DL-NLP Assignment #3
[Tianle Chen] [2200017841]

To run the code, please refer to the README.md file in the code directory.

1 Task1

1.1 environment

I conduct the experiment on my PC. The envionment is as following:

Hardware environment	
CPU	12th Gen Intel(R) Core(TM) i7-12700H
GPU	NVIDIA GeForce RTX 3060 Laptop CPU
Software environment	
python	3.9.1
cuda	12.4
torch	2.4.1+cu124

1.2 Task1.1

In this section, I use GPT-2 as the base model, the same dataset in Task1.2 (provided in the code base) to conduct the experiment. The batchsize is setting to 16 and I use nvitop for memory cost estimation.

For the part of KV-Cache, I simply utilize the KV-Cache realized in the transformers library. And for the part of quantization, I utilize gptq library[Frantar et al., 2023], which could perform 4-bit, and specially, 3-bit quantization. I perform static quantization on the dataset, and then test the performance of the quantized model on the same dataset.

The result I get is as following:

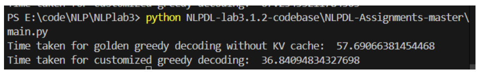
Method	Memory Cost (MiB)	Tokens	Time Cost (s)	Throughout (tokens/s)
Raw inference	820	8796	71.3	123.4
KV-Cache	740	8796	65.3	134.5
4-bit quantization	521	8702	133.3	65.26
3-bit quantization	332	8796	269.1	32.68

Analysis From the result we can find that utilizing KV-Cache, can speed up the inference at about 10% and save cuda memory at about 10% (Might because less memory cost cause by calculate extra quiries). And the quantization slow down the inference (might because the GPU have less modules for integer arithmetic), but can obviously save cuda memory, while the result (8:5:3) is not match expectations (16 bit :4 bit :3 bit), which might because the cost of the data and the parameters used for quantization cannot be ignored compare with the relatively small model size.

1.3 Task1.2

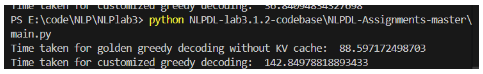
Method In this section, I realize KV-cache based on the code base. The method is relatively simple: in the self-regression progress, just save the key and values and utilize them in the new iteration. For the details, please refer to the code.

Experiment Might because of my environment settings, I go through some bug while I work on this part. I find that the original code base is able to be ran, while the result shows that the custom version is faster than the original one (even I haven't done anything yet). After communicating with some other fellow students [1], we suppose this might because of cuda or something in the environment save some cache silently. After adding "torch.cuda.empty_cache()" before each time calling the inference function, the result seems to be reasonable:



```
PS E:\code\NLP\NLPlab3> python NLPDL-lab3.1.2-codebase\NLPDL-Assignments-master\
main.py
Time taken for golden greedy decoding without KV cache:  57.69066381454468
Time taken for customized greedy decoding:  36.84094834327698
```

(a) Original Code Base

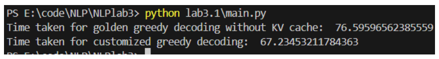


```
PS E:\code\NLP\NLPlab3> python NLPDL-lab3.1.2-codebase\NLPDL-Assignments-master\
main.py
Time taken for golden greedy decoding without KV cache:  88.597172498703
Time taken for customized greedy decoding:  142.84978818893433
```

(b) After adding empty cache lines

Figure 1: **Left** The results of running the original code base shows that the custom version is about twice faster than the original version. **Right** And after adding lines in the code to empty the cache of cuda, the custom version is slower than the original version, which is more reasonable. **Analysis** Specially, both of them is slower than the result in Task1.1, I suppose that might because the GPU memory contains two models at the same time, the left memory for inference is lower than Task1.1, therefore the inference is slower

After the modification mentioned above, I test my KV-cache, the result shows in the follow figure, which report a similar accelerating rate in the Task1.1:



```
PS E:\code\NLP\NLPlab3> python lab3.1\main.py
Time taken for golden greedy decoding without KV cache:  76.59596562385559
Time taken for customized greedy decoding:  67.23453211784363
```

Figure 2: The result of task1.2 after modification

[1]Thanks to Wenkai Yv and Dayuan Zhao for discussion on this problem

2 Task2

2.1 Experiment Setting

As for the reasoning task and dataset, I choose gsm8k as the dataset, and due to the limitation of time and budget, I only used the first 1000 data points of the test set of that dataset. As for the model, I simply choose 'deepseek-chat' for this task. As for the code-base, to speed up inference, I utilizing an open-source code base with ray-parallelism, and modified it for our assignment. For the details of the code, please refer to the README.md.

2.2 Method and Result

As for the method used to enhance the reasoning ability of the LLM, I realize five different method as following:

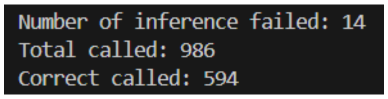
Plain IO Plain IO is the most naive LLM reasoning framework. It's just simply formatting the problem in certain format before pass it to the LLM. the chatting template for plain IO that I used is as following (The question is replaced by the question in gsm8k). And I simply use python build-in string tools to get the answer from the response of the LLM.

System prompt: "'You are a helpful assistant for math problems."'

User prompt:"'Here is the math problem you need to solve: [[QUESTION]]: question [[END]],

Please provide the answer in the following format, the answer should be a number: [[ANSWER]]: <Your answer>[[END]]"'

The result of Plain IO on the 1000 data point is 0.594, and among the wrong cases, 14 cases is wrong due to the incorrect output format. The raw result is showing as following:



```
Number of inference failed: 14
Total called: 986
Correct called: 594
```

Figure 3: The result of Plain IO

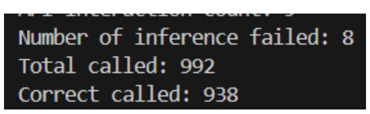
CoT CoT is the method that ask LLM to think step by step. To realize CoT, I simply ask the LLM to 'think step by step' and output its reasoning process explicitly. The chatting template for CoT that I used is as following (The question is replaced by the question in gsm8k). And I simply use python build-in string tools to get the answer from the response of the LLM.

System prompt: “‘You are a helpful assistant for math problems.”’

User prompt:“‘Here is the math problem you need to solve: [[QUESTION]]: question [[END]],

Please think step by step to solve this problem, and provide the answer in the following format, the answer should be a number: [[REASONING]]: <Your reasoning>[[END]], [[ANSWER]]: <Your answer>[[END]]”’

The result of Plain IO on the 1000 data point is 0.938, and among the wrong cases, 8 cases is wrong due to the incorrect output format. The raw result is showing as following:



```
Number of inference failed: 8
Total called: 992
Correct called: 938
```

Figure 4: The result of CoT

In-context learning To realize in-context learning, I simply add an example (Not included in the test dataset) before the chatting template of CoT, and other settings are totally same as the CoT. The chatting template for in-context learning that I used is as following (The question is replaced by the question in gsm8k). And I simply use python build-in string tools to get the answer from the response of the LLM.

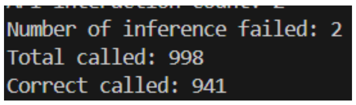
System prompt: “‘You are a helpful assistant for math problems.”’

User prompt:“‘Here is an example of solving a math problem: [[QUESTION]]: The price of Parmesan cheese is $11 per pound. Mozzarella cheese is $6 per pound. Amor buys 2 pounds of Parmesan and 3 pounds of mozzarella cheese. Is she starts with $50 cash, how much money, in dollars, will she have left to buy meat? [[END]], [[REASONING]]: The cost for buying 2 pounds of parmesan cheese is $11 x 2 = $«11*2=22»22.cost for buying 3 pounds of mozzarella cheese is $6 x 3 = $«6*3=18»18.total amount spent for the 2 kinds of cheese is $22 + $18 = $«22+18=40»40., Amor still has $50 - $40 = $«50-40=10»10 left to buy for meat.[[END]], [[ANSWER]]: 10[[END]]

Please follow the example to solve the following math problem: Here is the math problem you need to solve: [[QUESTION]]: question [[END]],

Please think step by step to solve this problem, and provide the answer in the following format, the answer should be a number: [[REASONING]]: <Your reasoning>[[END]], [[ANSWER]]: <Your answer>[[END]]”’

The result of in-context-learning on the 1000 data point is 0.941, and among the wrong cases, 2 cases is wrong due to the incorrect output format. The raw result is showing as following:



```
Number of inference failed: 2
Total called: 998
Correct called: 941
```

Figure 5: The result of In-Context Learning

Reflexion To realize reflextion, I ask the LLM to perform a CoT reasoning at first, and if the answer of LLM is wrong, I offer it with its old reasoning and answer, asking it to reflect on its previous answer and get a new answer. The chatting template for reflextion that I used is as following (The question is replaced by the question in gsm8k, the reasoning and answer is replaced by LLM'r response to the CoT prompt). And I simply use python build-in string tools to get the answer from the response of the LLM.

System prompt: "'You are a helpful assistant for math problems."'

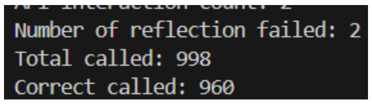
User prompt:"'Here is the math problem you need to solve: [[QUESTION]]: question [[END]],

You have tried to solve this problem before, the answer you provided is: [[REASONING]]: reasoning[[END]], [[ANSWER]]: answer[[END]]

But the answer is not correct, please try again.

Please reflect on your previous answer, and think step by step to solve this problem, and provide the answer in the following format, the answer should be a number: [[REFLECTION]]: <Your reflection>[[END]], [[REASONING]]: <Your reasoning>[[END]], [[ANSWER]]: <Your answer>[[END]]"'

The result of reflextion on the 1000 data point is 0.960, and among the wrong cases, 2 cases is wrong due to the incorrect output format. The raw result is showing as following:



```
Number of reflection failed: 2
Total called: 998
Correct called: 960
```

Figure 6: The result of Reflextion

Modified Reflexion This part is extraly done by myself. Actually I am working on studying LLM reasoning recently. Some up-to-date research reviews several methods for enhancing LLM's reasoning abilities (including reflextion) and doubts their experiment

[Huang et al., 2024]. The main concern can be concluded as: the process of reflexion actually leaks the information of the questions answer to the LLM through the evaluator (As is said in the paper of reflexion[Shinn et al., 2023], they use exact match (EM) as evaluator for reasoning tasks). To be extremely, if we write a loop to output range(1,10000), and each time when its output doesn't match the answer, we ask it to 'reflect' on its answer and get a new one. Then after upper to 10000 times 'reflexion', our 'model' would reach a relativly high accuracy on dataset gsm8k with extremely small parameters.

Therefore, follow the experiment in [Huang et al., 2024], I conduct an experiment of 'modified reflexion'. The main idea is that: for the first time, I ask the LLM to reasoning according to the CoT prompt. And no matter the answer is right or wrong, I always ask the LLM to reflect on its reasoning, if it is right, simply keep it, and if it is wrong, modify it. The chatting template for modified reflextion that I used is as following (The question is replaced by the question in gsm8k, the reasoning and answer is replaced by LLM'r response to the CoT prompt). And I simply use python build-in string tools to get the answer from the response of the LLM.

System prompt: "'You are a helpful assistant for math problems."'

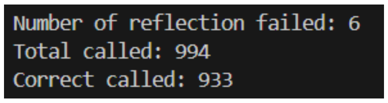
User prompt:"'Here is the math problem you need to solve: [[QUESTION]]: question [[END]],

You have tried to solve this problem before, the answer you provided is: [[REASONING]]: reasoning[[END]], [[ANSWER]]: answer[[END]]

We don't know whether the answer is correct or not, please review your answer, if it is correct, please provide the answer in the following format; if it is not correct, please rethink and provide the answer in the following format.

Please reflect on your previous answer, and think step by step to solve this problem, and provide the answer in the following format, the answer should be a number: [[REFLECTION]]: <Your reflection>[[END]], [[REASONING]]: <Your reasoning>[[END]], [[ANSWER]]: <Your answer>[[END]]"'

The result of modified-reflextion on the 1000 data point is 0.933, and among the wrong cases, 6 cases is wrong due to the incorrect output format, which agree with the idea mentioned in[Huang et al., 2024]. The raw result is showing as following:



```
Number of reflection failed: 6
Total called: 994
Correct called: 933
```

Figure 7: The result of Modified Reflextion

Method	Correct Count	Accuracy	Fail cases count
Plain IO	594	0.594	14
CoT	938	0.938	8
In-Context Learning	941	0.941	2
Reflexion	960	0.96	2
Modified-Reflexion	933	0.93	6

2.3 Result

The overall result is as following: The results shows the impressive performance of the CoT method. And the In-context learning built on CoT can lead to higher accuracy and reduce the fail cases due to the wrong output formation effectively. While the reflexion reported the highest accuracy, the added experiment modified reflexion shows that the reflexion method contains some problem. And in the real world experinment without a powerful evaluator, it could perform no so good. (While in the cases that can easily find an evaluator such as code writing, reflexion is still powerful).

References

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL `https://arxiv.org/abs/2210.17323`.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL `https://arxiv.org/abs/2310.01798`.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL `https://arxiv.org/abs/2303.11366`.