Globally Shared KV Caching for Large Language Models

Jun Bang Chen, Tanishk Deo, Bereket Yoseph, Mose Kim, and Samuel Shao
College of Computing
Georgia Institute of Technology

February 21 2025

Motivation and Objectives

Large-Language Models have become ubiquitous and can incur high inference latency due to repeated computation of key-value (KV) pairs during attention operations.

We propose a Globally Shared KV Cache, which stores and retrieves KV pairs across multiple queries and users. A context similarity layer identifies reusable KV pairs, reducing redundant computations. Our solution proposes a shared KV caching mechanism that would minimize duplicate key-values for multiple users and optimize large-scale memory usage. The integration of context similarity ensures efficient retrieval, enhancing model performance.
Previous methods, such as BatchLM [1], "Knowledge-Delivery Networks" [2], and prefix KV caching [3]. These work on a single node basis for their contexts - and we would like to expand this by incorporating multiple LLM nodes.

This proposed solution would reduce latency and computational costs, making LLMs more efficient for repeated workloads. This approach is particularly beneficial for chatbots, search engines, and AI systems in which repeated responses are often generated for multiple users. This is a key internet system as "centralizing" the cache for multiple users could lead to lower latency for LLM inference.

Related Work

In looking for related works, we have come across other projects that take a similar approach to solving the problem of optimization via KV Caches, though none have fully realized this particular method. One similar research paper is "A Survey on Large Language Model Acceleration based on KV Cache Management" [4], in which the focus was optimization at the Token Level through methods such as KV Cache Selection, KV Cache Merging, and Low-Rank Decomposition for compact storing, the Model Level through attention algorithms, and System Level through scheduling and hardware design.

Other similar research includes a research paper called "Efficient Memory Management for Large Language Model Serving with Paged Attention" [5], where the approach to the problem is to use an attention algorithm inspired by how virtual memory is handled on a computer, called PagedAttention. This is utilized primarily to avoid large KV Caches, which, noticeably, defeat the purpose of the cache.

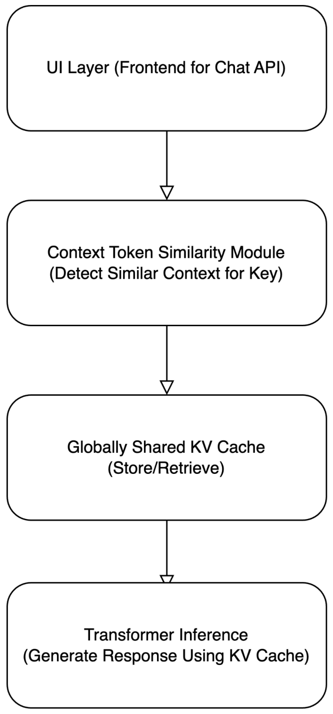
As previously mentioned, while there have been similar ideas carried out, the horizontal scaling approach of this project is a unique approach that has yet to be realized in terms of potential.

Proposed Work

This project aims to optimize KV Cache in transformer models used by modern day Large-Language Models by implementing a global shared KV cache for multiple user sessions. Currently, Large-Language Models have KV-Cache stored by sessions - however, we can improve the efficiency by "centralizing" this cache among multiple user agents. This project aims to improve memory load usage and inference timings for transformer models.

We aim to evaluate this project based on a key factor of inference speed for queries for multiple users to benchmark the improvements over per-user K-V cache. We will use our transformer model with a user-based cache and a multi-user based cache and see the difference while using an open source KV cache implementation.

We will use a global KV Cache, which is unlike traditional methods, in which the LLM will use multi-user caches to propagate a response to each user if the context has been previously seen. In this case, we can have a large cache that can be used for many more queries.


UI Layer (Frontend for Chat API)
Context Token Similarity Module (Detect Similar Context for Key)
Globally Shared KV Cache (Store/Retrieve)
Transformer Inference (Generate Response Using KV Cache)

The main components will be a UI Layer for interacting with the transformer model, a context similarity layer for finding the "key" for the specific context, and a globally shared KV cache for quick store and retrieval for K-V pairs - this cache could be a tree-based decision or just keys and values, and the transformer model (LLM).

The UI Layer will send user inputs to the backend system. The inputs are processed by the Token Similarity Module, which identifies potential keys based on token similarity. If a reusable state is found, the request is routed to the Global Shared KV Cache Buffer to retrieve the cached values. The retrieved or newly computed KV states are passed to the LLM, which generates the final response. The system updates the cache buffer with new states as needed, ensuring it remains relevant for future queries.

Plan of Action

The primary focus of this project is to improve upon current KV caching methodology with a global KV data store. This requires the development of supporting components and services which will be generic in nature and utilize open-source materials when possible. The plan of action can be divided into three primary stages.

Stage 1 - Observational Discovery. This first stage involves exploring our LLM of choice, gaining familiarity with its tools and repository contents, fine-tuning abilities, and token processing methodology. The LLM we use will support auto-regressive text generation therefore it should use transformer-decoder models. The current choice of LLM is Meta's LLaMA 2, though the selection of the model could change.

Stage 2 - Supporting Infrastructure. The second stage of this project involves setting up the key infrastructure and supporting components of this application. We will use a microservices deployment model to deploy the various services needed to support the primary caching mechanism/service. A general description of the primary services required are provided below.

- *API Gateway and Routing Layer (Frontend):* This service will provide the user an entry point to the inference model using a GUI as well as API endpoints. This service will also provide the routing backbone for other services to map requests to the corresponding service.
    - Frontend UI: React-Typescript
    - REST API & Request Routing: Golang
- *LLM Inference Service:* This service will host the large language model itself. Due to the high resource requirements of LLMs, it will require specialized cloud-hosting for LLMs. The inference service communicates with the *Caching Service* during token generation and attention calculation to skip pre-calculated attention scores.
    - Inference Framework: Hugging Faces Transformers & Meta LLaMA 2
    - Cloud Hosting: Amazon EC2
- *Authorization and Other Services:* A cloud hosted integrated platform will simplify deployment and communication and generic services such as user account authorization.
    - Cloud Services: Amazon Services

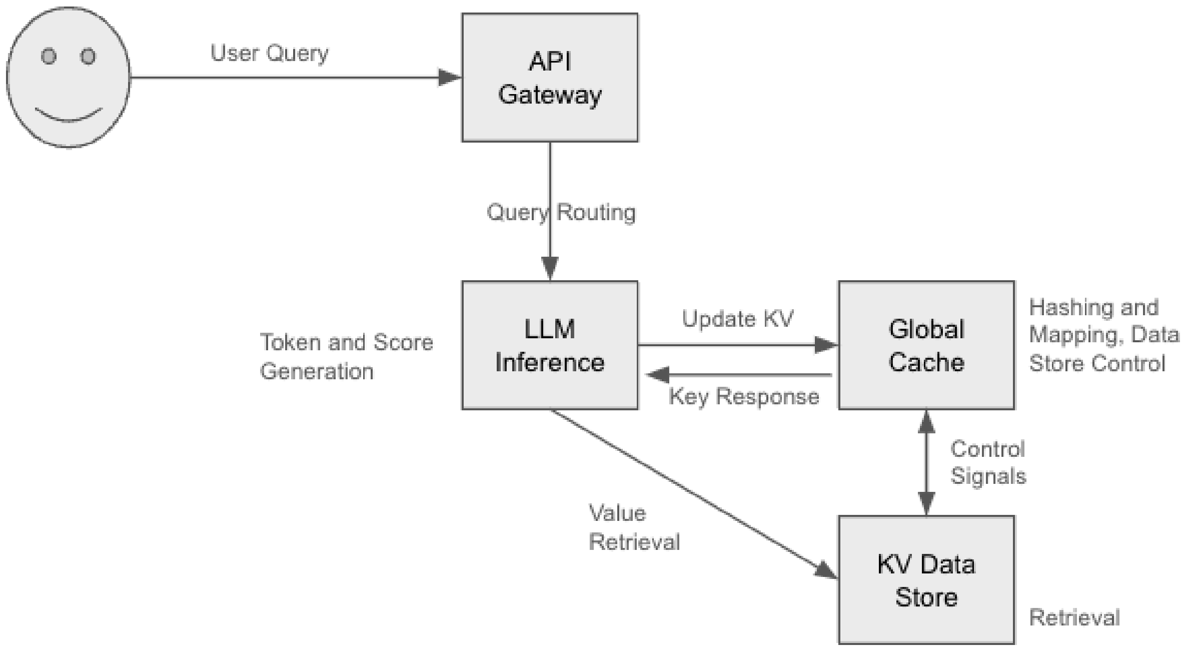
**During the development of this stage local options may be used instead of cloud-hosted options.

Stage 3 - Caching Service and Data Store. The bulk of work will occur in this stage, where we build our KV caching solution using a global data store. Global KV caching requires components such as key manager/token hashing, cache retrieval and storage, concurrency control, and eviction and replacement policies to prevent data overload. A general description of the two additional main services in this stage are provided below.

- *Global Cache Service:* The primary caching implementation and mechanism.
    - Primary Language: Python
    - Caching In-memory Datastore: Redis
    - Eviction Policy: LRU or default LLaMA KV cache eviction policy.
    - Concurrency Manager: Python

- *KV Data Store Service:* Stores the pre-calculated attention scores in a globally accessible data store.
    - Data Store: Redis

Resources will be obtained using free and open-source code. For certain cloud services, we plan on using the limited and free versions when possible or using alternative cloud options or even local options. Ideally, due to resource needs, the bulk of the application will use cloud-hosted ‘hardware’.
Component Design



Component Design. Note that additional services such as authorization services are not included as it is not essential to the primary purposes of the project.
Weekly Schedule

1. Proposal and Research: Feb. 14 - Feb. 21
2. Design Finalization and Scoping: Feb. 21 - Feb. 28 (1 week)
3. Stage 1: Feb. 28 - Mar. 7 (1 week)
4. Stage 2: Mar. 7 - Mar. 21 (2 weeks)
5. Stage 3: Mar. 21 - Apr. 11 (3 weeks)
6. Performance Benchmarking: Apr. 11 - Apr. 18 (1 week)
7. Report and Findings: Apr. 18 - Apr. 25 (1 week)

Evaluation and Testing Method

For the evaluation, we will focus on testing the effectiveness of the Globally Shared KV Cache in reducing inference latency, optimizing memory usage, and improving query performance for Large Language Models (LLMs).

Evaluation Criteria:
The system will be evaluated on the following criteria:

a. The system correctly identifies and retrieves reusable KV pairs based on token similarity.
b. The redundant computation across multiple user sessions is reduced in the shared KV cache
c. The caching mechanism efficiently handles high query volumes and concurrent users.
d. The eviction policy for maintaining an optimal cache state is effective.
e. The resilience of the system in handling failure and cache misses.

Validation Methods:

a. Utilize open-source LLM benchmarks to validate the effectiveness of cache retrieval.
b. Generate diverse queries with varying complexity to test the generalization ability of the cache.
c. Conduct quantitative performance benchmarking:
    i. Inference Latency - compare response times with and without the global KV cache.
    ii. Cache Hit rate - measure how often caches KV pairs are reused.
    iii. Memory Efficiency - track memory consumption
d. Conduct qualitative testing by asking users to compare responses generated with and without the global KV cache.

Bibliography

[1] Z. Zheng, X. Ji, T. Fang, F. Zhou, C. Liu, and G. Peng, “BatchLLM: Optimizing Large Batched LLM Inference with Global Prefix Sharing and Throughput-oriented Token Batching,” arXiv.org, 2024. https://arxiv.org/abs/2412.03594 (accessed Feb. 22, 2025).
[2] Y. Cheng, K. Du, J. Yao, and J. Jiang, “Do Large Language Models Need a Content Delivery Network?,” *arXiv.org*, 2024. https://arxiv.org/abs/2409.13761 (accessed Feb. 22, 2025).
[3] J. Yao *et al.*, “CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion,” *arXiv.org*, Jun. 03, 2024. https://arxiv.org/abs/2405.16444
[4] H. Li *et al.*, “A Survey on Large Language Model Acceleration based on KV Cache Management,” *arXiv.org*, 2024. https://arxiv.org/abs/2412.19442 (accessed Feb. 22, 2025).
[5] W. Kwon *et al.*, “Efficient Memory Management for Large Language Model Serving with PagedAttention,” *arXiv.org*, Sep. 12, 2023. https://arxiv.org/abs/2309.06180