DL-NLP Homework 3

Junting Zhou

December 9, 2024

Abstract

This report explores two critical aspects of enhancing large language model (LLM) performance. First, inference optimization techniques, including KV-cache and quantization, are evaluated for throughput and memory efficiency. Second, advanced prompting methods, such as Chain of Thought (CoT), In-Context Learning (ICL), and Reflexion, are analyzed for their impact on accuracy and reasoning quality using GSM8k datasets. All the code about this homework can be found at:https://github.com/zjtPKU/nlp-hw3

1 Introduction

Efficient LLM performance is crucial for practical deployment, requiring optimizations in both inference and reasoning capabilities. This report examines:

- **Task 1:** Optimizing LLM Inference using KV-cache and Quantization
- **Task 2:** Enhancing Reasoning through Advanced Prompting Techniques

2 Task1:LLM inference Acceleration

2.1 Baseline vs. KV-cache Inference

Technique	**Avg Time (s)**	**Tokens/sec**	**GPU Memory (Allocated/Reserved MB)**
Baseline	1.3932	68.6387	496.22 / 959.23
KV-cache	0.7364	169.1979	917.31 / 964.00

Table 1: Performance comparison of baseline and KV-cache inference on NVIDIA A40 GPU.

2.1.1 Analysis of Results

Throughput: The KV-cache implementation achieves a significant improvement in inference throughput, increasing from 68.64 tokens/second to 169.20 tokens/second—a 146.4% improvement. This is attributed to the reduction in redundant computations during sequential token generation.

Inference Time: Average inference time per task is reduced by 47.1%, from 1.3932 seconds to 0.7364 seconds, highlighting the efficiency gains from using cached key-value pairs.

GPU Memory Utilization: Despite the increase in throughput, GPU memory usage remains efficient. The allocated memory increases from 496.22 MB to 917.31 MB, and reserved memory increases slightly from 959.23 MB to 964.00 MB, which is within acceptable limits for practical deployments.

2.2 Customized KV-Cache Results

2.2.1 Baseline vs. KV-cache Implementation

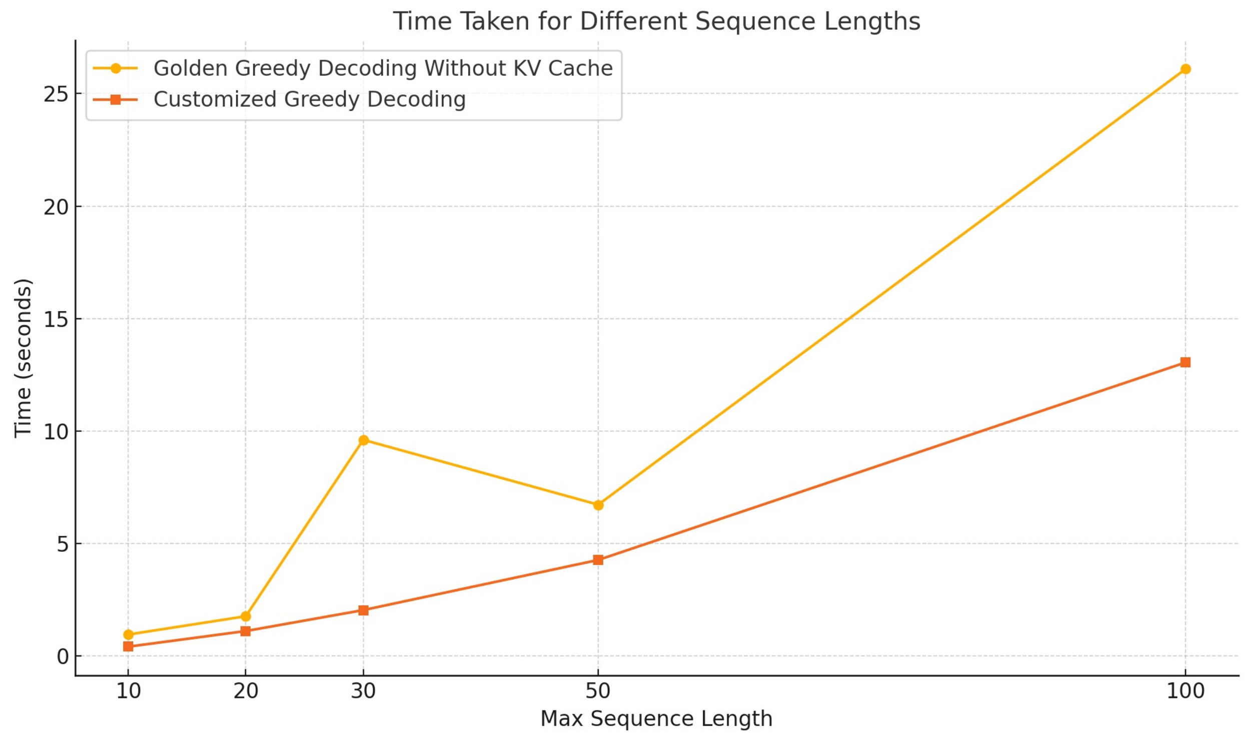

Figure 1: Customized-KVcache Performance Comparison

The graph in Figure 1 demonstrates the performance of the customized-KVcache method. As the sequence length increases, the time efficiency improves significantly due to the caching of intermediate key and value states. This method eliminates redundant calculations, especially in transformer models where intermediate states are recalculated for each step. For shorter sequence lengths, the time savings are less pronounced, as the overhead of maintaining the cache outweighs the benefits. However, for longer sequences, the KVcache drastically reduces inference time.

2.3 Quantization Experiment Results

In this section, we present the results of the performance evaluation for different quantization methods: FP32, FP16, INT8, and INT4. The key metrics of computation time and memory usage are summarized as follows:

Quantization Method	Computation Time (seconds)	Memory Usage (MB)
FP32 (32-bit Floating Point)	3.11	2237.19
FP16 (16-bit Floating Point)	1.67	1652.36
INT8 (8-bit Integer)	5.30	1652.47
INT4 (4-bit Integer)	2.77	1652.01

Table 2: Summary of Computation Time and Memory Usage for Different Quantization Methods

2.3.1 Analysis of Performance and Memory Utilization for INT8 and INT4

1. Time Costs Although INT8 and INT4 are expected to reduce time costs by using lower precision, the results indicate no significant reduction in time. This is likely due to several factors, such as insufficient hardware support for low-precision computations. Many GPUs are optimized for higher precision formats like FP32 or FP16, and do not have dedicated hardware acceleration for INT8 or INT4 operations. Furthermore, lower precision may introduce additional operations for error correction or rounding, which can negate the expected time savings.

2. Memory Usage As for memory, the reduction in memory usage from using INT8 and INT4 was less than expected. This is due to several reasons, such as the lack of optimization for low-precision formats. Some models may still store intermediate results in higher precision, or the system may need additional memory buffers for lower precision computations, limiting the overall memory savings. Moreover, the increased memory access costs or the need for error-correction data could offset the memory benefits of using INT8 and INT4.

3 Task 2: Advanced Prompting Techniques

3.1 Techniques Evaluated

- **Naïve Prompting:** Direct task instructions without reasoning scaffolds.
- **Chain of Thought (CoT):** Step-by-step reasoning in prompts.
- **In-Context Learning (ICL):** Providing examples for task guidance.
- **Reflexion:** Iteratively refining answers based on feedback.

3.2 Results

Technique	GSM8k Accuracy (%)
Naïve Prompting	43.06
CoT	94.01
ICL	93.10
Reflexion	92.95

Table 3: Performance comparison of prompting techniques.

4 Discussion

4.1 Inference Optimization

KV-cache significantly improves throughput by reducing redundant computations, while quantization offers a trade-off between memory usage and model precision. The custom KV-cache implementation demonstrated incremental gains.

4.2 Advanced Prompting Techniques

Reflexion and CoT excel in tasks requiring iterative or logical reasoning. In-Context Learning is effective when high-quality examples are available but lacks dynamic refinement.

5 Conclusion

The combined experiments demonstrate that:

- KV-cache and quantization optimize inference efficiency with minimal trade-offs.
- Reflexion provides the most significant reasoning improvement among advanced prompting techniques.
- Both tasks complement each other in enhancing LLM applications across diverse metrics.

A Prompt Tuning Examples

A.1 Naive Prompt

```
prompt = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": f"For each question, show the answer started with \"####
    Make sure that the answer is only a number without any other words!
    Question:
    {question}\n"}
]
```

A.2 ICL (In-context Learning) Prompt

```
prompt = [
    {"role": "system", "content": "You are a helpful assistant who solves math problems
    {"role": "user", "content": f"""
        For each question, provide a detailed solution. Here are some examples:

        ### Example 1:
        Question:
        Alice has 3 apples. She buys 2 more apples and gives away 1 apple. How many appl

        Step-by-step solution:
        1. Alice initially has 3 apples.
        2. She buys 2 more apples, so she has 3 + 2 = 5 apples.
        3. She gives away 1 apple, leaving her with 5 - 1 = 4 apples.

        Final Answer: #### 4

        Now, solve the following question.
        Question:
        {question}

        For each question, provide a detailed solution. Then, show the answer started wi
        Make sure that the answer is only a number without any other words!\n
    """}
]
```

A.3 CoT (Chain of Thought) Prompt

```
prompt = [
    {"role": "system", "content": "You are a helpful assistant who solves math problems
    {"role": "user", "content": f"For each question, provide a detailed solution.
    Then, show the answer started with \"####\".
    Make sure that the answer is only a number without any other words!
    Question:
    {question}\n"}
]
```

A.4 Reflection Prompt

```
initial_prompt = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": f"""
        For each question, provide a detailed solution. Then, show the answer started wi
        Make sure that the answer is only a number without any other words!
        Question:
        {question}
    """}
]

reflection_prompt = [
    {"role": "system", "content": "You are a reflective assistant who verifies and impro
    {"role": "user", "content": f"""
        Question:
        {question}
        Here is the initial solution I provided:
        {initial_answer}

        Reflect on the solution. Are there any errors or areas for improvement?
        - If any issues are found, provide a corrected solution.
        - If the solution is correct, confirm that it is accurate.

        Provide a detailed solution. Then, show the answer started with \"####\".
        Make sure that the answer is only a number without any other words!
    """}
]
```

B Hardware Conditions for KV-cache Experiments

The KV-cache related experiments were conducted on the following hardware setup:

Specification	**Details**
GPU Name	NVIDIA A40
Driver Version	560.28.03
CUDA Version	12.6

Table 4: Hardware setup used for KV-cache experiments.