

Deep Dive Report: Stateful Cache-Aware Routing for Distributed LLM Inference

Team: Aman Pandey Sanjeev Kumar, Aryan Khanolkar, Guna Avula
CS 59200: Machine Learning Systems, Purdue University

November 23, 2025

Abstract

The efficiency of distributed Large Language Model (LLM) serving is currently compromised by stateless load balancing, which destroys Key-Value (KV) cache locality across replicas. While frameworks like vLLM optimize intra-node memory, they lack mechanisms to coordinate cache states across a cluster. This report identifies the "Feedback Loop Gap" in existing distributed inference architectures. We propose a **Stateful Cache-Aware Router** that utilizes a novel **Batched Eviction Callback** mechanism injected into the vLLM kernel. This approach prioritizes "compute-to-data" routing, ensuring requests are directed to replicas holding the relevant prefix cache, thereby minimizing redundant prefill computation without the latency overhead of disaggregated storage.

1 1. The Core Research Problem

In high-throughput LLM serving, the **Prefill Phase** (processing input tokens) is compute-intensive and memory-bound, scaling quadratically $O(N^2)$ with sequence length. While **Automatic Prefix Caching (APC)** features in vLLM allow a single GPU to reuse previously computed KV tensors for shared prefixes, this optimization breaks down in distributed clusters.

Current industry deployments utilize stateless load balancers (e.g., Round Robin) that distribute requests arbitrarily. This leads to a "cache-blind" distribution where a user's second turn in a conversation is routed to a different replica than the first, forcing a redundant and expensive re-computation of the context history.

2 2. Gap Analysis: The Missing Puzzle Piece

To understand the necessity of this project, we analyze why current state-of-the-art frameworks fail to solve this specific locality problem efficiently.

2.1 vLLM: The "Island" Problem

vLLM implements PagedAttention and a Block Manager that efficiently handles KV memory *within a single node*. However, vLLM acts as an isolated "island." It possesses no mechanism to broadcast its internal cache state to a router or peer nodes. The internal BlockManager evicts blocks silently when memory pressure rises, leaving external schedulers blind to the actual state of the GPU memory.

2.2 LMCache: The "Data Movement" Latency

Frameworks like LMCache decouple the KV cache from the GPU, storing it on networked storage (Redis/Disk) to share between nodes. This prioritizes *availability* over *latency*. Fetching KV states

from remote storage—even over RDMA—incur network serialization overheads that often exceed the compute cost for short-to-medium prefixes.

2.3 The Identified Gap: Feedback-Loop Consistency

The missing piece in the ecosystem is **Application-Layer Consistency** without heavy orchestration. A router cannot simply "hash and hope" (Speculative Routing) because local LRU eviction policies will silently delete data, leading to high "False Hit" rates.

Our Contribution: We close this loop by implementing an active **Eviction Callback**. By modifying the inference engine to broadcast its memory state ("I just evicted Prefix X"), we enable a router that is *Eventually Consistent* with the physical GPU memory state.

3 3. System Architecture

The solution is a 3-tier architecture designed to decouple routing logic from infrastructure.

- **Tier 1: The Tokenizer-Aware Router (CPU).** A standalone Python service responsible for tokenizing prompts, hashing prefixes, and maintaining the GlobalCacheMap.
- **Tier 2: The Worker Pool (GPU).** Standard vLLM instances with surgical modifications to the BlockManager to detect and report cache evictions.
- **Tier 3: The Async Feedback Loop.** A non-blocking communication channel that updates the global map when workers free memory.

4 4. Detailed Implementation Plan

The implementation focuses on two critical technical fixes to standard routing approaches: Tokenizer Alignment and Batched Reporting.

4.1 Phase 1: The Router Logic (Tokenizer Alignment)

A critical flaw in naive routers is hashing the raw string prompt. LLMs cache based on **Token IDs**, not strings. Whitespace or encoding differences can result in different string hashes but identical token sequences. **The Fix:** The router must load the model's specific tokenizer.

```
from transformers import AutoTokenizer
import hashlib

# Initialize Tokenizer (CPU Side)
tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-Instruct-v0.2")

def get_routing_hash(prompt_text):
    # 1. Tokenize the incoming prompt
    token_ids = tokenizer.encode(prompt_text)

    # 2. Extract Stable Prefix (e.g., System Prompt)
    # Hashing the tuple ensures structure integrity
    stable_prefix = tuple(token_ids)

    # 3. Generate consistent hash for Map Lookup
    # Using hex digest prevents encoding issues
    return hashlib.sha256(str(stable_prefix).encode()).hexdigest()
```

Listing 1: Router Tokenization Logic

4.2 Phase 2: Surgical vLLM Modification (Batched Callbacks)

Sending an HTTP request for every single evicted block (as originally proposed) would cause a Distributed Denial of Service (DDOS) attack on the router during high-churn phases. **The Fix:** We implement a buffered, asynchronous background flusher inside the vLLM worker.

```
# File: vllm/core/block_manager_v1.py
import asyncio
from collections import deque

# Global Thread-Safe Queue
EVICTION_QUEUE = deque()

# 1. The Hook (Inside free_block method)
def free_block(self, block):
    # Instead of network call, just push to memory
    if block.is_cached_prefix:
        EVICTION_QUEUE.append(block.hash)
    # ... standard vLLM freeing logic ...

# 2. The Background Reporter (Async Task)
async def batch_reporter_loop():
    while True:
        await asyncio.sleep(0.1) # Flush every 100ms
        if not EVICTION_QUEUE:
            continue

        # Drain Queue
        batch = []
        while EVICTION_QUEUE:
            batch.append(EVICTION_QUEUE.popleft())

        # Single HTTP Call for 50+ blocks
        await send_batch_to_router(batch)
```

Listing 2: vLLM Worker Modification (Pseudocode)

5 5. Evaluation Strategy

We will validate the architecture using the **ShareGPT** and **MTRAG** datasets, which feature multi-turn conversations essential for testing prefix reuse.

Primary Metrics:

1. **Time-To-First-Token (TTFT):** We target a 30-50% reduction compared to the Round-Robin baseline.
2. **Stale Cache Rate:** Defined as the percentage of requests routed to a worker that results in a cache miss despite the router predicting a hit. This metric specifically evaluates the effectiveness of the *Eviction Callback* mechanism.

References

- [1] Kwon, W., et al. "Efficient Memory Management for Large Language Model Serving with Page-dAttention." *SOSP 2023*. <https://github.com/vllm-project/vllm>
- [2] Gim, I., et al. "LMCache: A High-Performance Distributed KV Cache for LLM Inference." *arXiv 2023*. <https://arxiv.org/abs/2310.13966>
- [3] Agrawal, A., et al. "Sarathi-Serve: Taming LLM Serving Latency with Chunked Prefills." *arXiv 2024*.

- [4] Patel, J., et al. "CacheBlend: Fast Large Language Model Serving for RAG with KV Cache Merging." *EuroSys 2024*.