MathPrompter++: Improving the reasoning capabilities of LLMs on numerical problems with robustness

Sudhansh Peddabomma
speddabomma@ucsd.edu

1 Introduction

Large Language Models (LLMs) often struggle with arithmetic reasoning tasks, frequently producing incorrect answers. Unlike natural language understanding, math problems have a definitive correct answer, making accurate solution generation a more difficult challenge for LLMs. MathPrompter addresses this problem leveraging zero-shot Chain of Thought (CoT) to generate multiple algebraic expressions or Python functions to solve the same problem in diverse ways, thereby boosting confidence in the results. This differs from conventional CoT methods, where intermediate steps remain unchecked for correctness.

Yet, this method fails to leverage the LLMs ability to reason through a problem *step-by-step*. Through this project, I propose **MathPrompter++** that embeds CoT in the steps to improve the accuracy while reducing the hallucinate rate. Furthermore, I compiled a new dataset **DiverseMath** that serves as a better benchmark for reasoning in numerical problems. The project report analyses different methods for this problem in detail and highlights the source of limitations and errors in each method for future work.

1.1 Project Status

Phase 1 (Setting Up) - Read and understood the paper. Found the limitations and suggest potential improvements to the method. Setup basic working code. **Done**

Phase 2 (Running Experiments at Scale) - Performed experiments with 4o mini. Setup Local LLM for Mac (this proved to be more challenging) to minimize costs with 4o mini. Experimented with MLX library with prompting and fine-tuning experiments. Got MathPrompter running with local LLM. **Done**

Phase 3 (Basic Improvements) - Suggested basic improvements in the code to improve the performance. Initial version of MathPrompter++ that gets better results. Fixed bugs in the code, and set up final working versions of the code with complete refactoring of the codebase. **Done**

Phase 4 (New Dataset Preparation) - Compiled and filtered the **DiverseMath** dataset. Rechecked all solutions with the help of LLMs and manually solved many of them. **Done**

Phase 5 (Final Drafting) - Obtaining significant improvements with MathPrompter++ using both 4o mini (done) and Local Llama model (did not work well). **Partially Done**

I have ommitted many results and other analysis from my work to account for the page limit of the report.

Github - Sudhansh6/MathPrompterpp

2 Related work

LLMs have remarkable performance in single-step system-1 (Stanovich and West, 2000) tasks with task-specific few-shot or zero-shot prompting (Liu et al., 2021). However, LLMs struggle with system-2 tasks that involve slow and multi-step reasoning (Rae et al., 2022). The scaling laws in the training and model size do not help with the performance in such tasks.

Addressing these issues, (Wang et al., 2023) and (Wei et al., 2023) introduce inference time reasoning through **Chain of Thought (CoT)** methodology. The method essentially fed LLMs with the step-by-step reasoning examples rather than standard question and answer examples. Such chain of

thought demonstrations facilitate models to generate a reasoning path that decomposes the complex reasoning into multiple easier steps. Notably with CoT, the reasoning performance then satisfies the scaling laws better and jumps up with the size of the language models. For example, when combined with the 540B parameter PaLM model [Chowdhery et al., 2022], chain of thought prompting significantly increases the performance over standard few-shot prompting across several benchmark reasoning tasks, e.g., GSM8K (17.9% →58.1%).

Then (Kojima et al., 2023) showed that LLMs have inherent step-by-step reasoning capabilities and introduced the paradigm of **zero-shot CoT**. They showed that simply adding a prompt similar to "Let's think step by step" replicates the CoT reasoning and scales well for system-2 tasks. It showed substantial improvement in mathematical reasoning with an improvement from 17.7% to 78.7% on the (MultiArith, 2023) dataset. It opened the potential of expanding the use of LLMs in reasoning tasks in different domains to work with deterministic problems.

Building on top of this zero-shot CoT framework, **MathPrompter** (Imani et al., 2023) extended the notion of structured prompting to guide the reasoning in LLMs. Particularly, the works addressed validation and confidence issues in typical CoT workflows. Problems similar to math reasoning require reliable answers inspite of the probabilistic generative nature of LLMs. a technique that improves performance of LLMs on arithmetic problems along with increased reliance in the predictions. MathPrompter uses the Zero-shot chain-of-thought prompting technique to generate multiple Algebraic expressions or Python functions to solve the same math problem in different ways and thereby raise the confidence level in the output results. This is in contrast to other prompt based CoT methods, where there is no check on the validity of the intermediate steps followed. Our technique improves over state-of-the-art on the MultiArith dataset (78.7% → 92.5%) evaluated using 175B parameter GPT-based LLM.

However, there are some compromises with this technique - although it leverages the impressive zero-shot capabilities of LLMs, it does not really use the CoT methodology. As a result, it struggles with some complicated problems. Furthermore, it enforces validity through strict checking in the process. Consequently, with smaller models that are not state-of-the-art and have high hallucination rate, the method is unable to output an answer to the question.

Through this work, I aim to critically test these methods and improve their performance by considering the limitations and considerations from the previous methods.

3 Datasets

The MathPrompter paper mainly contrasted the performance of the models in the **MultiArith** dataset. The dataset comprises of 600 word problems that are primary-school level and have a single numeric (integer) answer. The problems are simple - all the numbers are written as arabic numerals and there are rarely units involved. An example is shown below -

> **Q.** A florist had 5 roses. If she sold 3 of them and then later picked 34 more, how many roses would she have?
> **A.** 36

In addition, I also use the **SVAMP** (Patel, 2024) dataset which has been used as a part of the open-source implementation of MathPrompter (Kaspar, 2024). The dataset is similar to MultiArith but it addresses some issues by increasing the complexity of the problems and increasing the variety of the problems.

However, there are some significant limitations with the above datasets. As mentioned before, there are no units involved, the final answers are integers and the problems are simple requiring only one or two basic operations to arrive at the final answer. To address these issues, I composed a new dataset called **DiverseMath** that consists of 85+ numerical problems from different fields of science including Chemistry, Physics, Biology, Medicine and Economics -

- The reasoning steps are inter-twined with world knowledge, formulae and other facts to arrive at the answer. Such coupling tests the recalling and reasoning capabilities of LLMs together. The power of Chain of Thought is much more emphasized in such paradigms where the solution requires multiple steps of reasoning.

- The problems involve many units (as is the

nature of these problems) that tests the robustness of the methods with many numbers involved. As a result, the solutions require multiple steps, including converting between units, to arrive at the final answer which is often a floating point.

The difficulty of the problems in the dataset arises from the many numbers and constants involved in each problem, floating point values, and world knowledge required to solve the problems. Apart from these aspects, the mathematical operations involved are the same as the previous datasets. In such manner, the dataset tests the "reasoning" capabilities more thoroughly than the other datasets. An example from the dataset is shown below -

Q. A gas occupies 2 L at 1 atm pressure. If the pressure is increased to 3 atm at constant temperature, What is the new volume in liters?
A. 0.67

3.1 Data preprocessing and annotation

The MultiArith and SVAMP datasets have a simple structure and are already provided as a JSON format that can be directly used to test the different methods. These datasets do not require any annotation since the answers for the math problems are already given as part of the dataset without any noise. We use these fields directly to evaluate our models.

For the DiverseMath dataset, I generated some problems myself and used various Generative AI tools (Perplexity, ChatGPT 4o mini, Anthropic Claude Haiku) to generate more questions and their answers. After generating a rough dataset, I verified the solutions with other models (ChatGPT 4o mini, Claude Haiku) to flag the ones that seem to have suspicious answers. Using this data, I verified the solutions of the problems myself and filtered the dataset to have questions which require only basic operations (addition, subtraction, multiplication, division, exponentiation) for the solution. As a result of this process, starting from a dataset roughly of size 150 problems, the final dataset has 87 problems.

4 Baselines

As mentioned in the previous works section, I work with two baselines:

1. Zero-shot CoT from (Kojima et al., 2023)
2. MathPrompter from (Imani et al., 2023)

In the original paper of Zero-shot CoT, the authors had used a 175B parameter model (InstructGPT - text-da-vinci-002) for their analysis. They obtained 78.7% accuracy on MultiArith (MultiArith, 2023) and 62.1% accuracy on SVAMP (Patel, 2024) dataset. In my experiments, I tested the latest **ChatGPT 4o-mini** model (estimated to have 8B parameters but much better performance due to RLHF training), and it performed better on the datasets. The results are summarized in Table 1.

LLM	MultiArith	SVAMP
DaVinci-002	78.7	62.1
GPT-4o-mini	**96**	**84**

Table 1: Accuracy (percentage) comparison of LLMs on MultiArith and SVAMP benchmarks with (Kojima et al., 2023). *Note.* The results for 4o-mini are summarized from 100 randomized instance from the datasets.

We note that since GPT 4o-mini has much better zero-shot capabilities, this method works really well for the datasets.

The other baseline I have used for my analysis is MathPrompter (Imani et al., 2023). Since there is no official implementation of the technique, I implemented the method by modifying the open-source implementation (Kaspar, 2024). The performance is slightly lower than the results claimed in the paper. The decrement could be due to differences from the official implementation. In the paper, they have used the 175B parameter DaVinci model, and we again use the GPT 4o-mini in our experiments.

The statistical significance step in MathPrompter is skipped in the experiments. In the initial tests, I did not see much performance gain through this step. It mainly helps in reducing the hallucination rate, but the gains are negligible for our purposes considering the compute constraints.

The *temperature* is set to 0.0 for consistency throughout the experiments, and the *maximum output size for tokens* is set to 200 which is ample enough for all the methods to work correctly. However, it must be noted that CoT based methods use much higher number of tokens in output as compared to MathPrompter since the latter simply

generates algebraic expressions and Python code. These representations are succinct and capture the similar amount of information as the CoT based steps.

4.1 Hallucination

Since the zero-shot CoT methods do not have any validation steps, they often tend to hallucinate. The answer generated could be wrong due to multiple reasons - the steps of reasoning are wrong, the calculations in the steps may be wrong, or there might be a missing step. On the other hand, MathPrompter enforces strict checks through the MathPrompts steps to ensure that the answer is consistent across multiple methods. Since each method in the math prompting step is an *independent way of thought* for the solution, there is a high probability that the methods do not align with one another. As a result, in many cases, the method does not yield a final answer due to lack of consistency across the methods. In MathPrompter++, we propose a rough solution step to provide an alignment that narrows down the reasoning space for the algebraic and code generation methods and improve the consistency with one another. However, it must be noted that the if the alignment provided by the rough solution itself is wrong, then it can result in a high hallucination rate. In summary, MathPrompter++ trades off the strict consistency requirements of MathPrompter to introduce Chain of Thought thinking to better tackle complex problems.

5 Approach

Through this work, I have identified the limitations of MathPrompter particularly in regards to robustness, addressing complexities and deployment on much smaller models locally. As my contribution, I am proposing a modified method, **MathPrompter++**, that has the following optimizations

- **Streamlined code** that has been completely re-factorized to reduce the computational complexity and unnecessary initializations
- More robust prompting methods with diverse few-shot examples and rules to guide the LLM
- **Embedded Chain of Thought reasoning** - A rough-solution generation step before the math-prompts to allow LLM to think about the problem *step-by-step*.

5.1 Contributions

In the code, the scripts corresponding to my methods and the baselines are present in the `FinalCode` folder. The initial code has been forked from the opensource MathPrompter implementation (Kaspar, 2024). The implementation of MathPrompter++ is present in `methods/MathPrompterpp.py`. My contribution to the code base in addition to the changes mentioned above are:

- Integration of Local LLM to run on Mac architectures using `mlx` library
- A new dataset comprising of 85+ mathematical reasoning questions from different fields of science including Chemistry, Physics, Biology, Medicine and Economics. The dataset can be found at the file `FinalCode/data/domain.json`
- Restructuring of the code to provide better insights across the methods
- Addition of Zero-shot CoT method
- Integrating MathPrompter++ within the codebase

The `.ipynb` files run the evaulation of all the methods on the datasets.

5.2 Method

As we have seen in the Section 4, LLMs have impressive zero-shot abilities in terms of thinking about a given problem step-by-step. In MathPrompter, although the method takes advantage of zero-shot capabilities of the models, it does not approach the problem at hand in a step-by-step manner. If the problem is complex, then approaching the problem in a step-by-step manner helps break down the complexity into approachable simple steps. Writing a Python code for a problem implicitly involves some Chain of Thought reasoning, however, having an explicit CoT step can significantly reduce the hallucinations. Such approach is also seen amongst students, wherein they form a rough solution on how to approach the problem before they actually start writing down the solution.

With this motivation, we introduce a rough solution step in the MathPrompter++ method - we prompt the LLM to draft out a series of steps that

outline how to solve the given problem. Furthermore, this step also jots down any known mathematical shortcuts, formulae and results that can help solve the problem. It mainly helps manifest the logic to be used in the problem.

This way, we have embedded CoT Reasoning in MathPrompter. Once we have this rough solution, we use the other steps - Algebraic template, Math Prompts, and Compute Verification to obtain the final solution. However, in comparison to MathPrompter, we use improved prompting techniques to help with the robustness of the solution.

These improved prompts benefit the later stages to extract the answer. For example, for the same question, we get the following algebraic expressions and Python code from both the methods. An example flow for MathPrompter++ building on the previous example is shown in the Figure

In summary, with the modified approach, the hypothesis is that the method can make use of CoT capabilities of the LLMs and use the techniques from MathPrompter to ensure validity and confidence in the solution. As a result, the hallucination rate should decrease while increasing the accuracy of the method.

6 Experiments

We compare the accuracies and hallucination rate for the three methods (Zero-shot CoT, MathPrompter, MathPrompter++) on the MultiArith (MultiArith, 2023), SVAMP (Patel, 2024) and the DiverseMath datasets.

The underlying model is ChatGPT 4o-mini, which is a paid API service. To evaluate 100 questions in the dataset, the runtime for the three methods is approximately 25 minutes based on the network limitations and rate limits with the API.

The results for the three methods are summarized in Table 2

6.1 Observations

6.1.1 Accuracy

Both MathPrompter and MathPrompter++ achieve higher accuracy than zero-shot CoT on the DiverseMath dataset. This result shows that the validity of solutions generated by these methods is much higher. As a consequence, the hallucination rates are also much lower as discussed in the next section.

However, the performance of MathPrompter is not as high as claimed by (Imani et al., 2023). As mentioned before, this discrepancy could have arose due to differences from the official implementation. As a result, the MathPrompter based methods do not perform as well as the zero-shot CoT method as should have been the case. Nonetheless, in many cases, MathPrompter++ is close to the zero-shot CoT baseline or surpasses it. Furthermore, although the numbers are representative of the dataset, the results may include noise since the metrics are only computed over 100 instances from the datasets. Nevertheless, we can make inferences about the trends from these results.

In all the zero-shot settings MathPrompter++ performs better than MathPrompter. The performance gain is significantly higher for the DiverseMath dataset. On the other hand, in the few-shot settings, although MathPrompter++ achives significantly higher performance than MathPrompter in MultiArith and SVAMP dataset, it has a lower accuracy on the DiverseMath dataset. The reason for this is that the few-shot prompts for the rough solution generated in the MathPrompter++ method can bias the model to *think* in a specific manner that can often mislead and result in wrong solutions. In the zero-shot cases the model can arrive at the solution without any biases presented by the template prompts. Better template prompts can improve the performance of these methods in few-shot settings as well.

The Figure 1 shows in-depth comparison of the accuracies in the steps involved in each method.

6.1.2 Hallucinations

The results for the hallucination rates in the models is shown in the Figure 2

The hallucination rate for the Chain of Thought method is significantly high. In particular, it is very high in the DiverseMath dataset. It is expected since the questions in this dataset have many constants and numbers involved along with different domain-specific results. Since there are no validation checks, the results can often be wrong. The hallucination for the MathPrompter method for the DiverseMath dataset is also very high, and MathPrompter++ brings it down by a significant amount.

As theorized, MathPrompter++ significantly lowers the hallucination in all the zero-shot settings. However, as seen in the accuracy case, the hallucination for the few-shot setting decreases with MultiArith but increases with SVAMP and

Method	Zero-shot CoT	MathPrompter (Zero-shot)	MathPrompter++ (Zero-shot)	MathPrompter (Few-shot)	MathPrompter++ (Few-shot)
MultiArith	96	86	**89**	90	**98**
SVAMP	84	79	**83**	71	**82**
DiverseMath	49.41	60.00	**62.35**	**69.41**	65.88

Table 2: Accuracy comparison of various methods across different mathematical reasoning benchmarks in both zero-shot and few-shot settings using GPT 4o-mini.

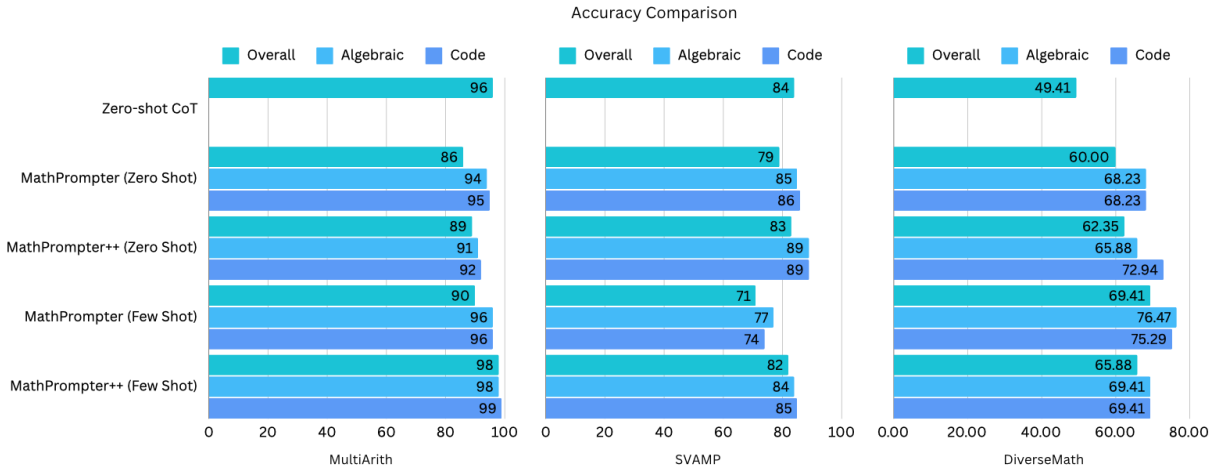

Figure 1: The figure represents the accuracies of different methods across all the datasets. The accuracies of the algebraic and coding methods are also shown in the figure. Higher is better.

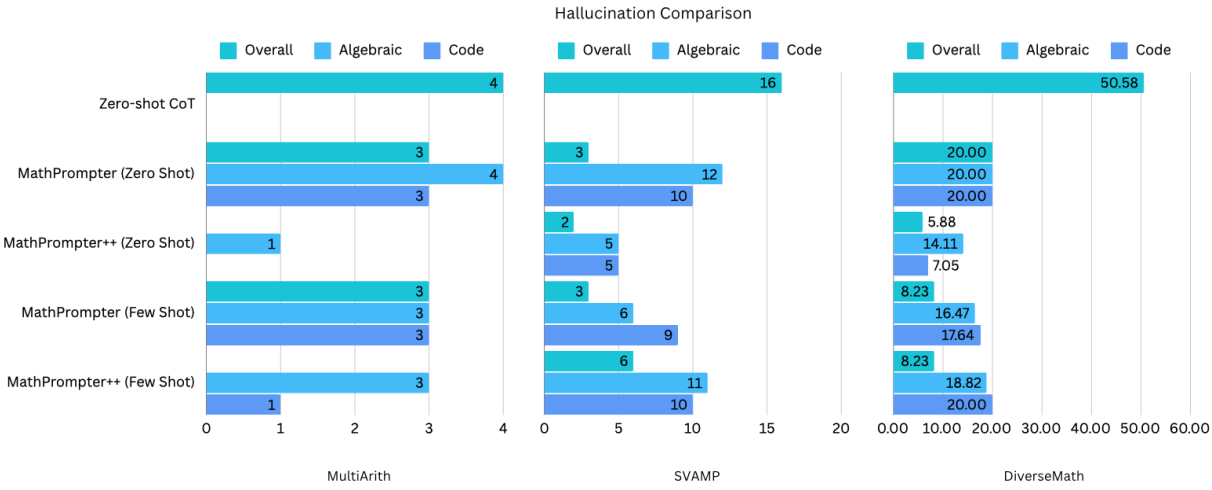

Figure 2: The figure represents the accuracies of different methods across all the datasets. The accuracies of the algebraic and coding methods are also shown in the figure. Lower is better. Chain of Thought has significantly high hallucination rates. MathPrompter++ brings down the hallucination in many cases as compared to MathPrompter.

DiverseMath datasets. This could be due to the bias introduced by the rough solution that leads the model to the wrong answer.

6.2 Error analysis

Table 3 highlights the major source of errors for each model. We show the results for the zero-shot based MathPrompter methods.

As highlighted in the previous sections, the major source of error in Chain of Thought is from the lack of validation in the steps. In some cases, CoT methods exceed the number of tokens which causes issues in the answer extraction steps. The answer extraction for the zero-shot CoT method is done using a regular expression which is prone to errors.

Question	Zeroshot CoT	MathPrompter	MathPrompter++

Q. A protein biomarker concentration is 25 ng/mL in diseased state. What is its concentration in μg/L?
A. 25000
	To convert the concentration from ng/mL to μ g/L, we can follow these steps:
	1. Understand units:
	- 1 ng (nanogram) = 0.001 μ g (microgram)
	- 1 mL (milliliter) = 1 L (liter) = 1000 mL
	⋮
	Tokens exhausted before next prompt
	25 ng/mL is equivalent to 25 μg/L. ✗
		Inconsistency between methods
		Variable Mapping: {var_A: 25}
		`# Algebraic answer`
		`Answer = var_A*1000`
		Returns 25000 ✓
		`# Python code`
		`def solution(var_A):`
		`return var_A * 1.0`
		Returns 25 ✗
			Wrong rough solution
			Rough Solution.
			1. Identify the conversion factor between ng/mL and μg/L. Recognize that 1 ng/mL is equivalent to 1 μg/L since both are measures of concentration in different units.
			⋮
			Returns 25 ✗

Q. Danny collects bottle caps and wrappers. He found 66 wrappers and 39 bottle caps at the park. Now he has 16 bottle caps and 68 wrappers in his collection. How many wrappers did Danny have at first?
A. 2
	Answer Extraction error
	To find out how many wrappers Danny had at first, we can break down the information given in the problem.
	...
	To find out how many wrappers Danny had at first, we can set up the equation based on the information provided.
	...
	Answer Extracted: 1 ✗
		Inconsistency between methods
		Variable Mapping: {var_A: 66, var_D: 68}
		`# Algebraic answer`
		`Answer = var_D - var_A`
		Returns 2 ✓
		`# Python code`
		`def solution(var_A, var_D):`
		`return var_A + var_D`
		Returns 134 ✗
			Inconsistency between methods
			Variable Mapping: {var_A: 66, var_C: 16, var_D: 68}
			`# Algebraic answer`
			`Answer = var_D - var_A`
			Returns 2 ✓
			`# Python code`
			`def solution(var_A, var_C, var_D):`
			`return var_A + var_C - var_D`
			Returns 18 ✗

Table 3: Major Errors in each of the methods on zero-shot prompting with ChatGPT 4o mini

In MathPrompter, the major source of error is the discrepancy between the methods used. When the methods don't match, this method does not yield a final value. This is a beneficial thing since it reduces the hallucination substantially by attacking the problem from multiple perspectives.

MathPrompter++ hallucinates when the rough solution itself is in the wrong direction. We have seen this happen in the few-shot instances and also from the examples shown in the table. The rough solution step should improve the accuracy but it may increase the hallucinations if the underlying model is not very good. In theory, if the underlying model is good, then the hallucination will decrease since there is a reasoning step added to arrive at the final answer. Nevertheless, sometimes the rough solution causes errors to arise in this method.

In some cases, the errors in the MathPrompter methods arise due to rounding errors - algebraic expressions round to integer whereas the Python code uses floor operation. These can be prevented with better coding solutions.

7 Conclusion

MathPrompter is an intuitive method that draws on the techniques used by students when they approach a math problem. However, it does not leverage the CoT capabilities of the underlying models. With MathPrompter++, I have shown that this is a potential direction of research. The performance of these methods is very poor when a smaller LLM is used underneath. For instance, I obtained below 10% accuracy on the datasets with Llama 3.2 1B 4-bit model on all the datasets. In future, these methods can be improved to achieve

higher accuracies with lower hallucination rates with smaller fine-tuned models.

In my implementation, I could not get MathPrompter achieve the same level of accuracies as claimed in the paper, and this proved to be very challenging. Furthermore, the improvements with the techniques I tried were marginal and they did not consistently improve the results. I got the idea of compiling a new dataset towards the end, and this experiment provided valuable insights and understanding of the methods. I performed another set of experiments using Llama 3.2 1B 4-bit, a quantized efficient model that can be run locally on a Macbook with M1 Pro 16Gb unified memory. These results can be improved in the future.

Overall, the project has provided valuable insights into how LLMs work, robust prompting methods and working with deterministic problems using these non-deterministic generative tools.

8 Acknowledgements

As mentioned previously, I used ChatGPT, Anthropic Claude and Perplexity to help generate the DiverseMath dataset. I used them to aid with the verification of the data as well. In the report, I used these tools to help generate initial templates for figures, tables and other LaTeXspecific syntax.

References

Imani, S., Du, L., and Shrivastava, H. (2023). Mathprompter: Mathematical reasoning using large language models.

Kaspar, R. (2024). Mathprompter. https://github.com/RamonKaspar/MathPrompter. Accessed on December 05, 2024.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2023). Large language models are zero-shot reasoners.

Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., and Chen, W. (2021). What makes good in-context examples for gpt-3?

MultiArith (2023). Multiarith dataset. https://huggingface.co/datasets/ChilleD/MultiArith. Accessed on December 05, 2024.

Patel, A. (2024). Svamp. https://github.com/arkilpatel/SVAMP. Accessed on December 05, 2024.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., van den Driessche, G., Hendricks, L. A., Rauh, M., Huang, P.-S., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X. L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J.-B., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T., Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., de Masson d'Autume, C., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., de Las Casas, D., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K., and Irving, G. (2022). Scaling language models: Methods, analysis insights from training gopher.

Stanovich and West (2000). Record on psycnet. *PsycNET*. Accessed on December 05, 2024.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models.

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.