A Comparison of Methods Employed During AIMO 2024 Competition

Aidan Gomez Lucas Neves

Boston University

Introduction

Our goal was to build a LLM-based system capable of solving novel math word problems with high accuracy and reliability. Our methods included leveraging a variety of prompting paradigms in an attempt improve the accuracy and consistency of answers. We also pursued fine-tuning models on externally sourced math datasets containing similar samples which were roughly within the difficulty range of the AIMO competition, namely highschool level geometry, arithmetic, and algebra.

Core assumptions of problem set

We made a several assumptions about the competition hidden set (which functions as a test set) based on the competition outline and description, as well as the public training set questions:

- The hidden public LB questions are in the same distribution as the final private LB questions
- The training set questions are representative of the hidden question distribution
- The questions are primarily high-school level algebra and geometry questions
- The questions are able to be solved

Methods

We utilized a variety of different prompting strategies and system level approaches to improve the consistency and accuracy of the model. These methods included: self-reflection, wherein the AI assesses its own thinking; self-consistency, which consists of repeated prompting with slight variations, finally taking the most often generated solution; Chain-of-Thought prompting, which leads the model to break down problems step-by-step. We also employed tools like the *Sympy* library in Python for complex calculations and Retrieval-Augmented-Generation to utilize past knowledge and guide the model to generate solutions based on similar known problems. Upon cursory examination of the AIMO task description, our immediate action was to acquire novel data not yet publicly available in any AIMO-related dataset. A large quantity of high quality data has proven to be a kingmaking factor in large language model based methods, due to its use in finetuning as well as its utility for testing our solutions across a more robust distribution of samples. Ultimately the search yielded the LILA Math Reasoning dataset (Mishra et al. 2023), which features problems, answers, and corresponding Python3 scripts which produced the solution. The data was then cleaned and formatted for our use. After examining popular models and drafting a list of eligible ones, we elected to begin by finetuning a quantized version of DeepSeekMath-7B-Base on the LILA dataset. This was done for a maximum of $39,000$ timesteps with a batch size of 2, however, not only was the accuracy of the quantized finetuned model worse, it also ran at a much slower speed (see our discussion of BFloat16 below) The finetuning process consumed a great deal of time and computational resources. In an attempt to recoup, we shifted to faster methods which might increase performance, namely an agent-based approach and chain-of-thought reasoning. Furthermore, we reproduced *Deeply Understanding Problems* (Zhong et al. 2024), which were shown to aid GPT3-5 in obtaining 97% on the GSM8K, a popular arithmetic focused benchmark. Concurrently, we implemented Retrieval Augmented Generation, which entails providing the model with a similar but solved example problem. We additionally tested the performance of a number of models (see Benchmarks) against the LILA dataset to gain further understanding of their individual performances. Our final approach involved blending all of these sans finetuning in a single implementation, randomly selecting a prompting paradigm or including retrieval augmented generation on each turn, taking 17 turns per problem.

Model selection

When selecting our base model we considered a few different 7B sizes models such as Mistral, Gemma, WizardMath, and DeepSeekMath. Of these models in our initial evaluation, DeepseekMath achieved the highest scores, and so was selected as our base model. The model selection choice was corroborated by public notebooks in the competition, many of which also gravitated towards DeepSeekMath as the highest performing base model for this competition. This is likely due to DeepSeekMath publicly outscoring all contemporary models of a similar or larger size in GSM8K and MATH benchmark datasets (Shao et al. 2024) that did not vi-

olate the February 23rd cutoff date. The list of models tested, as well as their performance, can be seen in *Benchmark Outcomes*.

Prompting methods

Self-consistency

In order to improve the stability of the answers the LLM model provided, we used self-consistency with a repetition iteration count of 17. Each iteration had a uniform random chance of selecting one of four prompt variations. The highest counted repeated answer was taken as the final answer of that question. The choice of 17 iterations was selected as this was the highest iteration count that would still complete within the required runtime, and provided the most stable results. Going above this resulted in occasional timeouts, and values lower than this led to worse accuracy.

Retrieval Augmented Generation

To improve the reasoning of our system, we integrated Retrieval Augmented Generation (RAG) into our prompting system. The idea being that we can prompt the model with known math questions and step-by-step answers that are similar to the one provided to guide the model in its approach. For our RAG database we combined the Math QSA dataset, a Kaggle dataset of 5k math questions and answers with chain-of-thought solutions in algebra, pre-algebra, and geometry, along with the LILA dataset(Mishra et al. 2022), which also includes python tool-use solutions for the category of problems being evaluated in the competition. In total, our RAG dataset comprised 122k questions and answer solutions both as chain-of-thought, as well as code solutions. To perform Retrieval on the RAG dataset, we tokenized all questions using the SentenceTranformer huggingface library and the all-MiniLM-L6-v2 transformer model. these tokens and models were then fed into the FAISS similarity search library, and for each new questions received, we would query FAISS to return the top-1 most similar question-and-answer from the RAG dataset.

Prompting with Chain of thought

In the original DeepSeekMath paper, (Shao et al. 2024) the authors noted that the model was specifically structured to take advantage of Chain Of Thought (CoT) prompting, providing an example CoT that was recommended to be used:

```
{question}\n Please reason step by
step, and put your final answer
within \boxed{}.
```

Unfortunately, we found that this prompt did not perform well on the competition dataset. Using a community developed chain of thought prompt as basis, we modified it into a staging style prompt, where we ask the model to break down the problem into multiple "steps" facilitating the CoT flow and improving response stability.

Deeply Understanding the Problem

In *Achieving $> 97\%$ on GSM8K: Deeply Understanding the Problems Makes LLMs Better Solvers for Math Word Problems*, Zhong et al. (Zhong et al. 2024) claim that chain-of-thought prompting falls short when confronted with a necessity for mathematical reasoning due to a high frequency of calculation errors (i.e, $2 + 2 = 5$ etc.), non-committal approach to the following steps, and misunderstanding the problem. In turn, they propose "Deeply Understanding the Problem" prompting, abbreviated as DUP. DUP is best summarized as forcing the model to engage in a multi-stage approach to the problem. Rather than only being asked to solve a problem, the model instead iteratively tackles subproblems one-by-one. In their paper, Zhong et al. provide the question alongside three successive prompt stages;

```
1. Core Question: "Please extract
the core question, only the most
comprehensive and detailed one!"
2. Key Information: "Please extract
the question-solving information
related to the problem"
3. Generate and Extract the Answer:
"Please understand the Hint and
question information, then solve
the question step by step and show
the answer"
```

Zhong et al. implemented this technique for GPT-3.5 and applied it to six major mathematical datasets, including GSM8K. To provide points of comparison, they tested it alongside Zero-shot chain-of-thought, least-to-most, and plan-and-solve. DUP offered a mean performance improvement of 3.2 percent over zero-shot chain-of-thought, our previous methodology. DUP was modified to possess the following stages.

```
1. Summarize the problem as a list
of short subproblems,
2. Determine the information you
will need in each subproblem.,
3. Write a Python script using
Sympy for to solve the problem.
```

In this configuration, the model was called three successive times. Unfortunately upon submission it was evident that the time increase that came about as a result was incompatible with the constraints of the competition. To reduce the time taken to solve one problem, the prompt was flattened into a single call.

Self reflection

To improve the result of each individual iteration of self-consistency, we also implemented a self-reflection pass where the models answer was fed back to it and it was asked to check it over. More often than not the model would declare that the prior answer was correct, even when it was not. One tangible benefit was that even though the model would often not actually self-reflect, it did restate its solution, making easier to properly parse the LLM's answer for submission.

Dueling 'Idiots'

As mentioned, $DeepSeekMathRL$ frequently flatly refused to re-do problems. In order to encourage self-

reflection, we introduce a prompt-engineering technique referred to as 'Dueling Idiots' - a manufactured adversarial situation. In the best performing public notebook (see run 0.2), the model is instructed to explain generated Python code simply enough for an 'idiot' to understand it. Taking this as inspiration, we created a prompt in which the model was first instructed to solve the problem in a staging style approach wherein the problem is decomposed to constituent sub-problems, key information is identified, and then a solution is drafted which it would then check. The novel component is the 'duel of the idiots' - after the initial four staging phases, the prompt was changed from direct instruction to the model qua addressing it as the author of the previous stages to a 'duel', an adversarial situation in which the previously model generated text was framed as originating with the user rather than the agent.

Furthering this approach, the model was then instructed to "Correct the work, which was done by an idiot". Appending this coarse and insulting phraseology resulted in dramatic increases in the frequency of successful self-reflection.

Agent methods

Agent-style approaches wrap large language models in a schema which facilitates tool-use and environmental awareness. Langchain, a library which provides large language models with the necessary agent accoutrements, was tested alongside Deepseek-7B-Base and Deepseek-7B-Instruct. Langchain provides a convenient format that permits models to make use of a Python REPL, calculator, and custom tools that seemed ideal for the core task presented in the AIMO challenge. Unfortunately, the documentation proved to be inconsistent and addressing the requirements of the library were siphoning development time away from the core task and towards features already implemented in core notebooks, such as the use of the Python REPL. Due to this as well as less-than-impressive performance, Langchain and similar agent-based approaches were shelved in favor of working within the format already provided.

Tool use

Since LLM models are not very stable at doing basic math, we leveraged tool use, having the model instead generate code that performs the required math operations. for this we instructed the model to use python and the Sympy library, to facilitate solving symbolic math algebra problems. the runtime looks for "python" keywords, and iteratively builds out the python script to be executed. Any errors or results from the code execution is then fed back into the model, where it may either correct the code and re-attempt to execute, or continue, taking the results of the prior computation into its solution. While this approach does improve the success rate of the system, some questions are not well structured for this type of tool use, so this method was integrated and used alongside non-tool use prompting, and leveraged during self-consistency passes, where the results of both tool-use and non-tool-use prompting is compared for stable, repeated responses.

Runtime considerations

With the additions of Self-reflection and Self-consistency, our runtime-per-question increased significantly, as each question now required many iterations through the language model to arrive to a solution. This began to lead to occasional timeout problems as the total runtime began to exceed the allotted nine hour maximum. to alleviate this, we explored using datatypes that had better hardware acceleration on kaggle, namely IEEE Float16 datatype instead of the default BFloat16 format used in the model. This was selected as the Kaggle hardware (Nvidia Tesla T4x2) is too old to have native BFloat16 (BrainFloat16) support, but *does* have native FP16 matrix math acceleration support. While this change did improve runtime and eliminate the notebook timeout issues, it came at a reduction in scoring quality, so this was reverted to BFloat16. In another approach to reduce runtime, we attempted to convert our runtime code from using the huggingface Transformers library to use vLLM instead, which offers much faster inference throughput, on the order of 10x, due to being purpose built for model serving and inference. The objective of switching to the vLLM runtime would be that we would be able to increase the number of self-consistency trials executed per question and implement more processing per question. Unfortunately this proved complicated to set up as installer packages needed to be pre-downloaded and setup at notebook runtime when scoring, and lead to late-in-run out of memory errors and runtime exceptions, and so was abandoned. In the end for runtime management, the maximum token generation per question was lowered slightly to keep overall runtime below timeout threshold.

Fine Tuning

Selection of Data

In order to gain the benefit of a more diverse finetuning dataset, the LILA Unified Math Reasoning dataset was selected for use in finetuning. The LILA dataset contains questions related to arithmetic, calculus, algebra, and reasoning alongside Python3 implementations and the corresponding solutions. (Mishra et al. 2022) After manually pruning files containing irrelevant or excessively long questions, the modified dataset contained 118,033 rows with as many $\{Question, Code, Answer\}$ pairs.

Challenges

Notebook timeouts

With the self-consistency and self-reflection passes, our runtime-per-question was frequently coming up against the competition limit of 9 hours, and so finding the right balance between consistency iterations, total token length generation per question, datatype, and runtime was a time consuming iterative process, made more difficult as the real runtimes per submissions were obfuscated and we could only make two attempts per day.

Lack of Computational Resources

One challenge we encountered when fine tuning the model was limited computing resources for training. This lead to a very slow iteration cycle for fine-tuning, and ultimately led to fine-tuning being a non-viable approach for us. Our computing resources were limited to Colab A100 instances, which have no guaranteed availability and a 12 hour time cap, and so would often quit before training completed. In addition to Colab hosted systems, we also had a single 24GB VRAM local GPU that while capable of training the model, did so much slower than an A100. These challenges in fine tuning led to us being unable to properly iterate and test fine-tuned models and adjust our datasets and data preprocessing in a timely manner.

DeepSeekMath brittleness to small changes to prompting/seed/temperature

We noticed that doing simple randomness changes such as selecting a different seed or making small adjustments to the temperature would lead to vastly worse scoring performance, and that there was a large variability in run-to-run scoring, with the same code scoring anywhere from 16-20 simply by re-running with no changes. This made it somewhat challenging to determine what changes were helping vs hurting scoring, as some improved run-to-run stability, but did not necessarily achieve a "lucky" high scoring solution set. Due to the limited number of daily submissions, it was not feasible to average the scores of a notebook in order to determine mean performance. Instead, we opted for significant architecture changes each iteration as opposed to tuning hyper-parameters in an iterative fashion.

Benchmark Outcomes

Different models and methods were bench-marked on the LILA dataset. This tests RAG in the absolute optimal circumstance - assuming the test problems being asked are of extremely high relevance to the problems stored in the embedding dataset.

100 randomly sampled problems from the audited LILA dataset were fed to the models. This included a percentage (20%) questions were not answerable within the schema provided to the model (non-integer answers), but nonetheless featured lines of reasoning that were desirable for the model to be able to emulate, and thus were retained in our dataset for the sake of fine-tuning. The temperature was fixed to 0.9.

Model Name	DUP	CoT	CoT+RAG
DeepSeekMath 7B RL	24%	1%	41%
DeepSeekMath 7B Instruct	29%	7%	38%
DeepSeekMath 7B Base	-	-	9%
yunconglong 13B-MOE	1%	-	-
WizardMath 7Bv1.1	-	4%	-

Results

With these changes, our highest scoring competition result was 20/50 questions correct, for a ranking of #603 on the public leaderboard.

Conclusions

Our results indicate a limited potential for reproducibility and/or generalization in regards to several of our cited sources. The DUP prompting introduced by Zhong et al. (Zhong et al. 2024) did not reliably result in the touted performance increases when applied to the LILA dataset - at least not to the degree claimed in the paper.

CoT + RAG offered the highest performance on our test data. However, this was in the optimal scenario. In terms of generalization to entirely novel data, DeepSeekMath-7B-Instruct combined with DUP prompting scored the highest, at 29%. Generally Instruct & RL models were both more accurate and, anecdotally, faster to complete problems.

A large source of testing error came from being unable to parse the correct answer from the model-generated text, as it was often construed, formatted incorrectly, or not given in the manner 'asked' of it. Given more time and computational resources, the authors would be interested in implementing a more versatile agent with reliable tool-use, potentially fine-tuning a 13-billion parameter model.

References

Mishra, S.; Finlayson, M.; Lu, P.; Tang, L.; Welleck, S.; Baral, C.; Rajpurohit, T.; Tafjord, O.; Sabharwal, A.; Clark, P.; and Kalyan, A. 2022. Lila: A Unified Benchmark for Mathematical Reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mishra, S.; Finlayson, M.; Lu, P.; Tang, L.; Welleck, S.; Baral, C.; Rajpurohit, T.; Tafjord, O.; Sabharwal, A.; Clark, P.; and Kalyan, A. 2023. Lila: A Unified Benchmark for Mathematical Reasoning. arXiv:2210.17517.

Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y. K.; Wu, Y.; and Guo, D. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

Zhong, Q.; Wang, K.; Xu, Z.; Liu, J.; Ding, L.; Du, B.; and Tao, D. 2024. Achieving¿ 97% on GSM8K: Deeply Understanding the Problems Makes LLMs Perfect Reasoners. *arXiv preprint arXiv:2404.14963*.