1}{2} - \frac{1}{3} + \frac{1}{3} - \frac{1}{4} + \frac{1}{4} - \ldots$ Notice how most terms are getting subtracted right after being added? They 'telescope' down each time. So how can we apply this concept to the sum from n = 1 to 1000?

Code Execution...

Aurek: Besh, it looks like after applying the concept of Telescoping Series to the sum, all terms except the last term of each fraction cancel out. The sum from n=1 to 1000 simplifies to just the first term of the first fraction minus the last term of the last fraction, which is $\frac{1000}{1001}$. Does that seem right to you?

Code Execution...

Besh: "That's correct, Aurek! After simplifying the series, we just need to calculate $1 - \frac{1}{1001}$, and this gives 0.999, which is the answer to your sum. This telescoping trick can make solving sums like this a lot easier—it's a good tool to have in your math toolbox!"

Solution To compute $\sum_{n=1}^{1000} \frac{1}{n^2+n}$, we can factor the denominator as $n(n+1)$. We then use partial fraction decomposition to write $\frac{1}{n(n+1)}$ as the difference of two fractions: $\frac{1}{n} - \frac{1}{n+1}$. When we apply this to the sum, we get a telescoping series where all terms cancel except for the first term of the first fraction and the last term of the last fraction. Therefore, the sum simplifies to: $\sum_{n=1}^{1000} \left(\frac{1}{n} - \frac{1}{n+1}\right) = \frac{1}{1} - \frac{1}{1001}$. Evaluating this, we have $1 - \frac{1}{1001} = \frac{1000}{1001}$. So, $\sum_{n=1}^{1000} \frac{1}{n^2+n} = \frac{1000}{1001}$. This result has been verified by both calculation and by consulting with my friend Besh, who agrees with the solution.

Grader Great job! Your solution is correct and your use of the partial fraction decomposition method to simplify the sum was perfectly implemented. You understood that it simplifies to a telescoping series where most of the terms cancel out. Keep up the good work!

Methodology

Solver Strategies

- **Raw** - A bare-bones agent with a simple system prompt, "Solve the problem. Show all steps."
- **Python** - A single-agent solution that includes a conversation programming interface with the IPython REPL.
- **Aurek** - A multi-agent solution with two Agents named 'Aurek' and 'Besh'. Each agent has access to the IPython REPL. Aurek can ask Besh for help by passing messages, however, Aurek alone decides when to submit the final answer.

Conversation Programming

The benefits of multi-agent solutions are highly dependent on the 'social structure' of the initial system prompts. The Aurek strategy works best when the agents are encouraged to disagree and debate as they work through the solution. Our early iterations modeled the multi-agent relationship as a student and teacher, but we found that the teacher had an outsized influence on the final answer. Therefore our system prompts were modeled on a peer relationship.

We also found that GPT-4 by default is often too polite and would not adequately challenge the assumptions of the other agent. Therefore our prompts for the multi-agent solution differed slightly. Aurek was encouraged to submit a solution as a tiebreaker, while Besh was encouraged to challenge assumptions and confirm important details.

Grading Strategy

We leveraged GPT-4 to grade the correctness of the generated answers, which we believe has better accounted for equivalent mathematical representations than other methods. The grading agent has unique access to the ground truth solution. All answers are judged as best-of-one.

	Raw	Python	Aurek
GPT-4	50.41%	56.74%	**62.69%**
GPT-3.5	33.33%	34.83%	**43.65%**

Table 1: Best-of-one accuracy on a subset of the MATH (Hendrycks et al. 2021) dataset.

Results

The results from our study, shown in Table 1, indicate a clear advantage of the multi-agent solution, Aurek, over the Raw and Python strategies. Specifically, in the case of GPT-4, the accuracy of solutions was measured at 62.69% for Aurek, compared to 56.74% for Python and 50.41% for Raw. For GPT-3.5, the performance gains were even more pronounced.

Related Work

Our evaluation here used the 'gpt-4-0613' model snapshot to achieve state-of-the-art performance with that architecture in a best-of-one, zero-shot, no-retrieval augmentation environment. However we are aware that more recent OpenAI models, not widely available during our experimentation, have demonstrated even better performance. For GPT-4 'with Code Interpreter' state-of-the-art accuracy results on MATH are 84.3% (Zhou et al. 2023), a significant improvement over our findings.

In a conversation programming framework where the agent has access to a vector database with retrieval augmentation we note that an accuracy of 69.48% is possible (Wu et al. 2023a). We decided to first baseline just the multi-agent approach in conversation programming with zero-shot, no-retrieval, and in future work we will add retrieval augmentation with access to a vector database to observe the incremental performance improvement, and whether it will exceed AutoGen's 69.48% results.

Finally, 78.2% accuracy has been shown if we allow model retraining and determine the final answer as the best-of-1860 (Lightman et al. 2023). Again, in future work we will be able to observe the improvement of results achieved in multi-agent conversation programming with "best-of" and model retraining techniques applied.

Conclusion

This study illustrates the potential of multi-agent conversation programming with specific social dynamics in enhancing the mathematical problem-solving abilities of Large Language Models like GPT-4. The introduction of a multi-agent system, characterized by both collaborative and conflicting interactions between agents, is often more effective than single-agent prompt strategies.

Our findings offer additional evidence for a paradigm shift in the approach to LLMs, where conversation programming and multi-agent collaboration become central to problem-solving strategies. The success of the Aurek strategy, characterized by peer-to-peer interaction and debate, underscores the importance of social dynamics in artificial intelligence.

Acknowledgements

We would like to recognize the review of and comments to this paper by Dr. Gokhan Tur, Dr. Dilek Haqqani-Tur, and Peter Relan.

References

Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; Nori, H.; Palangi, H.; Ribeiro, M. T.; and Zhang, Y. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. PAL: Program-aided Language Models. arXiv:2211.10435.

Gou, Z.; Shao, Z.; Gong, Y.; Shen, Y.; Yang, Y.; Duan, N.; and Chen, W. 2023. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. arXiv:2305.11738.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916.

Liang, T.; He, Z.; Jiao, W.; Wang, X.; Wang, Y.; Wang, R.; Yang, Y.; Tu, Z.; and Shi, S. 2023. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. arXiv:2305.19118.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2023. Let's Verify Step by Step. arXiv:2305.20050.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; Awadallah, A. H.; White, R. W.; Burger, D.; and Wang, C. 2023a. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

Wu, Y.; Jia, F.; Zhang, S.; Li, H.; Zhu, E.; Wang, Y.; Lee, Y. T.; Peng, R.; Wu, Q.; and Wang, C. 2023b. An Empirical Study on Challenging Math Problem Solving with GPT-4. arXiv:2306.01337.

Zhou, A.; Wang, K.; Lu, Z.; Shi, W.; Luo, S.; Qin, Z.; Lu, S.; Jia, A.; Song, L.; Zhan, M.; and Li, H. 2023. Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. arXiv:2308.07921.