MDS5210 FINAL PROJECT – LANGUAGE MODELS [*]

Zijin CAI 224040002@link.cuhk.edu.cn

Yixin ZHOU 224040090@link.cuhk.edu.cn

Guyuan XU 224040074@link.cuhk.edu.cn

Zhen QIN 224040054@link.cuhk.edu.cn

Ziying LIU 224040041@link.cuhk.edu.cn

Weixuan ZENG 224040003@link.cuhk.edu.cn

MSc in Data Science
School of Data Science

The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong Province, China

ABSTRACT

The part I of this project promotes basic concepts of language modeling and implements a simple pre-training of the GPT2 model to generate text that mimics the style of Shakespeare. This section culminates with a detailed account of the training and validation loss, complemented by illustrative samples of generative text. The part II fine-tunes the well pre-trained GPT2 model (openAI-community/gpt2) [8], based on Alpaca-GPT4 dataset, and then conducts human evaluation on efficacy of the generated text. The last part III section examines downstream generalization performance of the GPT2 model, assessing its performance on mathematical benchmark scores and further evaluating the efficacy of the chain-of-thought (CoT) prompt technique quantitatively.

***K*eywords** Large Language Models (LLMs) · GPT2 (Fine-Tuning) · Alpaca-GPT4 Dataset · Chain-of-Thoughts (CoT)

1 Introduction

Large Language Models (LLMs) are a transformative development in the field of artificial intelligence and machine learning, having a profound impact on numerous applications including text generation, translation, summarization, and question-answering, among others [1]. These models, which are trained on vast amounts of text data, have the ability to generate human-like text that can be remarkably coherent, insightful, and contextually nuanced. One of the most well-known LLMs is ChatGPT, developed by OpenAI, which is widely utilized demonstrating capabilities in understanding and generating text. Other notable LLMs include Llama [3] and Claude family [9], developed by Meta and Anthropic, respectively. Nowadays, LLMs represent a significant advancement in the field of artificial intelligence, providing a convincing approach towards artificial general intelligence (AGI).

2 Part I: A Simple Pre-training of the GPT2 Model

"GPT" is the acronym of "Generative Pre-trained Transformer", the magnificent of LLMs is mainly responsible for the pre-training process, which involves learning from vast amounts of text data to predict the next word in a sequence of words, given the context of the preceding words. This allows the model to learn the structure of the language, including grammar, syntax, and semantics, even the reasoning abilities, world knowledge and some level of common sense.

[*] *Zijin CAI / Guyuan XU*: prompt improvement and downstream metric investigation, code append and debug
Yixin ZHOU / Ziying LIU: implement CoT and prompting orders design, latex report documentation.
Zhen QIN / Weixuan ZENG: Llama-3.2 (-Instruct) evaluation on different datasets, results summarization.

The learning problem of LLMs is a maximum likelihood estimation (MLE) that, given a set of training data pairs $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, it minimizes the negative log-likelihood of the generation probability [10]:

$$\widehat{\mathbf{W}} \leftarrow \operatorname*{argmin}_{\mathbf{W}} \mathcal{L}(\mathbf{W}) = \sum_{i \in \mathcal{D}} \sum_{j=1}^{m} -\log\left(\mathbb{P}_{\mathbf{W}}\left[y_i, j | \mathbf{x}_i, \mathbf{y}_i, 1 : j-1\right]\right) \tag{1}$$

2.1 Training and Validation Loss

The training and validation loss are shown in Figure 1, and the final loss values at iteration 5000 are reported below.

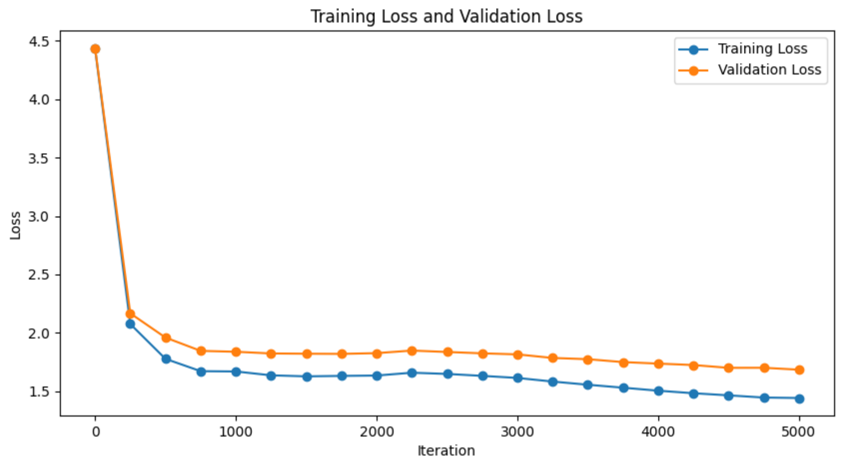


Figure 1: Training and Validation Loss

```
Train loss: 1.441351294517517 Validation loss: 1.6838688850402832
```

We can see a positive trend during the model's learning process that, as the iterations progresses, both the training and validation loss decrease steadily in the first few hundred iterations, indicating that the model is learning the structure of the Shakespear language from the training data. After the first few hundred iterations, the loss values continue to decline with slighter fluctuations, suggesting that the model's learning is stabilizing and converging. By the 5000th iteration, the training loss is 1.44, while the validation loss is slightly higher around 1.68, which are relatively low values indicating that the model has learned the language structure well. The relatively close values of the training and validation loss suggest that the model is not overfitting the training data, and it is generalizing well to unseen data.

However, it wasn't until 2500 iterations that we observed a significant increase in the rate at which the losses were converging. This suggests that there is potential for further convergence if we were to extend the training duration, given that the losses continued to drop sharply. Upon examining the rate of convergence, it appears that the learning rate's weight decay played a crucial role. It likely helped to slow down the search process, which could prevent the model from overshooting the optimal solution. Additionally, the widening gap between the training and validation losses indicates that overfitting was becoming a more pronounced issue. This makes sense considering that the small dataset may not be sufficient for training a compact 24-layer, two-block Transformer decoder effectively.

2.2 Generated Text for Model Trained

Since we trained the entire autoregressive model at the character level, which is expected that the generated text may not always adhere to standard word and sentence structures. While many of the words appear to be common, they often violate syntactic grammars rules. This is because language modeling focuses on predicting the next token with the highest conditional probability. When certain characters frequently co-occur, their conditional probability increases, leading the pre-trained model to predict entire words. However, by working at the character level, we lose significant syntactic information, making it challenging for the language model to produce full sentences that follow grammatical rules. Furthermore, the output contains numerous line breaks, likely due to the prevalence of short dialogues in the original Shakespeare dataset. (The example of generated text can be found in the Appendix A.)

3 Part II: Instruction Fine-tuning GPT-2 Model

Optimizing the autoregressive loss enables the model to effectively generate text mimicking the style of given corpus and capture the underlying language structures. In practice, pre-training a model from scratch on next token prediction loss is computationally expensive and time-consuming, which is usually in the tokens scale of billions enabling the model to learn wide knowledge domains and linguistic patterns. For instance, the GPT-3 model has 175 billion parameters and is trained on over tokens of 570GB text data, which can be challenging to replicate the training process. Nevertheless, training a language model on a vast amount of text does not automatically transform them into potent agents. Essentially, the pre-trained model functions as a text completion tool. To enhance its utility and effectiveness, additional fine-tuning of the pre-trained model is required to equip it with the skills necessary for specific tasks like answering questions, retrieving knowledge, and planning tasks.

Instruction Fine-Tuning – The IFT is a process that involves training a pre-trained model on a specific dataset to perform a particular task, which basically optimizes the same autoregressive objective function 1, i.e., the next token prediction likelihood, but on a fine-tuning dataset that is specific to the task (often much smaller than the pre-training dataset). IFT has been shown to be effective in improving the performance of LLMs on a wide range of tasks, including text generation, prompted question-answering, and following human instructions.

3.1 Alpaca-GPT4 Dataset and The GPT-2 Model

The Alpaca-GPT4 dataset contains human-written instructions for a wide range of tasks, which is designed to evaluate the ability of LLMs to follow human instructions, consisting of 52k instruction-following data and 1.5M tokens, which is a relatively small dataset compared to the pre-training dataset of GPT-2, but is sufficient for fine-tuning the model. The GPT-2 model is a transformer model developed by OpenAI, which pre-trained on a large corpus of English text data in a self-supervised manner, consisting of 137 million parameters. [2]

3.2 Instruction Tuning Pipeline

To equip model with the ability to follow human instructions and generate reasonable response, direct concatenation may not be efficient for model to learn the task-specific patterns. The following template is proposed for tokenizing the instruction-following data for training (as well as for generate answer, except the response part should be left blank):

```
### Instruction: <instruction text here>
### Input: <input text here>
### Response: <response text here>
```

The sample in fine-tuning are in different lengths (i.e., `block_size`), which is different from the pre-training. Padding text is required to perform batch training that ensuring all batch samples have the same length (e.g., the maximum length). The padding text corresponding to the default padding token ″50256″ of GPT-2 model is `<|endoftext|>`.

(please refer to `data/instruction_tuning/prepare.py` and `config/finetune_alpaca.py` for implementation details of `tokenize_dataset()` and `get_batch_for_IT()` for tokenization and padding logic for batch sampling)

3.3 Training under Limited Resources

Float 16 Training: The memory consumption and computational cost depend on the data type used for training. The `float16` training is a technique that uses 16-bit floating-point numbers instead of 32-bit floating-point numbers to reduce memory requirements (approximately a half) and speed up training process, depending on the hardware utilization. However, the `float16` training may lead to numerical instability and precision loss, because of the narrower range and precision of 16-bit floating-point numbers, which may affect the model performance.

Memory Efficient Optimization: The widely utilized optimization algorithms like Adam and AdamW [4] are memory-intensive, which store the model, gradient, momentum, and second momentum terms, therefore is $4\times$ the memory cost of storing model alone. The following three approaches are proposed to reduce the memory consumption:

[2] *GPT2 model*: `https://huggingface.co/openai-community/gpt2`

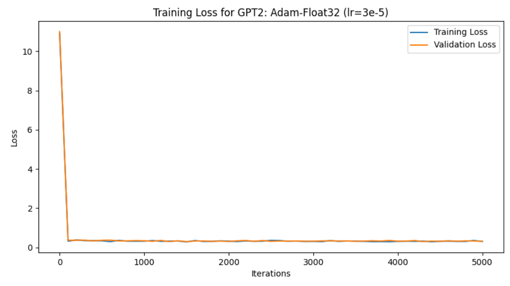
- **Stochastic Gradient Descent (SGD):** The SGD is a memory-efficient optimization algorithm that only requires storing the model and gradient terms. However, the convergence of SGD is usually slower than Adam and AdamW, which may require more iterations to converge.
- **Low-rank Adaption (LoRA):** [3] The LoRA freezes the pre-trained weight and only undate the factorized low-rank matrix:
$$\mathbf{W} = \mathbf{W}_0 + \mathbf{BA} \tag{2}$$
where $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$ is the pre-trained weight frozen, $\mathbf{B} \in \mathbb{R}^{m\times r}$ and $\mathbf{A} \in \mathbb{R}^{r\times n}$ are the factorized low-rank matrix to be trained. The hyperparameter r is the rank of the factorized matrix, which is usually much smaller than m and n, therefore the total trainable parameters are reduced to $(m+n)\times r$.
- **Block Adam (BAdam):** [4] BAdam partitions the model parameters into blocks and only updates a subset of blocks in each iteration. The block-wise update treat each single transformer layer as one block, which can be much smaller than the whole model, significantly reduces the memory consumption and computational cost.

3.4 Results and Analysis

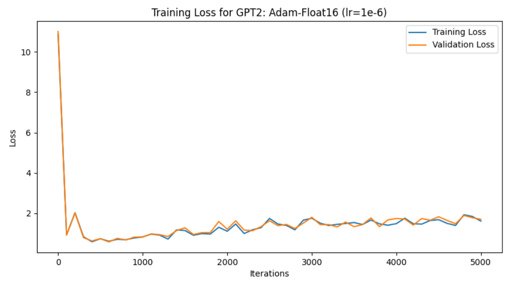
(please refer to `train.py` and `model.py` for detailed implementation of different optimization methods)

Effect of Float 16 Training:

The training and validation loss of GPT2-Adam model under float32 and float16 are shown in the following figures 2:



(a) Loss of Adam-float32 with GPT2



(b) Loss of Adam-float16 with GPT2

Figure 2: Training and Validation Loss of Adam with GPT2 (float32 and flaot16)

The loss for Adam float32 drops quickly in the initial stage and then levels off, indicating the model is learning rapidly and converges during training. The descent rate and the final plateau differ for the Adam float16, which shows more fluctuation after the initial drop resulting in a loss value higher than the float32. Despite the lower numerical precision of the float16, the loss performance is still comparable, with no significant degradation in the model's learning ability.

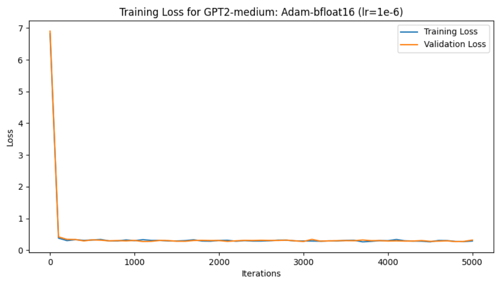
However, the float16 training costs significantly less memory reducing the computational time compared to the float32 (reported in table 1, which is approximately a half). This can be beneficial for training large language models on massive datasets, given the limited resources, and speeds up the training process with more efficient iterations progress.

Memory Efficient Optimization:

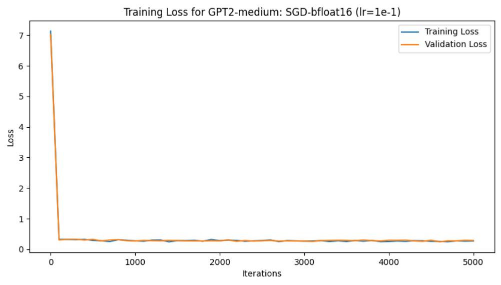
The following figures 3 show the losses of different optimization methods with GPT2-Medium (bfloat16):

All methods show a similar pattern of convergence such that, the training loss of Adam-bfloat16, SGD-bfloat16 and LoRA-bfloat16 are similar in the initial stage that decrease rapidly and then stabilize after the few iterations. However, the initial drop of SGD-bfloat16 is slightly more steep than the other two methods, due to the larger learning rate of SGD pre-determined. The BAdam-bfloat16 performs the smoothest descent in the initial stage and take more iterations to converge, which prevents the sudden drop of loss values and thus ensures a stable learning process.

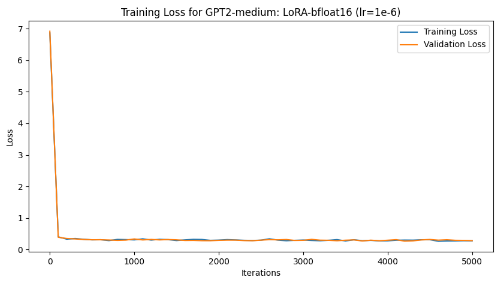
[3] *LoRA of LLM*: `https://github.com/microsoft/LoRA`
[4] *BAdam of LLM*: `https://github.com/Ledzy/BAdam`



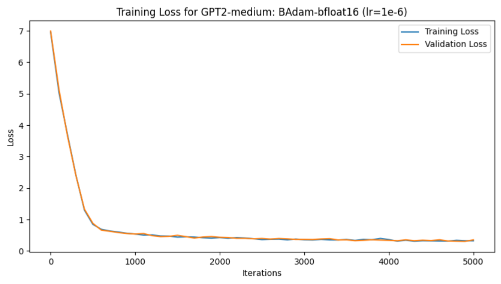
(a) Loss of Adam-bfloat16 with GPT2-Medium



(b) Loss of SGD-bfloat16 with GPT2-Medium



(c) Loss of LoRA-bfloat32 with GPT2-Medium



(d) Loss of BAdam-bfloat16 with GPT2-Medium

Figure 3: Training and Validation Loss of Different Methods with GPT2-Medium (bflaot16)

In terms of the memory consumption and computational cost(1), the Adam-bfloat16 requires the most memory, because of its memory-intensive nature for storing past gradients. Followed by the SGD-bfloat16, LoRA-bfloat16, and BAdam-bfloat16, the resource allocations are consistent with the memory efficiency of the optimization methods. BAdam-bfloat16 appears to be the most memory-efficient and computationally cost-effective method, beneficial from its design of handling the bfloat16 data type, which is preferred for training large models on limited resources.

	Adam fp32	Adam fp16	Adam bfp16	SGD bfp16	LoRA bfp16	BAdam bfp16
Max Memory	10.234 GB	6.174 GB	14.074 GB	12.654 GB	10.459 GB	8.648 GB
Time per Iter	441.88 ms	216.33 ms	593.99 ms	529.99 ms	449.28 ms	425.51 ms
Pre-train Model	`gpt2`	`gpt2`	`gpt2-medium`	`gpt2-medium`	`gpt2-medium`	`gpt2-medium`

Table 1: The Maximum Allocated Memory and Training Time for Different Optimization Methods

3.5 Evaluation of Instruction-Following Capability

The ultimate goal for evaluating the model's generation quality is to align the response with human inference. The following human grades evaluation are conducted to assess the model's instruction-following capability, criteria include helpfulness, relevance, and fluency of the generated response. There are 20 designed questions diversified in different aspects, (e.g., simple common knowledge, common sense, simple reasoning, and verbally instructed math questions, etc.) that each question is prompted as input text to the (fine-tuned) model for generating response. The model information for neither pre-trained base, nor fine-tuned model by different algorithms, is disclosed when performing evaluation. Out of the rating scale of $1 - 10$, the average scores for each method are reported in the following 4:

GPT2	float32	float16
Pre-trained	2.90	–
Adam	4.85	0.78

(a) Average Rating of GPT2

GPT2-Medium	Adam	SGD	LoRA	BAdam
bfloat16	5.72	5.54	5.28	4.50

(b) Average Rating of GPT2-Medium with bfloat16

Figure 4: Evaluation Table of Pre-trained and Finetuned Models

The evaluation results are coincide with the loss performance of different optimization methods, where the based model GPT2 responses colloquially and less relevant to the questions, and the fine-tuned models generate more relevant and helpful responses. There are significant improvements for fine-tuning the model based on GPT2-Medium, which shows the importance of pre-training model quality, despite some loss of precision in the type of float16 training.

The following is an example question and the corresponding generated responses with different methods:

```
### Instruction:
A company's revenue increases by 10% each year. If the initial revenue was R dollars, what will
be the revenue after 5 years?
### Input:
R=100,000 dollars
### Response:
(gpt2): So what kind of businesses are people buying through these companies? Who is buying the
product? This is a huge question that needs to be answered in every year's game of Survivor.

There are three categories of new game, and those who are seeking out these first are those

(gpt2-adam_float32): Based on the company's revenue over 5 years, 10% is the initial revenue
after 5 years, and the initial revenue after 5 years is R dollars.

(gpt2-adam_float16): Center expansion = 100 100 100 100 100 100 100 100 100 100 100 100 100

(gpt2_medium-adam_bfloat16):
1. R=100,000 dollars

2. After 5 years, the initial revenue will be R=100,000 dollars.

(gpt2_medium-sgd_bfloat16): From the company's revenue, the revenue after 5 years is $100.

(gpt2_medium-lora_bfloat16): The initial revenue for a company will be 100,000 dollars. After
5 years, the revenue will be R dollars.

(gpt2_medium-badam_bfloat16): If the initial revenue was $100,000 dollars, the company revenue
would be $109,200 dollars.
```

3.6 Discussion: Research Limitations and Difficulties

In general, there are two training paradigm: fixed and mixed precision training. The fixed precision training is that all the variables are fixed to be either fp32 or fp16. The bfp16 is commonly used when the precision is the same for all parameters, however, it might not be supported for some old hardware. Another approach is to train with mixed precisions, where low precisions (fp16 or bf16) are stored for forward and backward passing, then update the model with high-precision (fp32). The precision-wise training calculate the gradient with low precision, then transform it into high precision gradient, and thus update the model with high precision, copying the weight to low precision model. Therefore, more precise model update can be achieved with the advantage of memory and computational efficiency.

While our project has made significant strides in understanding and applying language models, particularly in the context of GPT2 and its fine-tuning capabilities, there are several limitations and challenges that warrant discussion. The training of large language models is resource-intensive, requiring significant computational power and memory. Our project was conducted with limited resources, which restricted the scale and duration of our training. This limitation impacted the model's convergence and its ability to achieve optimal performance. We experimented with different training precisions and optimization algorithms to balance between model performance and computational efficiency. However, the use of float16 and bfloat16 precisions, while beneficial for reducing memory usage, introduced potential numerical instability and precision loss, which could affect the model's accuracy. The GPT2 model, with its 24-layer, two-block Transformer decoder, is complex and may not be the most efficient architecture for the size of the dataset we used. Balancing model complexity with the available data is a challenge that affects the overall effectiveness of the model. In conclusion, while our project has achieved notable results, these limitations and difficulties underscore the need for further research and development in the field of language modeling. Addressing these challenges will be crucial for advancing the capabilities of large language models and their applications in real-world scenarios.

4 Part III: Chain-of-thoughts Prompting

Besides the fine-tuning, the prompting strategy can be responsible for the model's generation quality. The part III of the thesis is dedicated to the analysis of the chain-of-thoughts (CoT) prompts, which is one of the most effective prompting strategies widely adopted in improving the model's generation capability. Quantitative evaluations are conducted to investigate the model performance using math benchmarks scores instead of the human evaluation.

4.1 Dataset and Model

The `GSM8K` [2], `NumGLUE` [6], `SimulEq` [5] and `SVAMP` [7] datasets are used to evaluate the model's performance. Each dataset contains math questions and corresponding answers from different domains, where the problem difficulty varies from simple arithmetic to complex algebraic equations. The `Llama` [3] model is used for the evaluation, (In particular, the `Llama-3.2-3B` [5] and its instruction-fine-tuned `Llama-3.2-3B-Instruct` [6] models.)

4.2 Chain-of-Thoughts Prompt

The CoT prompt is a sequence of questions and answers that guide the model to generate the desired output. Different from the few-shot learning, the sample answers in CoT prompt detailed step-by-step instructions to help the model understand the problem-solving process. Additionally, the CoT prompt usually format the instructions in a conversational style, which is more human-friendly and easier to understand. (e.g., `Let's solve the problem step by step`)

Effect of CoT Prompt Amount

model (CoT) / dataset	`GSM8K`	`NumGLUE`	`SimulEq`	`SVAMP`
Llama-3.2-3B *(CoT-0)*	0.0546	0.1556	0.0487	0.1522
Llama-3.2-3B *(CoT-2)*	0.2466	0.2277	0.0780	0.2923
Llama-3.2-3B *(CoT-4)*	0.2473	0.2536	0.1072	0.3303
Llama-3.2-3B-Instruct *(CoT-0)*	0.7587	0.4640	0.4776	0.8198
Llama-3.2-3B-Instruct *(CoT-2)*	0.6517	0.4803	0.3938	0.6016
Llama-3.2-3B-Instruct *(CoT-4)*	0.6555	0.4803	0.4327	0.5866

Table 2: Model Performance (Accuracy) with Various Number of CoT Samples on Different Datasets

- *CoT Prompts Benefit*: The use of CoT prompts generally enhances model performance, with the 4-shot prompts often providing the best results, especially for the datasets using base model (Llama-3.2-3B).
- *Fine-tuned Model Sensitivity*: The fine-tuned Llama-3.2-3B-Instruct shows high initial performances with CoT-0 but may experience a decrease in accuracy with more CoT examples, particularly in the SVAMP dataset.
- *Dataset-Specific Effects*: The impact of CoT prompts varies across datasets. While GSM8K and SVAMP show significant improvements with CoT prompts, NumGLUE and SimulEq exhibit more moderate changes.
- *Optimal CoT Prompt Amount*: The optimal number of CoT examples may depend on the dataset and the model's fine-tuning. For the base model, more examples tend to be beneficial, while the fine-tuned model's performance peaks with fewer examples or no examples at all in some cases.

The concatenation of the CoT template is implemented before asking the real question, and the accuracies (scores) are reported on table 2, including all benchmarks for both the base and fine-tuned models under 0, 2 and 4 -shot prompt, with respect to the 4 datasets. The results show that the use of CoT prompts generally improves the model's performance compared to the 0-shot scenario across all datasets. The 4-shot prompts tend to yield better or comparable performance to the 2-shot prompts for most cases, given the effectiveness of the dataset-wise and model-specific CoT prompts.

[5]*Llama-3.2-3B*: `https://huggingface.co/meta-llama/Llama-3.2-3B`
[6]*Llama-3.2-3B-Instruct*: `https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct`

Effect of CoT Prompt Order

Experiments are conducted on the `GSM8K` dataset, evaluating the performance of the base Llama model with different CoT prompting orders. The results are summarized in table 3, and observations is that the model experiences issues with KV cache space, leading to many sequence groups being preempted in RECOMPUTE mode, which could affect the performances. There are several warnings about the inability to convert certain strings to floating-point numbers, which might be due to the format of the answers or errors in the data. The results suggest that the CoT prompt order can significantly impact the model's performance, with certain orders leading to better or worse results.

Table 3: Summary of Experiments (Accuracies) with Different CoT Prompt Orders

CoT Prompt Order	Accuracy		# Conv_to_float Fails	Notes
	Mean	Std_Dev		
Angelo and Melanie, Mark, Natalia, Bella	0.247	0.005	105/0	-
Angelo and Melanie, Mark, Bella, Natalia	0.222	0.006	150/0	-
Angelo and Melanie, Natalia, Mark, Bella	0.224	0.004	-	-
Angelo and Melanie, Natalia, Bella, Mark	0.243	0.003	50/	-
Angelo and Melanie, Bella, Mark, Natalia	0.225	0.007	-	-
Angelo and Melanie, Bella, Natalia, Mark	0.244	0.002	8/0	-
Mark, Angelo and Melanie, Natalia, Bella	0.250	0.004	3/	Best Performer
Mark, Angelo and Melanie, Bella, Natalia	0.212	0.008	1000/	Lowest Performer
Natalia, Angelo and Melanie, Mark, Bella	0.244	0.006	42	-

Prompt Improvement and Downstream Metric

There are several aspects for improving the CoT (Chain of Thought) example and the post-processing of the generated answers. One need to consider whether the CoT example provide a logical and easy-to-follow thought flow, while if the question is unambiguous, and if the answer is clearly stated. Ideally, the CoT example can be input to GPT and ask for suggestions on additional insights or alternative phrasings for effective improvements. To better handle the post-processing of the generated answers, we can also improve `answer_clean` and `compare_answer_with_groundtruth` functions. For instance, the adjustments are supposed to ensure that the function correctly extracts the answer from the model's response, while handling edge cases where the answer might be formatted differently.

The following code are introduced to improve `answer_clean()` and `compare_answer_with_groundtruth()`:

- *Enhanced Regular Expressions*: refine the regular expressions used to better match and clean the output. including ensuring that fractions and decimals are correctly identified and remove unnecessary characters.
- *Dataset-Specific Cleaning*: For the $math$ dataset, add specific cleaning logic to remove ordinal suffixes (e.g., st, nd, rd) and dollar signs, which are not typically part of the numerical answer.
- *General Cleaning Logic*: maintain a general cleaning process that removes special characters, leaving only letters, numbers, spaces, dots, and hyphens. This ensures that the answer is in a clean, standard format.
- *Case Insensitivity*: convert both the answer and the ground truth to lowercase for comparison case-insensitivity.
- *Numeric Comparison with Tolerance*: implement numeric comparison that allows small tolerance when comparing floating-point numbers, as floating-point arithmetic can sometimes lead to small rounding errors.
- *Handling Multiple-Choice Answers*: add logic to handle multiple-choice answers by checking if the answer matches the pattern of a single letter (a-z) and comparing it directly to the ground truth for the correct format.

By diligently addressing the aspects outlined in our discussion, we can achieve a marked enhancement in the Chain of Thought (CoT) prompting strategy and the subsequent post-processing of the model-generated answers. In essence, the pursuit of excellence in CoT prompting and answer post-processing is not just about refining a tool, but advancing the ability to harness the full potential of language models. The following python code snippet demonstrates the improved functions `answer_clean()` and `compare_answer_with_groundtruth()`, while table 4 illustrates the comparison between the original and improved CoT examples, with the corresponding accuracy scores for each case.

```python
def answer_clean(dataset, pred):
    # Existing cleaning logic...
    # Enhance the regular expression to match fractions and decimals
    pred = re.sub(r'\s+', ' ', pred)  # Replace multiple spaces with a single space
    # Remove special characters, keep letters, numbers, spaces, dots, and hyphens
    pred = re.sub(r'[^a-zA-Z0-9\s./-]', '', pred)

    if dataset == "math":
        # Specific cleaning logic for math problems
        pred = re.sub(r'\b(\d+)(th|st|nd|rd)\b', r'\1', pred)  # Remove ordinal suffixes
        pred = re.sub(r'\$', '', pred)  # Remove dollar signs from math expressions

    # Specific logic for other datasets...
    return pred.strip()

def compare_answer_with_groundtruth(answer, groundtruth_str, groundtruth_num=None):
    # Existing comparison logic...
    # Ignore case
    answer = answer.lower()
    groundtruth_str = groundtruth_str.lower()

    # Numeric comparison, using a tolerance
    if groundtruth_num is not None:
        try:
            answer_num = float(answer)
            groundtruth_num = float(groundtruth_num)
            return abs(answer_num - groundtruth_num) < 0.01  # Tolerance is 0.01

        except ValueError:
            pass

    # Handle multiple-choice answers
    if re.match(r'[a-z]', answer) and re.match(r'[a-z]', groundtruth_str):
        return answer == groundtruth_str

    # Other comparison logic...
    return answer == groundtruth_str
```

Original 1-Shot CoT Example Accuracy: 0.2086	**Improved** 1-Shot CoT Example Accuracy: 0.8751
### Instruction: Jade has 20 spades in her deck of cards. She takes 8 cards out and replaces them with hearts. She then takes 7 cards out and replaces them with diamonds. How many cards does she have left in the deck? ###Response: Jade started with 20 spades, and took 8 hearts out. After replacing the 8 hearts, she has 20 spades + 8 hearts = 28 hearts. Jade then took 7 diamonds out, and replaced them with 7 spades. Jade now has 28 hearts + 7 spades = 35 cards left in her deck. The answer is 35	### Instruction: Jade starts with a deck containing 20 spades. Firstly, she removes 8 spades and adds 8 hearts, then, she removes 7 spades and adds 7 diamonds. How many cards are there in total in Jade's deck now? ###Response: Jade began with a deck containing 20 spades. She removed 8 spades and added 8 hearts to the deck. After this exchange, she had 20 cards. Next, she removed 7 hearts and added 7 diamonds, maintaining the total count at 20 cards. The answer is 20.

Table 4: Table of Original and Improved CoT Example with the Corresponding Accuracy

References

[1] Tom Brown et al. "Language models are few-shot learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 1877–1901.

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

[3] Abhimanyu Dubey et al. "The LLaMA 3 herd of models". In: *arXiv preprint arXiv:2407.21783* (2024).

[4] Diederik P Kingma and Max Welling. "A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[5] Rik Koncel-Kedziorski et al. "Mawps: A math word problem repository". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016, pp. 1152–1157.

[6] Swaroop Mishra et al. "Numglue: A suite of fundamental yet challenging mathematical reasoning tasks". In: *arXiv preprint arXiv:2204.05660* (2022).

[7] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. "Are nlp models really able to solve simple math word problems?" In: *arXiv preprint arXiv:2103.07191* (2021).

[8] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[9] Hugo Touvron et al. "LLaMA 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288* (2023).

[10] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017.

Appendix A

Character	**Dialogue**
ISABELLA:	No, by you madam, in their corces shall in put Lare the happower home, where it's this is night Take thing of the eyes; and a chile And shall be will be the world busing.
LEONTES:	She the give in me; and have so not the poor.
HERMIONE:	Part being maint betty bold so that at be That to ping this to beater, And not these furth.
POLIXENES:	It say her he's so her so under my father; He speak him stands and that be havel That should hopet of the desilvest welcomple one, Friend that I ca *If the such these sperince foes princess son the lust makes aff.*
All:	Tis shall weep you no more they stay, The have your sun passents some of my lord.
CORIOLANUS:	Ay, sir, I am my lords some honoughter, he would have I this no love wing please to lead; and good Where this face you no love and true?
BRUTUS:	By the brother of this That will that him in in to thee.

Table 5: Dialogues from Shakespeare's Plays