



## NLP-DL Assignment 3 Report

Jiahao Li

2300017834

Yuanpei College

Peking University

2300017834@stu.pku.edu.cn

### 1 Task 1

#### 1.1 Part 1

In part 1, I implemented a pipeline to evaluate throughput and memory usage among: a naïve (baseline) LLM inference implementation, an implementation that utilizes the KV-cache directly from HuggingFace Transformers, an implementation that utilizes the KV-cache and int4 quantization directly from HuggingFace Transformers, and an implementation that utilizes the KV-cache and int2 quantization directly from Huggingface Transformers. The base model is gpt2, and the dataset for inference is that provide in part 2. Moreover, my experiments ran on a single RTX 4090.

Here is the results and settings(hyperparameters not listed are as default):

Table 1: Quantization and Throughput Comparison Results

Quantization Type	Configuration	Generate Speed (tokens/s)	GPU Memory Cost (GB)
BF16	No KV Cache	7.8998	14.9942
	With KV Cache	48.1328	16.4064
INT4	With KV Cache	7.2054	7.7318

Table 2: Experiment Settings Summary

Parameter	Value
Model	Qwen-7B-chat
Context Length per Experiment	1
Generate Length per Experiment	2048
Use Flash Attention	False

From the results, we can discover that quantization significantly impacts the trade-off between memory and speed by reducing model size and computational complexity. Lower-precision formats like INT4 decrease memory usage, enabling deployment on resource-constrained devices, while also accelerating inference by requiring fewer arithmetic operations. However, this comes at the cost of potential accuracy degradation, requiring careful tuning to balance performance and efficiency.

#### 1.2 Part 2

In part 2, a minimalistic KV-cache mechanism was implemented based on the codebase provided and the performance improvement was listed below(the batchsize is set to 1, the GPU is RTX 4090):

Table 3: Comparison of Time Taken for Golden Greedy and Customized Greedy Decoding

Decoding Method	Time (seconds)
Golden Greedy Decoding(no KV-cache)	10.87
Customized Greedy Decoding(with KV-cache)	6.35

In the process of implementing the minimalistic KV-cache mechanism, several challenges were encountered, particularly regarding how to integrate the code from transformers to the existing codebase. Besides, the initial unsatisfactory performance improvement was also a tricky challenge, but after examining carefully, the reason behind was discovered and I fixed the code logic so that the KV-cache could be fully leveraged. Despite these challenges, the KV-cache significantly improved performance, reducing the decoding time for Customized Greedy Decoding from 10.87 seconds (without cache) to 6.35 seconds (with cache), demonstrating its effectiveness in optimizing the decoding process while maintaining accuracy.

## 2 Task 2

In task 2, I compared four prompt techniques: naive, chain of thought, few-shot, and reflexion. To be specific, I tried them on gsm8k dataset(reflexion technique on a one-tenth subset due to limited API access), the accuracy of each technique is shown below:

Table 4: Comparison of accuracy among different prompt techniques

Prompt Method	Accuracy(%)
naive prompt	58.68
CoT	90.14
few-shot	90.90
reflexion	93.10

Further, I collected some cases to demonstrate the performance differences in the following. the table is different answers to the question "Kylar went to the store to buy glasses for his new apartment. One glass costs \$5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?", to which the correct answer is 64. The results are shown in Sec. 2

More specific configurations, such as prompts for each technique or the implementation of reflexion can be found in my repository. Now I would analyze the impact of each reasoning technique, highlighting cases where each technique was most beneficial. naive prompting is best for simple tasks, CoT excels in multi-step reasoning, few-shot improves generalization through examples, and reflexion enhances accuracy via iterative refinement. Each technique is most beneficial in specific scenarios, such as CoT for logical puzzles, few-shot for text classification, and reflexion for tasks requiring iterative refinement like code generation.

Table 5: Comparison of different answers among different prompt techniques

Prompt Method	Answer
naive prompt	#### 76
CoT	<p>Let's break down the problem step by step:</p> <ol style="list-style-type: none"> <li>1. <b>Cost of the first glass:</b> The first glass costs \$5.</li> <li>2. <b>Cost of every second glass:</b> Every second glass costs 60% of \$5, which is <math>0.6 \times 5 = 3</math> dollars.</li> <li>3. <b>Number of glasses:</b> Kylar wants to buy 16 glasses.</li> <li>4. <b>Pattern of costs:</b> - The first glass costs \$5. - The second glass costs \$3. - The third glass costs \$5. - The fourth glass costs \$3. - This pattern continues until the 16th glass.</li> <li>5. <b>Total cost calculation:</b> - There are 8 pairs of glasses (since <math>16 \div 2 = 8</math>). - Each pair consists of one \$5 glass and one \$3 glass. - The cost for each pair is <math>5 + 3 = 8</math> dollars. - The total cost for 8 pairs is <math>8 \times 8 = 64</math> dollars.</li> </ol> <p>Therefore, Kylar needs to pay:</p>
few-shot	<p>#### 64</p> <p>Kylar wants to buy 16 glasses. Every second glass costs 60% of the price, which is <math>\\$5 * 0.6 = \\$3</math>.</p> <p>For 16 glasses, there will be 8 pairs of glasses. The first glass in each pair costs \$5, and the second glass in each pair costs \$3.</p> <p>So, the total cost for the 16 glasses is: <math>(8 * \\$5) + (8 * \\$3) = \\$40 + \\$24 = \\$64</math></p>
reflexion	<p>#### 64</p> <p>To determine the total cost for Kylar to buy 16 glasses, we need to consider the pricing structure. The first glass costs \$5, and every second glass costs 60% of \$5, which is \$3.</p> <p>For the first 16 glasses, the pattern of costs will be: \$5 for the first glass, \$3 for the second glass, \$5 for the third glass, \$3 for the fourth glass, and so on. This pattern repeats every two glasses.</p> <p>Since Kylar is buying 16 glasses, there will be 8 pairs of glasses. Each pair consists of one glass at \$5 and one glass at \$3, totaling \$8 per pair.</p> <p>Therefore, the total cost for 8 pairs is <math>8 \text{ pairs} * \\$8 \text{ per pair} = \\$64</math>.</p> <p>Thus, Kylar needs to pay \$64 for 16 glasses. #### \$64</p>