Assignment - 5

Name: Srilalitha Lakshmi Anusha Chebolu
Date: 02/26/25

CAG:

Retrieval-Augmented Generation (RAG) has been widely used to enhance large language models (LLMs) by integrating external knowledge. However, it comes with challenges like retrieval latency, errors in document selection, and system complexity. This paper introduces Cache-Augmented Generation (CAG) as an alternative approach that eliminates the need for real-time retrieval by preloading all relevant knowledge into the LLM's extended context window and caching its key-value (KV) state.

By leveraging long-context LLMs, CAG ensures fast, efficient, and error-free responses, making it a better alternative to RAG in cases where the knowledge base is limited and manageable.

The proposed CAG framework operates in three main phases:

1. External Knowledge Preloading:
    - All relevant documents are preloaded into the LLM.
    - A precomputed KV cache stores the inference state of the model.
    - This eliminates retrieval during inference.
2. Inference Process:
    - Instead of retrieving documents dynamically, the model uses the precomputed cache.
    - This speeds up response time and ensures contextual accuracy.
3. Cache Reset:
    - The KV cache can be efficiently reset without reloading everything from scratch.
    - This helps maintain system performance across multiple queries.

KV caching is a technique used to speed up LLM inference.

To understand KV caching, we must know how LLMs output tokens.

- Transformer produces hidden states for all tokens.
- Hidden states are projected to vocab space.
- Logits of the last token is used to generate the next token.
- Repeat for subsequent tokens.

Thus, to generate a new token, we only need the hidden state of the most recent token.

None of the other hidden states are required.

Next, let's see how the last hidden state is computed within the transformer layer from the attention mechanism.

During attention:

The last row of query-key-product involves:
- the last query vector.
- all key vectors.

Also, the last row of the final attention result involves:
- the last query vector.
- all key & value vectors.

The above insight suggests that to generate a new token, every attention operation in the network only needs:

- query vector of the last token.
- all key & value vectors.

But, there's one more key insight here.

As we generate new tokens:

- The KV vectors used for ALL previous tokens do not change.

Thus, we just need to generate a KV vector for the token generated one step before.

The rest of the KV vectors can be retrieved from a cache to save compute and time.

This is called KV caching!

To reiterate, instead of redundantly computing KV vectors of all context tokens, cache them.

To generate a token:
- Generate QKV vector for the token generated one step before.
- Get all other KV vectors from the cache.
- Compute attention.

KV caching saves time during inference. Find a video in the comments depicting this.

In fact, this is why ChatGPT takes some time to generate the first token than the subsequent tokens.

During that time, it is computing the KV cache of the prompt.

That said, KV cache also takes a lot of memory.

Consider Llama3-70B:
- total layers = 80
- hidden size = 8k
- max output size = 4k

Here:
- Every token takes up ~2.5 MB in KV cache.
- 4k tokens will take up 10.5 GB.

More users → more memory.

Relavant Code:

CAG_Optimizing_LLMs_with_cache_augmented_generation: Code

Convert_Document_to_Knowledge_Graph_Langchain_Openai: Code