Cache-Aware Routing for Multi-Instance LLM Serving

Course: Distributed Systems
Date: February 9, 2026

Objectives and Goals

When serving a Large Language Model (LLM) to many users at once, it is common to run multiple copies of the model behind a load balancer. Each copy keeps its own cache of previous computations (called a KV cache). If a new request is sent to a copy that already has the relevant data cached, the response is much faster. However, most load balancers today ignore this cache entirely and just spread requests evenly.

This project has five goals:

1. **Compare routing strategies side by side.** We will test six different ways to route requests to model instances. Three of these focus only on spreading load evenly, and three also try to reuse cached data. We will measure how fast each one responds using the model engine's own built-in timers, not estimates.
2. **Find the best balance between caching and load.** We propose a simple scoring formula that weighs cache reuse against how busy each instance is. We will test many combinations of these weights to find what works best across different types of workloads.
3. **Study how fresh the cache information needs to be.** Cache-aware routing needs to know what each instance has cached. We will test how often this information needs to be updated, ranging from real-time all the way down to every 30 seconds, to see how stale it can be before performance drops.
4. **Project how results scale to larger setups.** Our physical setup has two model instances on one GPU. To understand what happens with 4, 8, or 16 instances, we will build a simulator that is calibrated using real measurements from our two-instance setup.
5. **Release everything as open source.** All code for routing, workload generation, data collection, and analysis will be shared publicly so others can reproduce and build on our work.

Design Workflow

Our system has three layers, as shown in Figure 1 below.

At the top, a **Workload Generator** replays real multi-turn conversations (from the ShareGPT dataset) or sends synthetic retrieval-augmented generation (RAG) requests into the system.

In the middle, a **Router/Gateway** built with FastAPI receives each request and decides which model instance should handle it. The router contains three parts: a Policy Engine (which runs one of six routing strategies), a Cache Directory (which tracks what each instance has cached), and a Metrics Collector (which polls each instance for load and performance data).

At the bottom, two **vLLM instances** run the Llama-3.2-3B model on a shared RTX 5080 GPU, each using 45% of the GPU memory. Both instances have prefix caching turned on, so repeated prompts are served from cache instead of being recomputed.

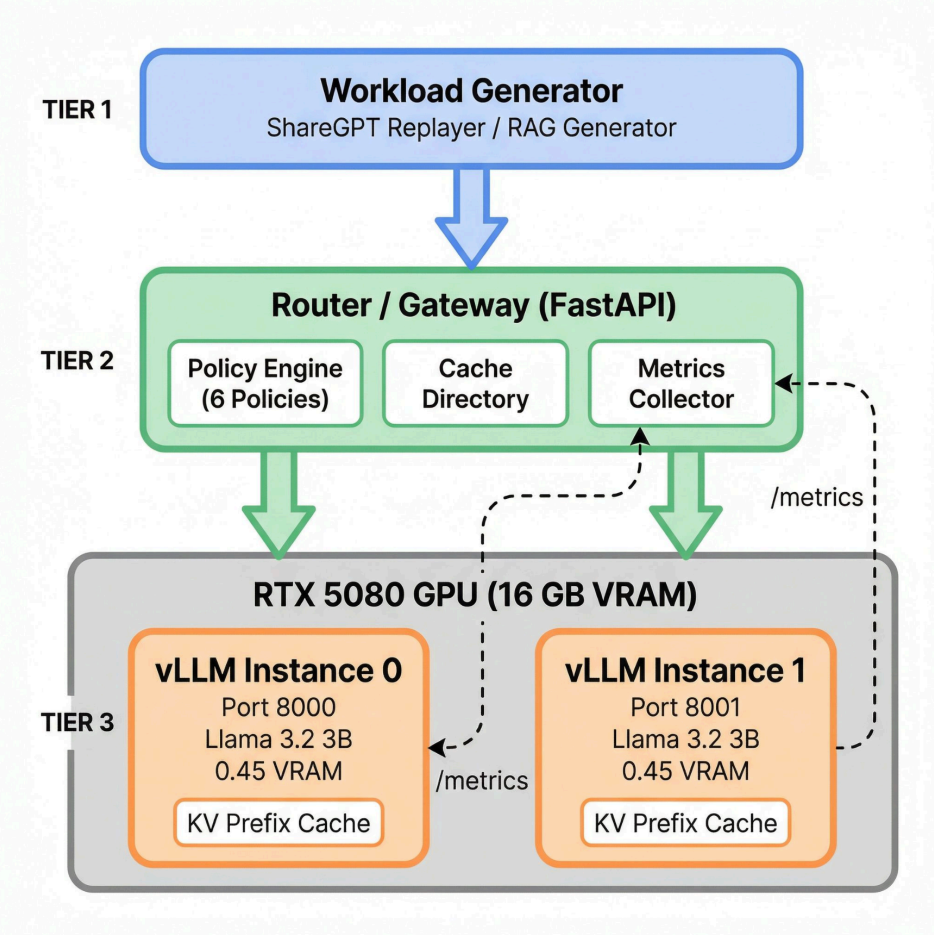

Figure 1: Figure 1: System architecture showing the three-tier design for workload generation, routing, and model serving with KV prefix caching.

Expected Outcomes

- Cache-aware routing will noticeably reduce response times compared to simple load balancing, especially for multi-turn conversations where consecutive messages share long cached prefixes.
- Our scoring-based policy will achieve the best overall results by avoiding the extremes. Pure cache routing causes some instances to get overloaded, while pure load balancing wastes cached data. Our approach balances both.
- Cache metadata does not need to be perfectly fresh. Updating it every few seconds will be good enough, making cache-aware routing practical without high overhead.
- As the number of instances grows, cache-aware routing will become more important, since there are more places for related requests to get scattered to.

References

[1] Y. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," *SOSP*, 2023.

[2] Q. Qin *et al.*, "Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving," *FAST*, 2025.

[3] H. Gao *et al.*, "Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention," *USENIX ATC*, 2024.

[4] T. Zhu *et al.*, "DualMap: Enabling Both Cache Affinity and Load Balancing for Distributed LLM Serving," *arXiv*, 2025.

[5] L. Zheng *et al.*, "SGLang: Efficient Execution of Structured Language Model Programs," *NeurIPS*, 2024.