CMU Spring 2024 11-711: Assignment 4 - Final Project

Ansh Khandelwal **Shrey Madeyanda** **Rajeev Veeraraghavan**
{anshk, smadeyan, rveerara}@andrew.cmu.edu

1 Introduction

The rapid advancement in Large Language Models (LLMs) has catalyzed a shift across various domains, from natural language understanding to creative content generation (Naveed et al., 2024). These models, trained on vast corpora of text data, have demonstrated remarkable proficiency in linguistic tasks, pushing the boundaries of what artificial intelligence can achieve with language. However, despite these strides, LLMs continue to face challenges in specific areas, notably in numerical reasoning and evaluation (Srivatsa and Kochmar, 2024).

Numerical evaluation in LLMs is not merely a function of understanding numbers in isolation but involves grasping the context in which these numbers are used. This includes performing arithmetic operations, understanding statistical data, making numerical comparisons in regular text to extract relevant data, and solving mathematical problems embedded in text. The existing challenges in this domain range from the inherent limitations of LLMs' architecture, which is optimized for text processing rather than numerical computation, to the quality and diversity of numerical data present in the training data.

This report focuses on the technical challenges and innovations in improving numerical reasoning abilities of open-source LLMs, like Llama. It examines advanced training techniques and architectural modifications, and then proposes a path forward to augment such LLMs with robust quantitative evaluation skills.

2 Literature Review

We survey the development and application of multiple methodologies that can be used to improve the numerical evaluation of open source large language models.

Chain-of-Thought Prompting (Wei et al., 2023) has become widely adopted method of improving the reasoning abilities of large language models, and has inspired several of the approaches being discussed in this survey, even if not directly used. It significantly improves the capability of LLMs to perform complex reasoning tasks by guiding them to generate a series of intermediate reasoning steps before arriving at a final answer, achieving a higher performance on tasks requiring arithmetic, commonsense, and symbolic reasoning. CoT prompting involves providing LLMs with input-output examples that include intermediate reasoning steps, emulating a human-like reasoning process. It provides interpretable window into the model's reasoning process, and its benefits are more pronounced with increasing model size, indicating that this capability emerges as models scale.

There are some limitations to this method, such as the potential for incorrect reasoning paths, which is compounded by most LLMs being unable to indicate their confidence in their generated outputs.

SELF-REFINE (Madaan et al., 2023) uses a method of self-feedback and iterative refinement to enhance the output of large language models. This approach was inspired by the human process of iterative self-improvement, and enables LLMs to refine their initial outputs through successive rounds of feedback and modification, without the need for additional supervised training data, further model training, or reinforcement learning techniques. The framework was evaluated across 7 tasks, ranging from dialog response generation to mathematical reasoning.

The SELF-REFINE framework operates by generating an initial output from an input prompt, using the same LLM to provide feedback on this output, and then refining the output based on this feedback. This process is iterated until an acceptable output is obtained or a predefined condition is

met. It also relies on few-shot prompting to guide the LLM in each step of the process, requiring no additional training or external data, thus demonstrating an efficient and cost-effective method for output enhancement.

The authors show that SELF-REFINE significantly improves task performance across all evaluated tasks, with improvements of approximately 20% absolute on average when compared to conventional one-step generation methods. However, there is potential for diminishing returns with additional iterations and challenges in tasks where errors are less easily identifiable through feedback.

NumeroLogic (Schwartz et al., 2024) proposes a simple adjustment to the textual representation of numbers to enhance the numerical reasoning capabilities of large language models (LLMs). The key idea is to prefix each number with the count of its digits, enabling the model to know the place value of digits before reaching the end of the number during the left-to-right processing.

This method serves two purposes: 1) it aids the model's comprehension of numerical values in the input by providing place value information upfront, and 2) it prompts the model to reason about the expected number of digits when generating numbers, acting as a form of Chain of Thought reasoning.

A notable strength of NumeroLogic is its simplicity and general applicability. It can be easily implemented as a text pre-processing and post-processing step, without requiring any modifications to the model architecture. The authors present NumeroLogic as a powerful yet straightforward technique to enhance the numerical capabilities of LLMs across a wide range of tasks and domains.

Self-Verify (Weng et al., 2022) propose a novel method that enables large language models (LLMs) to self-verify the conclusions they generate, allowing them to select the most accurate result. Their approach involves two key steps: 1) Forward Reasoning where the LLM generates multiple candidate answers using chain of thought (CoT) prompting, and 2) Backward Verification where each candidate is scored by the LLM's ability to predict the original problem conditions when using that candidate answer as an additional constraint.

The authors hypothesize that LLMs possess an innate self-verification capability akin to how humans double-check their reasoning steps. By masking certain conditions from the original problem and asking the LLM to predict those masked conditions given the candidate conclusion, they can assess how consistent each candidate is with the known facts. The candidate with the highest "verification score" is selected as the final prediction.

Across 8 datasets spanning arithmetic, commonsense, and logical reasoning tasks, the authors demonstrate that self-verification significantly improves accuracy over the baseline chain-of-thought approach, establishing new state-of-the-art results on 6 of the datasets. They find larger gains on arithmetic tasks, which allow well-defined condition masking compared to other reasoning tasks. Self-verification also combines effectively with techniques like self-consistency decoding that improve the initial forward reasoning stage.

AutoCoT (Zhang et al., 2022) proposes a novel method called Auto-CoT which consists of two main stages: question clustering and demonstration sampling. In the question clustering stage, the questions are partitioned into a small number of clusters based on their semantic similarity. In the demonstration sampling stage, a representative question from each cluster is selected, and its reasoning chain is generated using Zero-Shot-CoT (a simple prompt like "Let's think step by step") with the help of heuristics to encourage simpler questions and rationales. The authors evaluate Auto-CoT on ten benchmark datasets from three categories of reasoning tasks: arithmetic reasoning, commonsense reasoning, and symbolic reasoning. The experimental results show that Auto-CoT performs competitively compared to Manual-CoT, which requires manual designs of demonstrations. This suggests that LLMs are inherently capable of solving multi-step reasoning problems and only need to be prompted to do so.

3 Baselines

For this project, we aim to develop a pipeline that can improve the numerical reasoning capabilities of any large language model (LLM). We have selected LLama 2 7B as our base LLM to demonstrate the efficacy of our proposed pipeline.

LLama 2 7B was chosen as the baseline model for several reasons:

1. Performance on Numerical Reasoning Tasks: LLama 2 7B exhibits poor performance on numerical reasoning tasks, such as those in the MATH and GSM8K datasets. This provides a clear opportunity to showcase the improvements offered by our pipeline.

2. Model Size: With 7 billion parameters, LLama 2 7B is a reasonably large model, allowing us to test our pipeline's scalability and effectiveness on models of practical size.

3. Availability: LLama 2 7B is a publicly available model, ensuring reproducibility and accessibility for our experiments.

To establish comprehensive baselines, we will consider three approaches:

1. Zero-shot Prompting with LLama 2 7B: This baseline represents the out-of-the-box performance of the model without any task-specific fine-tuning or prompting.

2. 8-shot Prompting with LLama 2 7B: Following the approach used by Meta AI for their GSM8K baselines, we will employ 8-shot prompting with standard GSM8K questions as prompts. This baseline represents a few-shot learning setting with a fixed set of prompts.

3. Fine-tuned LLama 2 7B on GSM8K: We fine-tune LLama 2 7B on the GSM8K dataset, a widely-used benchmark for numerical reasoning in LLMs. This baseline will demonstrate the potential gains from task-specific fine-tuning.

By considering these three baselines, we aim to establish a comprehensive understanding of LLama 2 7B's performance on numerical reasoning tasks and provide a robust foundation for evaluating the improvements offered by our proposed pipeline.

The choice of baselines is motivated by our goal of developing a pipeline that can be applied to any LLM, regardless of its initial performance or whether it has undergone task-specific fine-tuning. By demonstrating improvements over these diverse baselines, we can validate the pipeline's generality and effectiveness in enhancing numerical reasoning capabilities across a wide range of LLM configurations.

4 Methodology

We propose a three-fold approach to enhance the numerical understanding of large language models (LLMs):

Number Encodings Traditional LLMs process numbers in a left-to-right manner, making it challenging to comprehend the place value of digits until the entire number is processed. This can lead to inefficiencies in arithmetic operations and numerical reasoning tasks. To address this issue, we introduce a novel encoding scheme that expands upon the one proposed in (Schwartz et al., 2024). In addition to appending the count of digits before each number, we also map the digit counts and numerical value with the textual representation of numbers (e.g., "3:593" instead of "593", "3:593:five ninety three" instead of just "five ninety three"). This encoding enhances the LLM's comprehension of numerical structure by providing place value information upfront, along with the numerical representation, facilitating more efficient processing of numerical data. We fine-tune the LLM with the encoded training data, allowing it to adapt to this new representation. We will also experiment with different formats for the encodings, and observe the differences in performance. This step will involve pre-processing the dataset we will be using for finetuning, which in this case will be the GSM8k dataset. We will discuss the different formats/techniques tried in Secion 5. It will also require a simple post-processing steps to remove the tags that are added to the answer by the LLM after finetuning. The examples we will show in this report will not include this post-processing, for demonstrative purposes.

Retrieval-Augmented Few-Shot Prompting (RAFSP) While standard few-shot prompting with a fixed set of examples can be effective, our intuition is that having prompts more closely related to the input question can further improve performance. To this end, we employ a retrieval-based few-shot prompting strategy. We construct a knowledge base comprising questions, associated chains of thought, and corresponding answers from the training sets of GSM8K (Karl Cobbe, 2021), SVAMP (Patel et al., 2021), and ASDIV(Miao et al., 2020) datasets. This knowledge base, also known as the prompt storage (Figure 1), serves as a repository of relevant examples.

During inference, we retrieve relevant question-answer pairs from this knowledge base based on the input question. We leverage UAE-Large-V1 embeddings (Li and Li, 2023) to encode all the question-answer pairs and store them in the prompt storage. Our strategy combines Information Retrieval and a mixture of Retrieve-Q-CoT and Manual-CoT approaches. Specifically, we retrieve the top 4 most similar question-answer pairs based

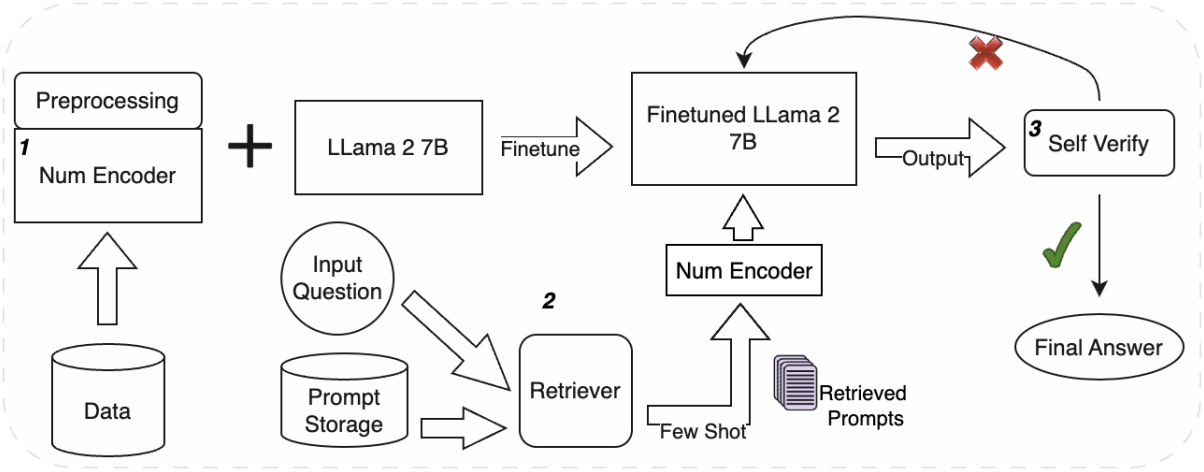

Figure 1: Proposed Pipeline

{"question": "A water tower that serves <se>1<de>4<ne>four<fe> neighborhoods around it holds <se>4<de>1200<fe> barrels of water and is filled to the top each week. If <se>1<de>1<ne>one<fe> neighborhood uses <se>3<de>150<fe> barrels of water in a week, the second neighborhood uses twice as many barrels of water as the first neighborhood in a week, and the third neighborhood uses <se>3<de>100<ne>one hundred<fe> more barrels of water than the second neighborhood in a week, how many barrels are left for the fourth neighborhood?", "answer": "The second neighborhood uses <se>3<de>150<fe> * <se>1<de>2<fe> = <<<se>3<de>150<fe>*<se>1<de>2<fe>=<se>3<de>300<fe>>>><se>3<de>300<fe> barrels of water.\nThe third neighborhood uses <se>3<de>300<fe> + <se>3<de>100<fe> = <<<se>3<de>300<fe>+<se>3<de>100<fe>=<se>3<de>400<fe>>>><se>3<de>400<fe> barrels of water.\nThe first, second, and third neighborhoods together use <se>3<de>150<fe> + <se>3<de>300<fe> + <se>3<de>400<fe> = <<<se>3<de>150<fe>+<se>3<de>300<fe>+<se>3<de>400<fe>=<se>3<de>850<fe>>>><se>3<de>850<fe> barrels of water.\nThus, the fourth neighborhood has <se>4<de>1200<fe> - <se>3<de>850<fe> = <<<se>4<de>1200<fe>-<se>3<de>850<fe>=<se>3<de>350<fe>>>><se>3<de>350<fe> barrels of water left for it to use.\n#### <se>3<de>350<fe>"}

Figure 2: Example of encoded finetuning training instance

on cosine similarity between their embeddings and the input question's embedding.

These retrieved examples are then used as prompts to guide the Large Language Model's (LLM) reasoning process. By providing relevant context and examples, this approach should minimize the risk of hallucinations and enhance the fidelity of generated responses, particularly in numerical problem-solving scenarios. The retrieved examples serve as a starting point for the LLM to understand the problem, follow the chain of thought, and generate accurate solutions.

Furthermore, this retrieval-based prompting strategy allows for dynamic adaptation of the prompts based on the input question, enabling the LLM to leverage the most relevant information from the knowledge base. This approach ensures coherence and accuracy in generated solutions while reducing the reliance on fixed, potentially irrelevant prompts. [1]

Self-Verification: To further enhance the accuracy and problem-solving capabilities of the LLM, we implement a self-verification approach, inspired by (Lightman et al., 2023) and (Weng et al., 2023). Rather than relying solely on the LLM's initial output, we leverage the LLM as a feedback model to verify the generated solution. The original intuition behind this approach is that LLMs possess an innate ability to evaluate their own outputs, similar to how humans cross-check their reasoning steps.

We modify this approach by feeding our model's entire initial output back into it. We also add a prompt stating that the steps performed and previous answer might not be correct, and explicitly ask the LLM to redo the calculation. Since the initial output CoT is present in the new prompt, the LLM can both refer to the earlier solution, while generating a new one. This approach has two benefits. One, we are directly verifying both the output and the CoT in one go. In the absence of ground truth CoT steps, we hypothesize that this approach best trades off between step and answer verification. Two, we constrain the LLM's output each time to included reasoning steps, demarcated by a new line. This makes parsing the LLM's final answer a lot cleaner and also allows us to analyze the number of steps being taken each time we attempt to re-verify the answer.

We use two broad variations of the method above.

[1] Our code is available at https://github.com/smadeyan/anlp-numerical-reasoning

The first one involved prompting the LLM at most five times, and checking if any two consecutive answers were equal. We keep track of the number of verifications, and also measure a confidence score. An LLM is confident in its answer if it can generate the same answer twice consecutively within a total of five attempts. If not, we use the last generated answer, and deem the LLM to be unconfident with the specific question-answer pair. Our other approach involved prompting the LLM four times and keeping a count of each returned answer. We use the answer returned the most number of times, and deem the LLM to be confident in its answer, if it can generate the same answer, at least twice in the four attempts. This confidence measure is more relaxed than the previous one, but we don't believe that there should be any difference in accuracy between the methods.

By combining these three approaches – number encodings, retrieval-based few-shot prompting, and self-verification – we aim to significantly enhance the numerical reasoning capabilities of LLMs. The number encoding scheme provides a more intuitive representation of numerical data, facilitating efficient processing. The retrieval-based prompting strategy ensures relevant context and examples, minimizing hallucinations. Finally, the self-verification approach allows the LLM to iteratively refine its solutions, leveraging its own feedback to improve accuracy continuously.

4.1 Experimentation Process

We will examine the efficacy of our proposed methodology in an incremental manner,first evaluating the performance of each of the methods mentioned above individually. We will then evaluate the performance of an approach that combines retrieval-based few-shot prompting and self-verification, to help us understand if they work well together, even if one of them does not provide any significant performance gains on their own. And finally, we will combine these with the number encodings. This path of experimentation will allow us to determine how will our three approaches work together in a pipeline, and if any one of them has a greater impact on the LLMs numerical capabilities.

4.2 Finetuning Details

We ran our inference on the base version of llama-2-7b available on Amazon Sagemaker Jumspstart, and proceeded to finetune those base models on the same platform. The model's were deployed on a Sagemaker endpoint, and were running on a *ml.g5.2xlarge* instance. Finetuning was also done with the same instance. We ran this for 6 epochs, using LoRA (with alpha set to 32, and R set to 8).

Our methodology involved adding special tokens to Llama-2-7B's tokenizer, and then expanding its size. We were unable to do this on SageMaker, and hence had to do it with a 8-bit quantized version of Llama, and finetune using LoRA, due to computational constraints. Since its been shown that the quantized versions don't have a significant loss in performance (Bhandare et al., 2019), we feel that this will give us comparable performance metrics.

The dataset used for finetuning the models, with and without our encodings, was the GSM8K dataset (Cobbe et al., 2021), with a training set size of almost 7500 instances, and a train set of around 1300 instances. It consists of middle-school level math problems, that take from 2 to 8 steps to solve.

5 Results

Our obtained results, compared against the baseline, can be seen in Table 1. As mentioned in Section 4.1, we have obtained results iteratively for combinations of our pipeline components, leading up to all three being used together.

The 8-shot prompting specified in the results table refers to few-shot prompting the model with the same set of 8 examples for each inference, which is method used by Meta to achieve its baseline results on GSM8K [2]. Retrieval Augmented Few-shot prompting (RAFSP), on the other hand, is making use of 4-shot prompting, with prompts retrieved from the vector store. There are two reasons for using 4 prompt examples here - we wanted to use those examples returned with the highest confidence (and therefore likely to have the highest quality), and also compare the performance with using a set of 4 good quality and varying examples, versus a set of 8 good quality but constant examples.

A note on comparing results - one of the baselines we have added in Llama-2-7B finetuned on the GSM8k dataset. This is because our number encoding methodology finetunes the base model on a pre=processed version of the same dataset, and we want to account for any unfair advantage this would give our model when it comes to solving the GSM8K problem statements. When evaluating the performance of any combination involving the

[2] https://huggingface.co/meta-llama/Llama-2-7b

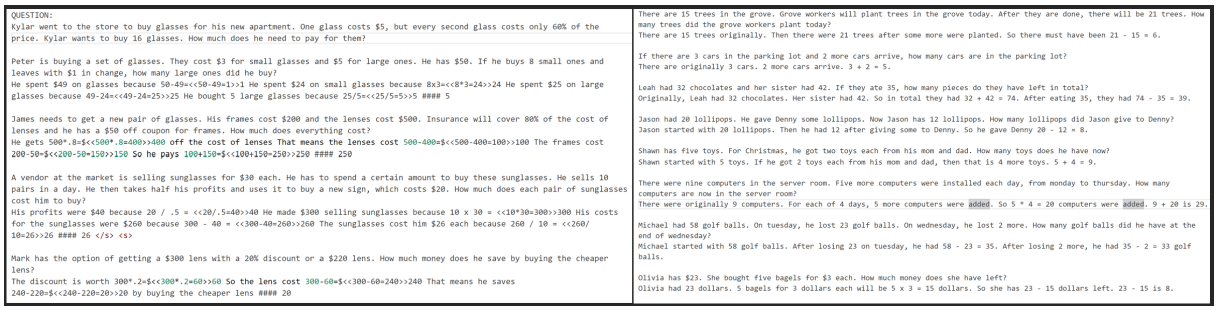

Figure 3: Examples of 8-shot constant prompt set vs retrieved prompt set

Table 1: Comparison of results

Model	GSM8K	MATH
Llama-2-7B (Zero Shot)	27.4	12.2
Llama-2-7B (8-shot/4-shot)	29.7	13.5
Llama-2-7B-GSM-8K (Zero-Shot)	31.3	12.3
Llama-2-7B-GSM-8K (8-shot)	34	13.8
Llama-2-7B + RAFSP	30.2	11.02
Llama-2-7B + Self Verify (Method 1)	31.04	11.51
Llama-2-7B + Self Verify (Method 2)	29.43	11.12
Llama-2-7B + RAFSP + Self Verify (Method 1)	27.95	10.23
Llama-2-7B + RAFSP + Self Verify (Method 2)	28.98	10.36
Llama-2-7B + Number Encoding (Zero-Shot)	28.61	11.07
Llama-2-7B + 8-shot + Number Encoding	29.65	11.24
Llama-2-7B + RAFSP + Number Encoding	29.54	11.27
Llama-2-7B + RAFSP + Self Verify + Number Encoding	29.72	9.58

number encodings, we have made to sure to also compare it against the llama-2-7b model finetuned on the regular GSM8K dataset. The combinations not involving our number encodings are mainly compared against the base llama-2-7b baselines.

To further ensure that we are not evaluating with an unfair advantage, we also **compare the performance of our number encoding methods** against the baselines on the MATH dataset (Hendrycks et al., 2021).

6 Analysis

We now analyze the results of each of the combinations mentioned in Table 1, leading up to the full pipeline detailed in Fig 2.

Llama-2-7B + RAFSP Running the base model with our retrieval augmented few-shot prompting resulted in minor improvements over the standard 8-shot prompting. One reason for this would the reduced number of examples (4) provided. Despite this, it slightly improved on the baseline performance. We did not try to retrieve 8 examples because of the lower scores for the ones beyond the top 4. However, there is scope to investigate different embeddings for the retrieval which would provide better results.

Llama-2-7B + Self Verify (Method 1) Self-verify worked reasonably well, improving on the baseline result by 3.6%. The average number of verification steps were 2.78 and nearly 70% of all answers were verified, as shown in Figure 4. However, of these verified answers, only 27% of them are correct. This means that Llama2 returned the same wrong answer consecutively 42% of the time, indicating that there are consistent flaws in its mathematical reasoning. These need to be looked at qualitatively to better understand them.

Interestingly, 6% of answers went unverified but were correct. Llama2 is capable of finding the correct answer(s), but cannot verify it by generating the same answer consistently. Potentially, more sophisticated versions of self-verification,such as stepwise verification can helpe guide its reasoning process.

Llama-2-7B + RAFSP + Self Verify (Method 1) Counter intuitively, retrieval augmented few

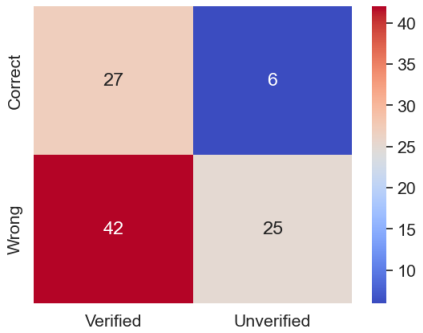

Figure 4: Confusion matrix of verified and correct answers with self-verify

shot prompting does not improve over regular constant few shot prompting, even for the GSM8K test set, suggesting that the question-answer pairs retriedved are actually confusing the model further. One possible reason for this could be that the answers retrieved contain more steps, equations and symbols than the original few-shot prompts.

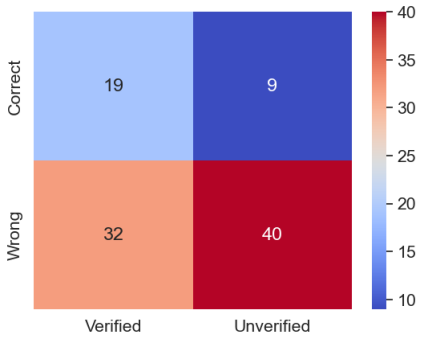

Figure 5: Confusion matrix of verified and correct answers with self-verify + RAFSP

Llama-2-7B + Self Verify (Method 2) Here, we prompt LLama2-7b four times for every question. We feed the previous output CoT back to the model every time, and ask it to verify its steps and generate the answer. The "most common answer" counts in figure 6 represent the count of the most common answer returned across these four attempts. We see that wrong answers have lesses confidence, measured by the most common answer count, indicating that these questions often confuse the LLM, and it cannot return consistent answers. This strategy can thus be further developed to steer Llama away from the wrong answer. As for correct answers, Llama consistently outputs the same answer most of the time (as indicated by the blue bars being taller on for counts 3 and 4.

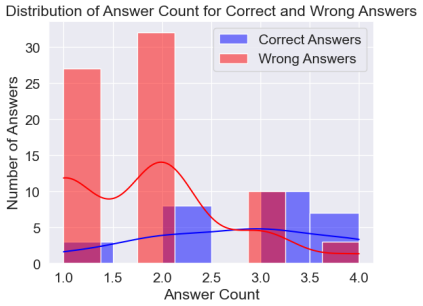

Figure 6: Distribution of "most common answer" counts for self-verify method II

Llama-2-7B + RAFSP + Self Verify (Method 2) On adding RAFSP to self-verify method II, the distribution of wrong answers' "most common counts" shifts to the left. The disribution for correct answers also shifts to the left, but not as significantly. Possibly, the decrease confidence in wrong answers can be attributed to the similarity of the questions retrieved to the main question we are prompting the LLM with. This might point to a flaw with this approach to self-verification. Instead of prompting a fixed four times, and using the latest answer, we can continue prompting till an answer that was already generated previously (if not consecutively) is generated again. At the same time, the model's confidence in correct answers also decreases and answers are generally more inconsistent. This is in line with our hypothesis from using RAFSP with self-verify's first method that RAFSP is confusing the model.

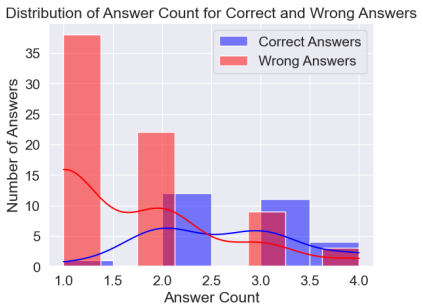

Figure 7: Distribution of "most common answer" counts for self-verify method II + RAFSP

Llama-2-7B + Number Encoding (Zero Shot) We initially finetuned the base llama-2-7b model with the encoded GSM8K training set (See without making any changes to the tokenizer. Our assumption was the the transformer architecture should be able to detect and maintain the pattern of the added tokens without having to expand the tok-

{"question": "Forty percent of the students have elected to learn from home during the pandemic. The remaining students are divided into two equal groups, only one of which is physically in school on any day. What percent of students are present in school?", "answer": "100-40=<<100-40=60>>60 percent of students are remaining.\n60/2=<<60/2=30>>30 percent are present in school.\n#### 30"}

[{'generated_text': '40% of the students are learning from home, so <40<fe>% * <<3<fe>,<se>1<de>000<fe> students> = <<<<se>4<de>4000<fe>*<<se>3<de>010<fe>*.<se>1<de>001<fe>=<se>3<de>400<fe>>><se>3<de>400<fe> students are learning from home.\nThe remaining <se>3<de>3600<fe> students are divided into <se>1<de>2<fe> equal groups, each of which is present in school <se>1<de>2<fe> days per week. <se>1<de>2<fe> groups * <se>1<de>2<fe> days/group = <se>1<de>4<fe> days per week.\n<se>3<de>3600<fe> students / <se>1<de>4<fe> days/'}]

Figure 8: Example of inference with encoded finetuning **without** expanding the tokenizer

Cherrie bought gifts for <se>1<de>5<fe> friends, and each of those friends wanted either <se>1<de>5<fe> or <se>1<de>2<fe> gifts. So, Cherrie bought a total of: <se>1<de>5<fe> * (<se>1<de>5<fe> gifts + <se>1<de>2<fe> gifts) = <<<se>1<de>5<fe>*<se>1<de>5<fe> +<se>1<de>5<fe>*<se>1<de>2<fe>=<se>2<de>25<fe>+<se>2<de>10<fe>=<se>2<de>35<fe>>><se>2<de>35<fe> gifts. Since Cherrie also got <se>2<de>10<fe> random gifts, the total number of gifts she bought is: <se>2<de>35<fe> + <se>2<de>10<fe> = <<<se>2<de>35<fe> +<se>2<de>10<fe>=<se>2<de>45<fe>>><se>2<de>45<fe> gifts. #### <se>2<de>45<fe>

Figure 9: Example of inference with encoded finetuning **after** expanding the tokenizer

enizer. However, while we were correct in the assumption that the transformer can maintain the pattern without breaking down the new tags into its characters, the LLM did not handle the numbers within those tags as expected, and the answers were somewhat incoherent. Fig 8 is an example of the response generated. The details about the number of digits are not maintained, and the calculations performed are correct in some places and incorrect in others, giving us inaccurate results.

We then proceeded to finetune the base llama model after expanded the tokenzier to include the special tokens '<se>', '<de>', '<ne>' and '<fe>'. This gave us more coherent outputs, and more consistent calculations. This is because the model now reconizes that these are special tokens which should not be broken down, and focuses on the values between the tokens and identifying the relevant patterns for those values. An example of inference after expanding the tokenizer can be seen in Fig 9.

The results are slightly higher than the values obtained by the base llama-2-7b model, but lower than the GSM8K-finetuned model (which provides a better comparison). Looking at some of the calculation mistakes the model makes, it's apparent that it sometimes combines the values with the tags into a single number. For example, *<se>2<de>34<fe>* can get combined to give 234, which gets used in subsequent calculations, leading to inaccuracies. Other reasons for the lower performance is inadequate training set size and epochs. Considering that our aim is to change how the LLM is parsing numbers, its likely we need more than 7000 instances to get it to do this in a consistent manner, and train it for more than 6 epochs. RefinedWeb (Penedo et al., 2023) could be an example of a dataset to consider, for a larger number of training instances, which are also more generalized in content.

We see a degraded performance on the MATH dataset. In addition to the considerably greater difficulty level of the questions, it's likely that the presence of the additional LaTeX characters, combined with our additional tokens, are making it difficult for the model to parse the numbers effectively.

Llama-2-7B + 8-shot + Number Encoding The performance with number encoding increased slightly with 8-shot prompting (with a constant set of examples). While it is noticeable lower than the performance of the GSM8K finetuned Llama (without 8-shot prompting), it is encouraging that the encoding-finetuned model responds well to well-established prompting techniques. However, **these examples, and the base prompt, also need to be encoded** in order for the model to parse them correctly. We initially tried prompting, with examples, using regular text, but the model's performance degraded noticeably (by around 5%). The probablity distribution for the model's tokenizer has changed, since there are many numbers, and textual representation of numbers, which have been encoded with the addition of the new tokens. Therefore, preprocessing step will need to apply to the prompts as well.

When evaluated on the MATH dataset, we again see a degradation in performance. The reasons for this would remain the same as the ones we discussed for the previous combination, but this would also indicate that the model is not generalizing well enough, and would need to be finetuned on a larger and more generic encoded dataset.

Llama-2-7B + RAFSP + Number Encoding Performance with this combination remained roughly the same as with 8-shot prompting. Since

the prompt examples are retrieved before any encoding is performed, we can be sure that the new tags play no role in impacting the quality of the examples being returned. But it does not beat the baseline performance of the GSM8K finetuned Llama. The analysis from the previous combinations would hold here - researching more effective retrieval embeddings, and training the model on a larger corpus of encoded data, for a longer period.

Llama-2-7B + RAFSP + Self Verify + Number Encoding On using the whole pipeline, we do not see any noticeable improvement in performance when compared to some of the previous combinations, and a reduced performance when compared to others. This implies that the components in our pipeline are not working in a synergistic manner. As stated in the earlier analysis for RAFSP + Self-Verify, the additional symbols are most likely confusing for the model in the self-verify steps. Additionally, based on our analysis of the earlier combinations, the number encodings are not working as well as expected, and would benefit from a much larger training size and training time. Finding a better embedding for our retrieval, and more careful curation of our prompt store should also yield an improved performance. Another point to note is that there is an increase in overall response time, once we factor in the time taken for the retrieval step.

7 Limitations and Future Work

Despite the innovative approaches and methodologies we applied, we acknowledge the following limitations:

1. Model and Data Constraints: Our experiments were confined to a single model, LLama 2 7B, and specific datasets like GSM8K and MATH. Although these choices were made to establish a robust baseline, they may limit the generalizability of our findings across different models and broader types of numerical reasoning tasks.

2. Complexity of Number Encoding: While our novel number encoding approach showed potential, it introduced complexities in model training and interpretation. The encoded format requires the model to adapt to a significantly altered input structure, which might not seamlessly integrate with traditional LLM training paradigms.

3. Error Propagation in Self-Verification: The self-verification method, while beneficial in improving answer accuracy, also risked propagating and reinforcing incorrect reasoning if the initial outputs were flawed. This could lead to consistent but incorrect answers, as observed in several instances during our testing.

4. Dependency on Retrieval Quality: The effectiveness of the retrieval-augmented few-shot prompting approach heavily depends on the relevance and quality of the retrieved examples. The embedding becomes a bottleneck for retrieval. Misalignment in this context can adversely affect the model's performance, as seen in some of our experimental results.

To address these limitations and further refine our approach, the following future work is proposed:

1. Synthetic Data Sets: Incorporating a broader range of datasets, including those with more complex and varied numerical reasoning challenges, could help in developing more robust numerical reasoning capabilities in LLMs. The use of datasets like RefinedWeb might also provide a more comprehensive training base.

2. Improved Retrieval Mechanisms: Enhancing the retrieval mechanisms to ensure higher quality and more relevant prompt examples could improve the efficacy of retrieval-augmented prompting methods. Leveraging recent advancements in semantic understanding and retrieval could be particularly promising.

8 Acknowledgements

We thank the following individuals for their guidance through out our project:

1. Dr. Graham Neubig

2. Akshay Goindani

3. Vishwa Shah

References

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset.

Mohammad Bavarian Mark Chen Heewoo Jun Lukasz Kaiser Matthias Plappert Jerry Tworek Jacob Hilton Reiichiro Nakano Christopher Hesse John Schulman Karl Cobbe, Vineet Kosaraju. 2021. Training verifiers to solve math word problems.

Xianming Li and Jing Li. 2023. Angle-optimized text embeddings.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2024. A comprehensive overview of large language models.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems?

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only.

Eli Schwartz, Leshem Choshen, Joseph Shtok, Sivan Doveh, Leonid Karlinsky, and Assaf Arbelle. 2024. Numerologic: Number encoding for enhanced llms' numerical reasoning.

KV Aditya Srivatsa and Ekaterina Kochmar. 2024. What makes math word problems challenging for llms?

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Yixuan Weng, Minjun Zhu, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. 2022. Large language models are reasoners with self-verification. *ArXiv*, abs/2212.09561.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models.