Advanced LLM Caching with RAG

Group members: Alexander Ding, Yuxuan Liao, Weili Shi, Jialiang Zhou

Motivation and Description

Background and Related Work

Large Language Models (LLM) such as GPT-4 performs well in a number of tasks such as question answering, but is also very costly to use. Each invocation of the LLM costs money. For example, OpenAI's GPT-4 costs $0.01 per 1,000 input tokens and $0.06 per output tokens. Retrieval Augmented Generation (RAG) helps with the inference of LLM by retrieving relevant document chunks from a document database and prepends these chunks to the prompt, increasing the cost even more. There have been recent work on cost-saving techniques on the use of LLMs.

- FrugalGPT shows that it is possible to use smaller and cheaper language models (LM), such as a locally-hosted GPT-J, to achieve similar performance as GPT-4 for particular tasks [3].
- GPTCache attempts to reduce LLM calls by caching previous responses from the LLM, and returning these responses directly if determined to be similar [2]. However, whether the cache is a hit is directly determined by the similarity score of the embeddings, although semantically similar sentences might not be able to answer the question correctly. Also, there might be answers in the cache that could correctly answer the question, but are discarded due to the similarity score being larger then the threshold.

For this project, we first limit ourselves to question answering tasks.

Motivation

Based on the below observations, we plan to propose a system with better caching in the context of RAG with LLMs, to save cost and reduce latency by reducing LLM calls, and improve response quality compared to naïve caching.

- *Observation 1*: LLM responses might contain knowledge/information different from the one directly obtained from RAG documents.
- *Observation 2*: within a single session, it is common for users to ask related questions.

Therefore, we ask the following questions:

- *Question 1*: Are there fundamental differences in the content of the cache and RAG documents? In other words, does LLM generate more knowledge from user questions compared to the RAG documents? If so, is caching previous LLM responses helpful for the local LLM to answer future questions?
- *Question 2*: Are there patterns in users' question-answering workloads that allow useful prediction for future questions? If so, can prefetching help?

System Description and Challenges

The system is composed of the following components: an LLM (such as GPT-4), a vector store storing documents that are considered to be the truth, a semantic cache similar to GPTCache, and a local LM for synthesizing a better response for the user, as shown in the figure below. When a user query comes in, the system performs a similarity search in both the documents and the cache for the top results, and compose a query to the local LM. If the local LM successfully answers the question, the answer is returned to the user. If not, invoke the LLM for a better response.

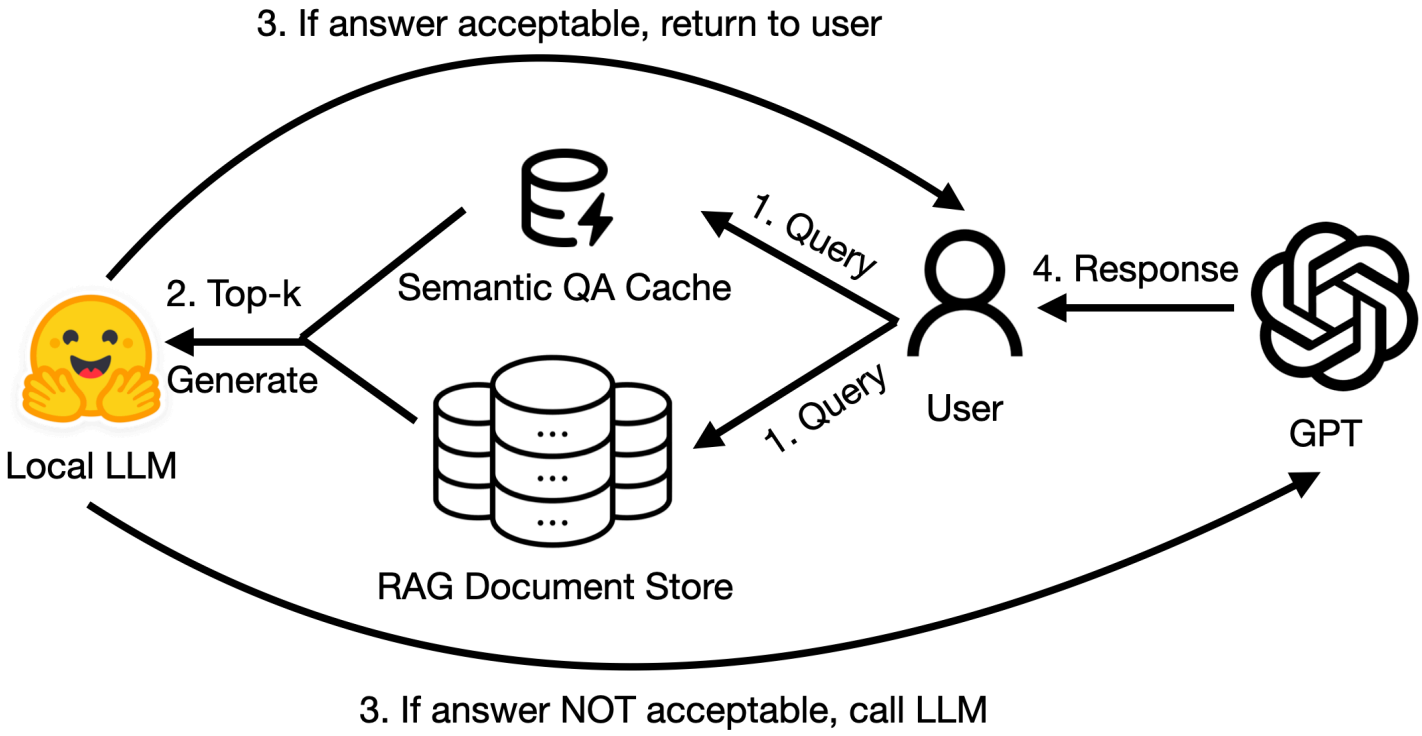

Figure 1. User Query Workflow of the Advanced RAG System

Challenges for implementing the above system might include the following:

- *Challenge 1*: How to determine whether a cache entry, consisting of a question-answer pair, is useful? Using the similarity score of the question, or the answer, or both?
- *Challenge 2*: In addition to semantic metrics such as the similarity score, what more could be done to determine whether a cache entry could be used to answer the question correctly?
- *Challenge 3*: How to determine whether the local LM's response successfully answers the prompt? FrugalGPT's LLM cascade approach might provide a good reference.
- *Challenge 4*: What is a good cache-eviction policy? Is LRU good enough?

Based on observation 2, prefetching is also worth considering if time allows. A preliminary idea of prefetching involves the following steps:

- A local LLM is used to construct a Prompt Prediction Tree (PPT), based on user-specified depth and branching factor. Shown below is an example PPT with depth 2 and branching factor 2.
    - The root of the PPT is the user's prompt.

    - The depth of the tree tree determines the number of subsequent prompts to predict.
    - The branching factor determines the number of possible next questions to predict based on the previous question.
- The system invokes the LLM to prefetch an answer that could answer all questions in the PPT, and stores it in the cache, which hopefully improves the cache hit rate for subsequent user prompts in this session.

Figure 2. Illustration of PPT with Depth 2 and Branching Factor 2

Even though prefetching might make an extra call to the LLM, or incur more input/output token costs, a rationale would be that within a single session, if the predicted queries are correct, it is possible to avoid multiple LLM calls in subsequent questions, thus achieving latency reduction and overall cost savings.

Resources

- LLM: GPT-3.5/GPT-4
- Vector store: Milvus
- Local pre-trained or fine-tuned LM
- Benchmark dataset/workload: QuAC (Question Answering in Context) [1]
- A server might also be needed for timing/benchmark

Deliverables

- A working RAG system with naïve caching (the default for GPTCache)
- An evaluation on the performance of naïve caching
- Research results on each of the challenges

Timeline

- Feb -> Mid March: deploy RAG system with naïve caching
- Mid March -> Spring Break: evaluate naïve caching with QuAC benchmark

- Spring Break -> Late April: research on the 4 challenges and refine the system
- If time allows: explore prefetching with PPT

References

[1] QuAC : Question Answering in Context
[2] GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings
[3] FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance
[4] Milvus: A Purpose-Built Vector Data Management System
[5] Query Expansion by Prompting Large Language Model