Report

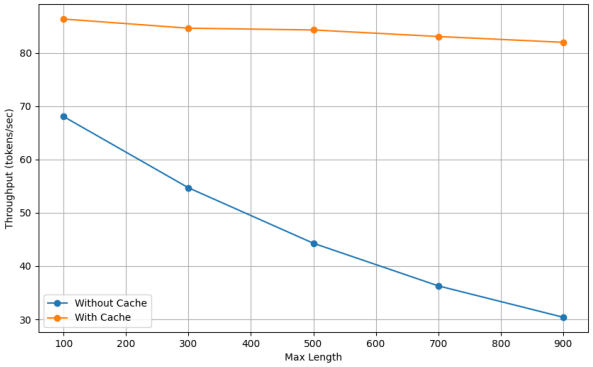
1 Task 1

1.1 KV Cache

This experiment is run on a single NVIDIA A40, using the GPT2 model loaded from Hugging Face.

The throughput (tokens/second) is measured for inferencing with and without KV caching across different generation lengths. For each length, 5 different prompts are used as input. The model generates text based on these prompts, and the average throughput is reported. Additionally, the peak memory usage is also reported, measured in a similar way.

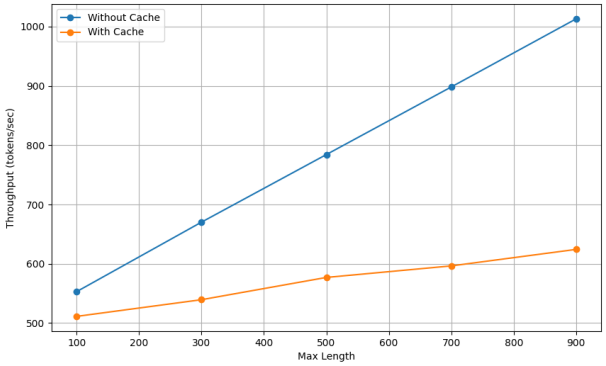
1.1.1 Throughput



According to the results, the throughput with cache is basically stable, with a slight, approximately linear decay, which is probably the consequence of the increasing time needed for calculating with the linearly expanding key and value tensors. Generally, it fits well with the theoretical results of KV cache, and it is way faster than inferencing without cache.

As for the thoughput without cache, it decays way faster due to the increasing computational complexity as the length grows. This rapid decay is expected because, without the cache, the model has to recompute the key and value tensors for each inference step, leading to significantly higher computational overhead. This behavior aligns with the theoretical expectations and highlights the efficiency gains provided by using the KV cache for inference.

1.1.2 Peak Memory Usage



The peak memory usage during inference without cache is larger than inferencing with cache, and it increases linearly as the length increases, probably due to the linear growth of the size of the hidden states with respect to the length.

The peak memory usage during inference with cache also grows linearly (due to the linearly growing KV cache) at a slower rate.

1.2 Quantization

This experiment is run on a single NVIDIA A40, using the `bigscience/bloom-560m` model loaded from Hugging Face, models are quantized using `optimum-quanto`.

1.2.1 Results

Each model with a certain quantization level is given 5 attempts to generate a sequence with maximum length of 500 tokens, the average throughput and model memory usage is reported. The results are as follows:

Quantization	None(fp32)	int8	int4	int2
Throughput(tokens/sec)	67.09	36.45	24.18	23.90
Model Memory Usage(MB)	2133.23	1294.17	1155.33	1071.33

1.2.2 Analysis

According to the results, we can see that as the model memory usage decreases as higher levels of quantization is applied to the model. However, the model size after quantizing it to int8 isn't 25% of the unquantized model size, which might be due to the fact that not all parts of the model are fully quantized. Additionally, quantization introduces some overhead due to the storage of scaling factors and offsets required for dequantization during inference. This overhead, along with partially quantized components, results in the quantized model size being larger than 25% of the original size when using int8 quantization, and probably the same logic can be extended to the case for int4 and int2.

Suprisingly, instead of getting faster inference, the throughput slows down substantially as the quantization level increases. This might be due to the additional computational overhead introduced by quantization during inference. Computation with lower precision may not be as well-optimized on standard GPUs designed primarily for 32-bit floating-point operations. Also, the quantized model require extra steps to convert low-precision weights back to higher precision during computation, resulting in lower efficiency. As a result, despite the smaller model size, the inference speed decreases with higher levels of quantization.

1.3 KV Cache Implementation

This experiment is run on a single NVIDIA A40, and the results are obtained by running `1_2/main.py`.

	With Cache(Customized)	Without Cache
Time(seconds)	9.91	13.77

With the customized cache, the inference time is reduced to 72% of the time needed to do the same inference without cache.

The main challenge was figuring out a way to tell whether the input comes from the same sequence that the model was previously working on or is it a new prompt, which means the cached contents should be renewed with the keys and values of this forward pass. The solution is caching the previous input and compare the current input with the previous input, and if the previous input is the prefix of the current input, the cache can be reused, if not, the cache should be cleared and renewed.

1.4 Prefix Caching

This experiment is run on a single NVIDIA A40, and the results are obtained by running `1_bonus/main.py`.

	With Cache(Customized)	Without Cache
Time(seconds)	7.29	14.01

With the customized cache, the inference time is reduced to 52% of the time needed to do the same inference without cache, which means a further 20% improvement in efficiency, compared to KV cache.

2 Task 2

In this task, I applied various reasoning techniques to improve the performance of deepseek-v2 on the test set of GSM8k.

2.1 Naive

The LLM is simply prompted to output the answer in a formatted manner.

```
instructions = """You are a helpful assistant that answers math questions.
The question is delimited by triple backticks. Return the final answer,
prefixed by 'Answer: ', ended with '.'.
"""
messages = [
    {"role": "system", "content": instructions},
    {"role": "user", "content": f"```{question}```"}
]

response = get_response(messages, temp=0)
```

2.2 CoT

In this experiment, I used **zero-shot** CoT. First, the LLM is prompted to generate a step-by-step reasoning process. Then, the original question and the generated answer is given to the LLM to extract the final answer.

```
# Reasoning extraction
instructions = """You are a helpful assistant that answers math questions.
The question is delimited by triple backticks. Return the step-by-step
reasoning and answer for the question. Provide a step-by-step reasoning
process leading to the final answer. Ensure that each step is clearly explained
and logically follows from the previous one.
"""
user_content = f"```{question}``` Let's think step by step."
messages = [
    {"role": "system", "content": instructions},
    {"role": "user", "content": user_content}
]
response = get_response(messages, temp=0)

# Answer extraction
instructions = """You are a helpful assistant that answers math questions.
You will be given a math question and its step-by-step reasoning process leading
to the final answer. Return the final answer, prefixed by 'Answer: ', ended with '.'.
"""
user_content += response
user_content += "What is the final answer?"
```

```
messages = [
    {"role": "system", "content": instructions},
    {"role": "user", "content": user_content}
]
response = get_response(messages, temp=0)
```

2.3 In-Context Learning

The LLM is given a correct question-answer pair from the training set, and a question from the test set. First, the LLM generates the answer with the correct question-answer pair as a reference. Then, the question and the generated answer is given to the LLM to extract the final answer.

```
# Generate similar answer with example
instructions = """You are a helpful assistant that answers math questions.
You will be given a correct question-answer pair followed by a new question
delimited by triple backticks. The correct question-answer pair is for reference.
Return an answer to the new question delimited by triple backticks.
"""
user_content = f"Question: {example['question']}\nAnswer: {example['answer']}\n```New question: {question}```"
messages = [
    {"role": "system", "content": instructions},
    {"role": "user", "content": user_content}
]
response = get_response(messages, temp=0)

# Answer extraction
instructions = """You are a helpful assistant that answers math questions.
You will be given a math question and its answer. Return the final numerical
answer, prefixed by 'Answer: ', ended with '.'.
"""
user_content = f"Question: {question}\nAnswer: {response}\nWhat is the final answer?"
messages = [
    {"role": "system", "content": instructions},
    {"role": "user", "content": user_content}
]
response = get_response(messages, temp=0)
```

2.4 Reflexion

According to the Reflexion paper, for the task of reasoning, the evaluator is a exact match grading function, instead of a LLM. Therefore there are only two LLMs, one for self-reflection, one as the actor. The actor uses zero-shot CoT to generate the answer, and the self-reflection LLM takes in the actor's output and its correctness to generate reflective texts that is used in the next round of generation.

Due to not having much free tokens left, the framework is only given 2 attempts for each question.

Because the prompts are quite a lot for this technique, please see the code in `reasoning_reflexion.py` for further information.

2.5 Results and Analysis

Technique	Naive	CoT	ICL	Reflexion
Accuracy	0.65732	0.94693	0.93935	0.96816

Clearly, the results of naive prompting is far worse than the other three, showing that the advanced techniques are effective, as they are claimed to be.

Among the three advanced methods, ICL and CoT achieve similar performance, with CoT leading by a slight margin of 0.76%, and Reflexion greatly surpasses the other two methods.

CoT instructs the LLM to reason step-by-step, which leads the model to generate detailed continuations. This approach helps the model break down complex problems into smaller, manageable steps, improving its performance on tasks that require logical reasoning and multi-step calculations, which makes it suitable for the math problems in GSM8k.

First, concerning the performance similarity between ICL and CoT, I think it is due to the fact that in GSM8k, the answers are given as a step-by-step procedure that leads to the final answer, quite similar to CoTs. Therefore, when using ICL, the LLM tries to imitate the reasoning process, resulting in a similar behaviour. An example is given in the following block to exhibit their similarity.

> ICL results:
> Janet's ducks lay 16 eggs per day.
> She eats 3 eggs for breakfast and uses 4 eggs for baking muffins, so she uses 3 + 4 = <<3+4=7>>7 eggs per day for herself.
> This leaves her with 16 - 7 = <<16-7=9>>9 eggs to sell at the farmers' market.
> She sells each egg for $2, so she makes 9 * 2 = <<9*2=18>>18 dollars every day at the farmers' market.
> #### 18
>
> CoT results:
> Let's break down the problem step by step to determine how much Janet makes every day at the farmers' market.
>
> ### Step 1: Calculate the total number of eggs Janet has each day.
> Janet's ducks lay 16 eggs per day.

> ### Step 2: Calculate the number of eggs Janet uses for herself.
> - Janet eats 3 eggs for breakfast every morning.
> - Janet bakes muffins with 4 eggs every day.
>
> So, the total number of eggs she uses for herself each day is:
> \[3 \text{ (for breakfast)} + 4 \text{ (for muffins)} = 7 \text{ eggs} \]
>
> ### Step 3: Calculate the number of eggs Janet sells at the farmers' market.
> To find out how many eggs she sells, we subtract the number of eggs she uses from the total number of eggs laid by her ducks:
> \[16 \text{ (total eggs)} - 7 \text{ (eggs used)} = 9 \text{ eggs} \]
>
> ### Step 4: Calculate the amount of money Janet makes from selling the eggs.
> Janet sells each egg for $2. Therefore, the total amount of money she makes from selling the eggs is:
> \[9 \text{ eggs} \times 2 \text{ dollars per egg} = 18 \text{ dollars} \]
>
> ### Final Answer:
> Janet makes **$18** every day at the farmers' market.

But still, CoT is slightly better than ICL, probably owing to the fact that it instructs the LLM to reason step by step explicitly, while ICL does this implicitly by giving few-shot examples, making it less clear that it needs to reason in coherent chains of thoughts, which can be seen in the previous example: there are explicit steps in the CoT result, while it is more implicit in the ICL result, and the reasoning clarity and quality is better in the CoT result.

At first, I expected that ICL should be far worse than CoT, because the methods of solving math problem may differ quite a lot on a problem-by-problem basis, while ICL is best for problems similar to provided examples, and it is beneficial when examples closely match the target question, which is not the case for such math problems. However, the detailed answers provided in GSM8k is surely a remedy for this issue, and made its performace comparable to CoT.

As for Reflexion, it is not suprising that its performance is better than CoT and ICL, because the actor itself is a LLM prompted using zero-shot CoT, and the LLM is given 2 attempts for each question. Reflexion is best for challenging problems where initial reasoning may be flawed and scenarios where accuracy is critical, this fits well with the math problems in GSM8k, resulting in a substantial improvement.

In Reflexion, 68 out of 1319 test cases failed at first attempt(94.8% accuracy, similar to CoT performance), 26 out of the 68 aforementioned cases succeeded at the second attempt, showing that self-reflection does help the LLM to improve its results.

2.6 Appendix

Here, I give an example where advanced prompting techniques led to better responses compared to naïve prompting.

Question:

Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How load does it take to download the file?

Naive(wrong):

Answer: 220 minutes.

CoT(correct):

Let's break down the problem step by step to determine how long it takes Carla to download the 200 GB file, considering the interruption due to the Windows restart.

Step 1: Calculate the initial download time before the interruption

Carla can download at a rate of 2 GB per minute. To find out how long it takes to download 40% of the 200 GB file, we first calculate 40% of 200 GB:

\[
0.4 \times 200 \text{ GB} = 80 \text{ GB}
\]

Next, we calculate the time it takes to download 80 GB at a rate of 2 GB per minute:

\[
\frac{80 \text{ GB}}{2 \text{ GB/minute}} = 40 \text{ minutes}
\]

Step 2: Account for the Windows restart

The Windows restart takes 20 minutes. Therefore, the total time spent so far is:

\[
40 \text{ minutes (initial download)} + 20 \text{ minutes (restart)} = 60 \text{ minutes}
\]

Step 3: Calculate the time to download the entire file after the restart

After the restart, Carla has to start the download from the beginning. The entire file is 200 GB, and she can download at a rate of 2 GB per minute. Therefore, the time to download the entire file is:

\[
\frac{200 \text{ GB}}{2 \text{ GB/minute}} = 100 \text{ minutes}

\]

Step 4: Calculate the total time including the initial download and the restart

The total time to download the file, including the initial download, the restart, and the subsequent download from the beginning, is:

\[
60 \text{ minutes (initial download + restart)} + 100 \text{ minutes (subsequent download)} = 160 \text{ minutes}
\]

Final Answer

It takes Carla a total of **160 minutes** to download the 200 GB file, considering the interruption due to the Windows restart.

ICL(correct):

Carla can download 2 GB per minute, so it would normally take her 200 GB / 2 GB/minute = <<200/2=100>>100 minutes to download the entire file.

However, 40% of the way through the download, Windows forces a restart, which takes 20 minutes. 40% of 200 GB is 0.4 * 200 GB = <<0.4*200=80>>80 GB. So, she had already downloaded 80 GB in 80 GB / 2 GB/minute = <<80/2=40>>40 minutes.

After the restart, she has to start the download from the beginning, which takes another 100 minutes.

Adding the time for the initial download, the restart, and the second download, the total time is 40 minutes (initial download) + 20 minutes (restart) + 100 minutes (second download) = <<40+20+100=160>>160 minutes.

160

Reflexion(correct):

First attempt(incorrect):

Let's break down the problem step by step to determine how long it takes Carla to download the 200 GB file, considering the interruption due to the Windows restart.

Step 1: Calculate the initial download time before the interruption

Carla can download at a rate of 2 GB per minute. To download the entire 200 GB file, the time required would be:

\\[\\text{Time} = \\frac{\\text{Total file size}}{\\text{Download rate}} = \\frac{200 \\text{ GB}}{2 \\text{ GB/minute}} = 100 \\text{ minutes} \\]

Step 2: Determine the point of interruption\nThe problem states that 40% of the way through the download, Windows forces a restart. To find out how much of the file was downloaded before the interruption:

\\[\\text{Downloaded before interruption} = 200 \\text{ GB} \\times 0.40 = 80 \\text{ GB} \\]

Step 3: Calculate the time taken to download 80 GB\nSince Carla can download 2 GB per minute, the time taken to download 80 GB is:

\\[\\text{Time to download 80 GB} = \\frac{80 \\text{ GB}}{2 \\text{ GB/minute}} = 40 \\text{ minutes} \\]

Step 4: Account for the time lost due to the restart

The restart takes 20 minutes. Therefore, the total time lost due to the interruption is:

\\[\\text{Time lost} = 20 \\text{ minutes} \\]

Step 5: Calculate the total time to download the file after restarting from the beginning

Since Carla has to restart the download from the beginning, she will need to download the entire 200 GB again. The time required to download the entire file is:

\\[\\text{Time to download 200 GB} = 100 \\text{ minutes} \\]

Step 6: Add the time lost due to the interruption

The total time to download the file, including the time lost due to the restart, is:

\\[\\text{Total time} = \\text{Time to download 200 GB} + \\text{Time lost} = 100 \\text{ minutes} + 20 \\text{ minutes} = 120 \\text{ minutes} \\]

Final Answer:

It takes Carla a total of **120 minutes** to download the 200 GB file, considering the interruption due to the Windows restart.

Second attempt(correct):

Let's break down the problem step by step to determine how long it takes Carla to download the 200 GB file, considering the interruption due to the Windows restart.

Step 1: Calculate the initial download time before the interruption

Carla can download at a rate of 2 GB per minute. To download the entire 200 GB file, the time required would be:
\[\text{Time} = \frac{\text{Total file size}}{\text{Download rate}} = \frac{200 \text{ GB}}{2 \text{ GB/minute}} = 100 \text{ minutes} \]

Step 2: Determine the point of interruption

The problem states that 40% of the way through the download, Windows forces a restart. To find out how much of the file was downloaded before the interruption:
\[\text{Downloaded before interruption} = 200 \text{ GB} \times 0.40 = 80 \text{ GB} \]

Step 3: Calculate the time taken to download 80 GB before the interruption

Using Carla's download rate of 2 GB per minute:
\[\text{Time to download 80 GB} = \frac{80 \text{ GB}}{2 \text{ GB/minute}} = 40 \text{ minutes} \]

Step 4: Account for the time taken by the Windows restart

The Windows restart takes 20 minutes.

Step 5: Calculate the total time taken if the download had to restart from the beginning

Since the download restarts from the beginning after the 20-minute restart, Carla has to download the entire 200 GB file again. The time required for this is:
\[\text{Time to download 200 GB again} = 100 \text{ minutes} \]

Step 6: Sum up all the time components

The total time taken includes:

1. The initial 40 minutes to download 80 GB.
2. The 20-minute Windows restart.
3. The 100 minutes to download the entire 200 GB file again.

\[\text{Total time} = 40 \text{ minutes} + 20 \text{ minutes} + 100 \text{ minutes} = 160 \text{ minutes} \]

Final Answer:

It takes Carla a total of **160 minutes** to download the 200 GB file, considering the interruption and the need to restart the download from the beginning.