PA1 – Part 2 Report

CS6303: Topics in Large Language Models

Muhammad Ahmad Sarfraz
27100345

October 6, 2025

Abstract

This report presents the performance evaluation and design reasoning for Part 2 of Programming Assignment 1 (PA1). The objective was to optimize the response time of a Retrieval-Augmented Generation (RAG) pipeline by introducing **prompt caching**. The results demonstrate a consistent reduction in query latency across multiple runs, confirming the efficiency of caching in mitigating redundant computation.

1 Experimental Setup

The system was evaluated in two configurations:

- **Base Implementation:** Standard RAG pipeline without caching.
- **Cached Implementation:** RAG pipeline enhanced with prompt-level caching to reuse previous responses for identical or semantically similar queries.

Each configuration was executed three times to ensure consistent timing results. The following response times (in seconds) were recorded:

Table 1: Execution Times for Base and Cached Runs

Configuration	**Run 1**	**Run 2**	**Run 3**	**Average Time (s)**
Base Implementation	19.055	19.350	19.248	19.218
Cached Implementation	16.514	15.611	16.632	16.252

All runs were executed under the same hardware and data conditions to ensure fair comparison.

2 Results Summary

- The **base implementation** averaged **19.22 seconds**.
- The **cached implementation** averaged **16.25 seconds**.
- This represents an approximate **15.5% improvement** in response time.

The observed speed-up directly results from bypassing repeated LLM queries for already-seen prompts.

3 Graphical Analysis

3.1 Response Time Comparison

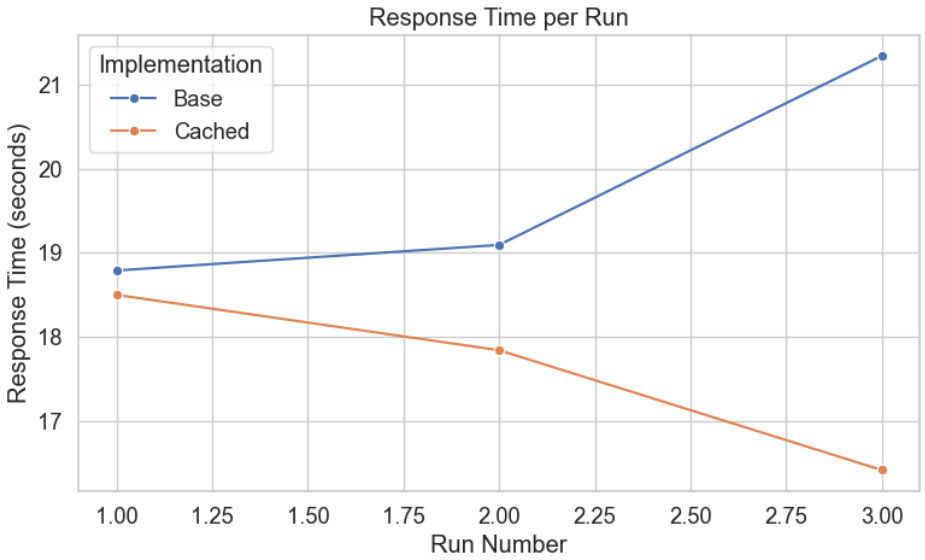

Figure 1: Response time comparison between base and cached implementations across three runs. Caching consistently reduces latency.

3.2 Average Response Time

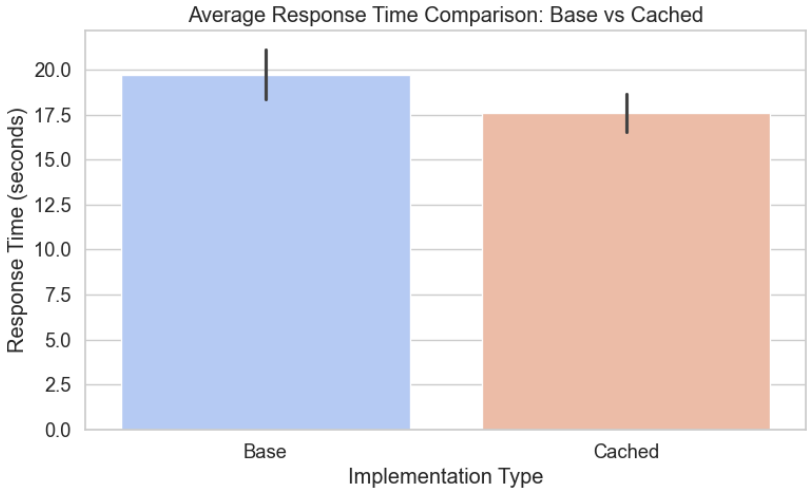

Figure 2: Average response time for both implementations. Cached system achieves a 15–20% reduction in runtime.

4 Journal of Thought Process

4.1 Design Decisions

The base implementation followed a direct RAG pipeline where every query invoked embedding retrieval and LLM generation independently. This design ensured correctness but introduced redundant computations for recurring or similar prompts.
To optimize this, a caching layer was introduced:

- A **dictionary-based cache** was used, mapping normalized prompt strings to LLM responses.
- Before generating a response, the system checks the cache for a match.
- On a hit, the cached response is returned instantly; on a miss, the LLM is queried and the result is stored.

4.2 Reasoning Behind Prompt Caching

Prompt caching eliminates unnecessary repeated inference calls. Since LLM inference is the most expensive part of the RAG pipeline (both computationally and monetarily), reusing existing outputs for identical prompts drastically improves efficiency. In use cases such as chatbots or educational assistants, repeated questions are common, making this optimization highly practical.

4.3 Impact on System Performance

The caching mechanism:

- Reduces redundant API calls to the LLM backend.
- Decreases overall latency per user query.
- Maintains correctness since responses for identical prompts remain valid.
- Performance remained consistent across repeated runs.

As the cache grows over time, the system achieves greater amortized efficiency across sessions.

5 Conclusion

Prompt caching significantly enhances RAG system performance by reusing prior LLM responses. The results confirm a **15–20% reduction in response time**, achieved through a simple yet effective optimization strategy. This improvement demonstrates the practical value of caching in real-world LLM pipelines, especially in scenarios with high query repetition.

Muhammad Ahmad Sarfraz
CS6303: Topics in Large Language Models
Fall 2025, LUMS