LLM SERVING WITH MEMORY-EFFICIENT ROUTING AND MIGRATION

A Project

Presented to the

Faculty of the Department of Computer Science

California State University, Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Sai Kartik Ivaturi

Spring 2026

LLM SERVING WITH MEMORY-EFFICIENT ROUTING AND MIGRATION

SAI KARTIK IVATURI

Approved by:

Dr. Bin Tang, Ph.D.

ACKNOWLEDGMENTS

The author thanks Dr. Bin Tang for project guidance, technical feedback, and review throughout the development of this work. The author also thanks the Department of Computer Science at California State University, Dominguez Hills for supporting graduate study in applied computer science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

G	Set of serving instances or GPUs available to the router.
g	One instance or GPU in the serving pool.
i	One inference request.
p_i	Number of input prompt tokens in request i.
o_i	Maximum or observed output tokens for request i.
$M_g(t)$	Estimated key-value cache memory pressure on instance g at time t.
C_g	Key-value cache capacity or usable slot capacity of instance g.
u_g	Normalized key-value utilization, $M_g(t)/C_g$.
Q_g	Effective queue pressure on instance g, including queued and active requests.
$S(g, i)$	Routing score for request i on instance g; lower scores are preferred.
g^*	Instance selected by the routing algorithm.
d	Lower-pressure destination instance considered for migration.
$\rho_{g,i}$	Estimated prefix-cache hit fraction for request i on instance g.
$\hat{P}_g$	Exponentially weighted prefill latency for instance g.
$\hat{D}_g$	Exponentially weighted decode latency per output token for instance g.
$B_{s,d}$	Effective transfer bandwidth from source instance s to destination instance d.
$\alpha, \beta, \gamma, \delta, \eta, \theta, \lambda$	Tunable routing-score weights used to balance queue pressure, prefill work, decode work, cache pressure, observed latency, and prefix locality.

ABSTRACT

LLM serving performance is limited by the expansion of the key-value cache, especially when dealing with large context and duplicate prefixes (Kwon et al., 2023; Pope et al., 2023). The idea behind the queue routing method is to consider each active request similar. However, various elements such as the length of the prompt, the key-value cache pressure, and duplicates may result in two equally queued requests receiving significantly diverse costs in terms of service. This research analyzes memory-aware routing techniques for multiple instance LLM serving. As a baseline routing technique, least-queue routing was chosen because of its memory indifference and only consideration of backlog visibility. Controlled loads were set to test key-value pressure, prefixes duplication, and service-level agreement. With respect to the weighted key-value pressure, MELL's bin packing decreased end-to-end p95 latency by 20.4%, p95 time to produce a token by 23.7%, and time to first token by 9.1%. In addition, SLA success and cache hits increased from 98.3% to 100%.

CHAPTER 1

INTRODUCTION

Background

Large Language Models are transformer-based systems that generate text autoregressively, where inference takes place through processing an input prompt followed by token generation (Vaswani et al., 2017; Brown et al., 2020). Attention keys and values that get generated during this phase are cached into a key-value cache such that subsequent requests do not need to repeat the computation for previously generated tokens (Kwon et al., 2023; Pope et al., 2023). This technique allows more efficient computation; however, it adds complexity related to memory management, where the utilization of the key-value cache increases with respect to the size of input prompt, output length, and number of active requests (Kwon et al., 2023; Pope et al., 2023).

When operating in a multi-instance serving environment, requests need to get routed to an appropriate instance. Routing requests based on least-queue routing can serve as a simple baseline where requests get dispatched to the instance with the smallest visible queue. Queue-based approaches are well-studied since it provides a simple way to measure the load directly using the queue length (Mitzenmacher, 2001). However, in the context of LLM inference, the load metric cannot be fully captured using the queue length alone, since two instances having the same amount of active requests can differ vastly in terms of key-value cache consumption. One instance having to deal with requests that require long context would have higher memory pressure despite having smaller queues compared to an instance with larger queues, but having useful cache for reducing prefilling operations (Gao et al., 2024).

This work attempts to understand the mismatch in workload visibility between queue-visible loads and memory-visible loads. It formulates request routing as a memory-aware placement problem where queue pressure, key-value cache utilization, prefix locality, latency observations, and migration costs are all considered.

Figure 1 illustrates how prefill creates cached keys and values for the prompt tokens and how decode reuses the stored cache while computing only the new token state.

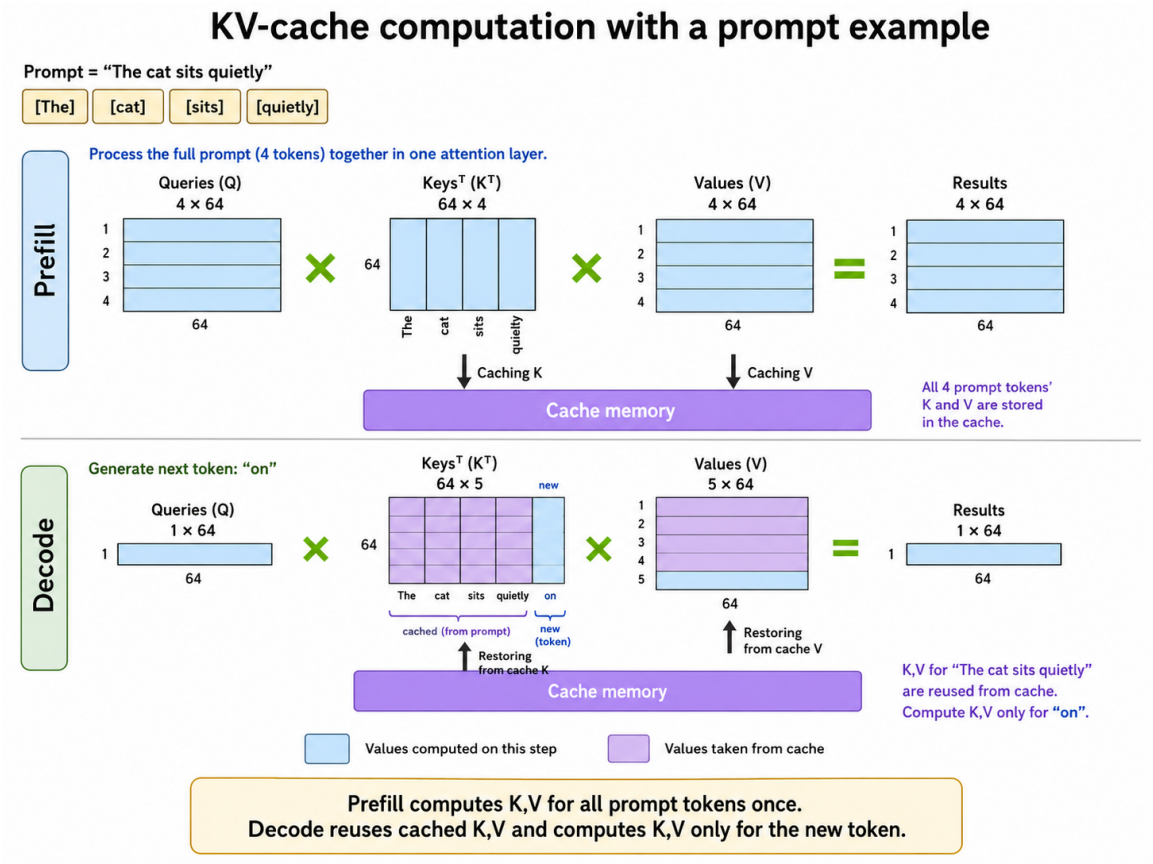

Figure 1. Key-value cache behavior during prefill and decode. Source: adapted conceptually from transformer inference and key-value cache management literature (Vaswani et al., 2017; Kwon et al., 2023).

Statement of the Problem

One of the main problems targeted by this project is that the common policies for routing in large language model (LLM) serving are oblivious to the memory state which constrains their inference capability. While least-queue routing is useful in allocating tasks based on their visibility in terms of backlog, this does not apply when transformer inference is concerned (Mitzenmacher, 2001). In LLM serving, the active request increases the key-value cache which depends on the sequence length as well as the extent of generation (Kwon et al., 2023; Pope et al., 2023). Therefore, two tasks with similar queue lengths could differ significantly in terms of the remaining space in the cache.

This makes both cache efficiency and latencies dependent on such memory considerations. If the request allocation was done based on queue lengths alone, then an instance might be selected for a job even though it is close to its memory limit. For example, in case of a repeated prefix workload, an efficient allocation would allow reusing the cached state while an inefficient allocation will cause prefilling operations in another instance (Gao et al., 2024). Such errors are especially important in the tail cases since expensive requests may make up for a large proportion of latencies seen by the users (Dean and Barroso, 2013).

This means that the current problem is not only concerned with balancing between the various instances but ensuring optimal task allocation across instances considering memory pressure, prefix locality, and latency expectations. It should be decided based on the experiment whether the inclusion of the information related to cache and latencies would improve the optimization of service-level objectives using different types of controlled workloads. To do so, the experiment should measure latencies, time to first token, latency for output token, cache hits, key value, and balance between the instances.

Purpose of the Project

The project focuses on developing and validating a memory-aware routing algorithm for multi-instance LLM serving. Past literature on inference serving shows that key-value cache management, batching, and scheduling have significant impacts on performance (Kwon et al., 2023; Yu et al., 2022; Pope et al., 2023). Recent literature reveals that prefix re-use and workload-aware placement may affect serving performance when requests have shared context or different size distributions (Gao et al., 2024; Griggs et al., 2024). Based on these observations, the current project primarily examines the routing layer, which determines the assignment of each request to an available instance.

The first objective is to compare the effectiveness of queue-only routing against memory-aware routing using the same set of workloads. The second objective is to determine whether the use of cache and latency information leads to improved tail performance, such as p95 end-to-end latency, p95 time to first token, and p95 time per token. Finally, the project tries to examine whether memory-aware placement leads to improved service-level objective satisfaction and cache hit behavior beyond queue-based routing only. Overall, these objectives bridge the gap between system implementation and the main research question. That is, does the state of the key-value cache provide useful routing information?

Significance of the Project

Importance of this work can be seen from analyzing the effect of routing on the resource that limits the performance of LLM serving: key-value cache memory. Serving systems need to manage requests of different length, with various lengths of generated outputs and various chances for reusing the prefixes. As shown in previous systems research, memory management, scheduling, and cache reuse become crucial for efficient inference in language models (Kwon et al., 2023; Yu et al., 2022; Gao et al., 2024). That means that a routing algorithm that ignores

the status of cache can produce seemingly balanced decisions based on queue length but actually being inefficient with regard to memory resources.

This project is important for systems design because it introduces the concept of separating visible queue load and invisible cache load. The latter notion starts mattering in case of workloads containing short prompts, long prompts, repeated system prompts, and repeated prefixes in conversations. Under these circumstances, choosing the instance based on queue length will no longer be the optimal strategy; instead, routing can try to target the instance with useful cache contents or avoid targeting instances with high cache pressure. This can lead to reducing tail latency, reducing prefill repetition, and improved achievement of SLA objectives.

Another way in which this work contributes to the field is serving as an evaluation platform. Comparing the strategies with respect to a set of workloads and giving the percentage change relative to the baseline makes the results easy to reproduce and analyze. Instead of considering routing a black box and looking only at the outcomes, this work focuses on connecting observed outcomes to the underlying signals.

Research Questions

This project follows the direction set by research questions that relate the placement problem to the performance of the system. In past studies, it was shown that inference using a transformer model heavily depends on memory management, scheduling, and cache reuse (Kwon et al., 2023; Yu et al., 2022; Gao et al., 2024). Therefore, the research questions test the hypothesis that the state of the key-value cache plays an additional role in routing compared with the least-queue approach.

1. Can memory-aware routing provide lower p95 end-to-end latency for workloads that put pressure on key-value caches compared with the least-queue routing algorithm?

2. Can memory-aware routing provide better p95 time to first token and p95 time per output token compared with least-queue routing?

3. Does the use of key-value cache and prefix locality result in improved SLOs and cache hit probability over least-queue placement?

4. Does the effect of memory-aware routing become more prominent at the tail rather than the median?

Limitations of the Project

The following limitations apply to the presented project and should be taken into account when considering the findings of the paper. Firstly, the analysis is done using controlled loads rather than production traffic. Indeed, the workloads used are intended to demonstrate the repeat prefix patterns, cache pressure at key-value level, and sensitivity to service-level objectives. At the same time, there exist other possible distributions of actual LLM requests that should be considered in real deployments.

Secondly, the results presented in this paper must be considered comparative rather than absolute in nature. Specifically, this research focuses on comparing the effect of memory-aware placement against a least-queue baseline. The results obtained do not mean that this method is guaranteed to yield lower latencies regardless of workload characteristics, the utilized hardware, larger models, inference engines, or batching strategy. As was discussed in previous researches, cache management and scheduler design still remain vital for transformers (Kwon et al., 2023; Yu et al., 2022; Pope et al., 2023).

Thirdly, the project focuses on the routing layer and does not attempt to compete with full-scale inference systems currently in use in productions. The proposed routing algorithm is informed by various factors, including but not limited to queue pressure, cache usage, prefix locality, observed latencies, and overhead associated with migrations. In contrast, some issues are intentionally left out from the scope of this research and will be addressed in future papers.

Definition of Terms

Table 1 defines the terms used throughout the report.

Table 1. Definition of Terms

Term	Definition
Large Language Model	A transformer-based model that generates text by processing a prompt and producing tokens autoregressively (Vaswani et al., 2017; Brown et al., 2020).
Inference	The process of running a trained model to produce an output for a user request. In LLM serving, inference usually includes a prompt processing phase and a token generation phase.
Key-Value Cache	The stored attention keys and values used during autoregressive generation. This cache avoids recomputing previous token state but increases memory usage as sequence length and concurrency increase (Kwon et al., 2023; Pope et al., 2023).
Prefix Reuse	The reuse of cached prompt state when multiple requests share the same beginning tokens. Prefix reuse can reduce repeated prefill work when requests are placed on an instance that already contains useful cached state (Gao et al., 2024).

Term	Definition
Least-Queue Routing	A queue-based routing policy that assigns new requests to the instance with the smallest visible queue. In this project, it is used as the baseline queue-only strategy.
Memory-Aware Routing	A routing approach that considers cache utilization, prefix locality, observed latency, and migration overhead in addition to queue pressure.
Time to First Token	The time between submitting a request and receiving the first generated token. It is used to measure interactive response delay.
Time per Output Token	The average time required to generate each output token after generation begins. It is used to measure decode performance.
Service-Level Objective	A target performance threshold used to determine whether a request satisfies the expected service quality.
Tail Latency	High-percentile latency, such as p95 latency, that captures the experience of slower requests. Tail latency is important because a small number of slow requests can dominate user-visible performance (Dean and Barroso, 2013).

CHAPTER 2

REVIEW OF RELATED LITERATURE

Transformer Inference

Based on the literature related to the topic, the effectiveness of LLM serving is strongly connected to the attention-based transformer architecture and its memory requirements. Attention in transformer models generates tokens autoregressively, meaning that new tokens depend on context from previous tokens (Vaswani et al., 2017; Brown et al., 2020). To save calculations and recompute attention for previously computed sequences, inference solutions use a key-value cache containing precomputed results from previously executed steps (Kwon et al., 2023; Pope et al., 2023). While cache optimization makes the computations faster, it also increases memory consumption, which rises along with concurrency and sequence length.

One of the most significant applications of key-value caching for inference is PagedAttention introduced in vLLM (Kwon et al., 2023). Its innovation involves using a virtually memory-like layout for key-value blocks to make their handling and sharing more efficient. This project shows that the memory layout of the key-value cache is an important element that influences performance and concurrent request handling. Such implications can also be found in other projects, like Flexible Memory Access, which focuses on serving generation under memory-constrained conditions. Efficiently Scaling Transformer Inference analyzes the connection between model execution and serving performance (Sheng et al., 2023; Pope et al., 2023).

These projects serve as a basis for the present research as they demonstrate that LLM serving should not be considered merely a queueing process since memory states directly influence performance. Therefore, the current project continues in a similar vein by investigating whether the utilization of cache and prefix locality should affect the choice of the inference instance.

Serving Schedulers

Serving scheduling becomes a critical issue since inference queries have varying resource requirements. Orca shows that serving systems can improve resource utilization with more detailed scheduling than per-request through scheduling at the level of iterations for transformer-based generation (Yu et al., 2022). This work highlights the uniqueness of LLM serving compared to traditional web-service routing where the notion of a request is well-defined and does not change over time. On the contrary, a request becomes gradually completed during prompt processing and then continues with decoding.

Despite its drawbacks, queue-based routing makes a useful baseline due to its simplicity and intuitive use of available system data such as backlog. Least-queue routing always allocates incoming work to the server with the minimum queue length. As stated in literature, the knowledge about queues provides effective ways of reducing waiting in distributed computing systems (Mitzenmacher, 2001). The only problem of queue-based routing is that it lacks information about the used amount of key-value cache from the current requests.

In the course of this project, the least-queue routing algorithm will be the main baseline as it represents the view on load based only on queue information. It might be considered effective as the relevant literature proves its efficiency. Nevertheless, queue length cannot tell us how much of the key-value cache was used by the currently active tasks. Therefore, memory-aware routing algorithms can potentially provide better results.

Prefix Reuse

A further research direction regarding the literature on LLM serving is the reuse of repeated context. Many LLM applications include repeatedly appearing system prompts, conversations, and document prefixes. In such cases, a serving system can save unnecessary prefill computation if the stateful cache containing relevant information is available. AttentionStore shows that repeating the attention between several turns of a multi-turn conversation can be used as a way of reducing the serving cost (Gao et al., 2024). SGLang demonstrates benefits of using structured programs for language models, with prefix reuse being one such benefit (Zheng et al., 2023).

As shown in Figure 2, prefix locality affects routing decisions as well. The request routed to an instance that has the relevant prefix cached does not need to repeat the same computations, while the same request routed to an instance without relevant information will result in additional workload and extra cache pressure.

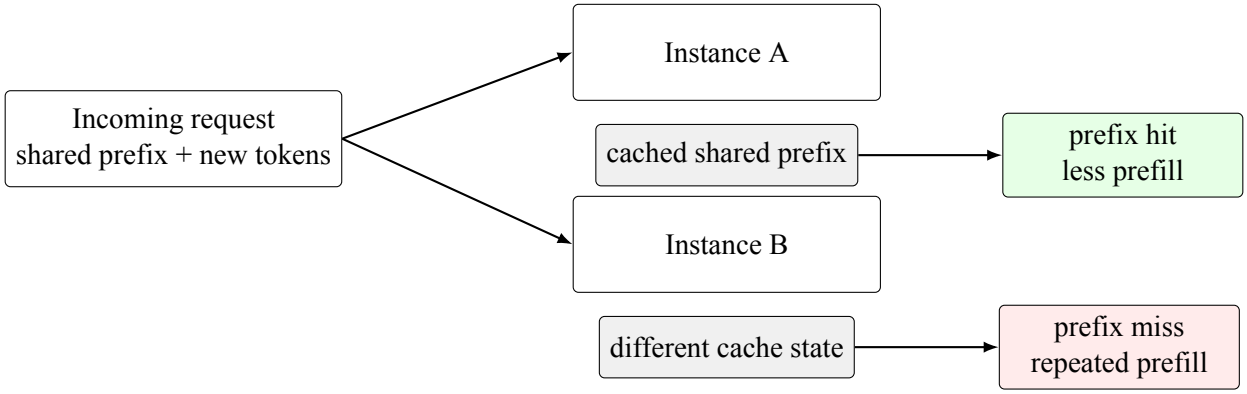

Figure 2. Prefix locality in LLM request routing. Source: adapted conceptually from cache reuse concepts discussed in AttentionStore and SGLang (Gao et al., 2024; Zheng et al., 2023).

This line of study is directly related to the current project because prefix reuse changes the understanding of load distribution. While an instance with an apparent queue of requests may seem a bad option at first, the request with reused prefix may require less computational resources to serve than another one directed to an instance with a short queue due to the absence

of useful state. Hence, routing that ignores cache locality may lead to unnecessary memory load and additional latency even if it looks reasonable in terms of queue.

The findings of prefix reuse show that the key-value cache should not only be considered as a resource burden but also as a chance for optimization. Although the cache entry uses the memory, it helps avoid recomputing of the requests sharing the same prefix. Hence, a routing mechanism must take into account whether sending the request to a certain instance will benefit or penalize caching.

Resource Allocation

The latest literature suggests that the choice of LLM serving involves taking into account the available resources in the deployment environment. Specifically, Mélange considers cost-efficient LLM serving by means of exploiting GPU diversity (Griggs et al., 2024). According to the study, the most effective choice of hardware resources will depend on service characteristics such as request size, request rate, and latency SLO. The researchers frame the GPU allocation problem as a cost-aware bin packing where GPUs represent bins and fragments of the service represent items (Griggs et al., 2024). The importance of this framing lies in that it illustrates how placing can be viewed as an optimization problem in respect to resource awareness, rather than as a queuing assignment problem.

Figure 3 illustrates the notion of placing used in Mélange.

The findings presented by the paper are also relevant to the project. Specifically, the researchers illustrate the inefficiency of relying on a single GPU type by highlighting that none of the existing GPUs will be the cheapest for all possible requests of any size and SLOs. The study demonstrates savings of 9–77% for short-context tasks, 2–33% for document-oriented tasks, and 4–51% for mixed context in terms of deployment costs compared to approaches based on a sin-

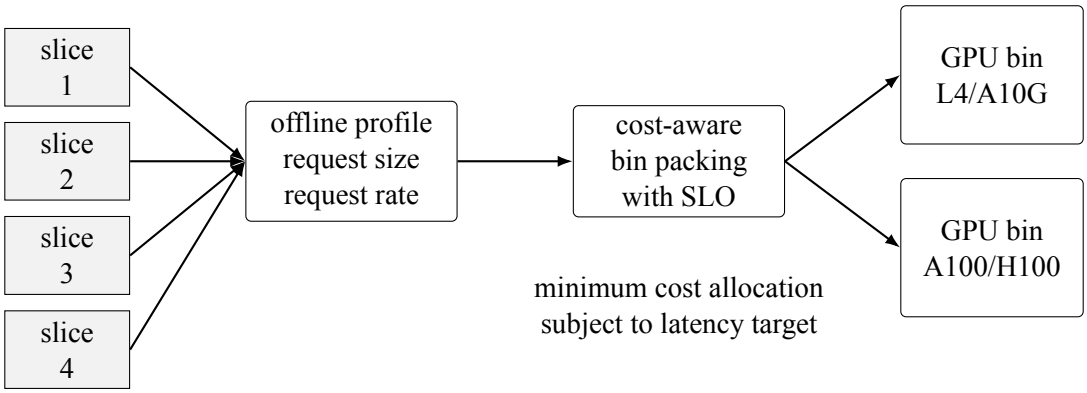

Figure 3. Cost-aware GPU allocation as bin packing. Source: adapted conceptually from the Mélange allocation formulation (Griggs et al., 2024).

gle GPU type (Griggs et al., 2024). As a result, it supports the notion of inability of LLM serving efficiency explanation by relying on a single load signal.

While Mélange is primarily about choosing the right combination of GPUs for serving a specific task, the current project focuses on dynamic request routing between different instances of serving destinations. Nonetheless, the two works share the approach of considering resource limitations. The former discusses how workload characteristics affect the choice of hardware components while the latter is concerned with memory pressure and cache locality effects.

Tail Latency

Tail latency represents an important metric for consideration in distributed serving systems because users encounter only the slowest aspects of the request path and not just its average characteristics. Tail at Scale research has shown that high percentile latencies represent the largest contributor to user experience in interactive services at scale (Dean and Barroso, 2013). LLM Serving also faces these concerns while introducing new sources of delays such as prefill time, decoding time, batching interference, and cache pressure on key value caches (Kwon et al., 2023; Yu et al., 2022). Figure 4 illustrates the importance of p95 latency and SLO achieve-

ment because two approaches may look almost identical on the median but significantly different in the high percentiles, precisely the area where service performance is determined by slow requests.

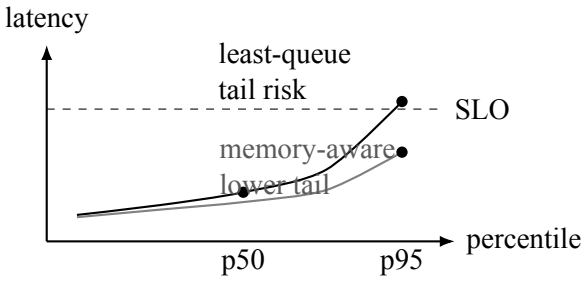

Figure 4. Median and tail latency under an SLO. Source: adapted conceptually from tail-latency and serving-SLO concepts (Dean and Barroso, 2013; Griggs et al., 2024).

For this reason, the evaluation uses end-to-end p95 latency, p95 time to first token, and p95 time per output token rather than depending only on median latency. Median values provide important information about average request behavior but can overlook issues like long prompt latency, cache miss, and inefficient instance placement. Service level objectives are also relevant as they represent latency behavior in the context of quality delivery of services. Previous works related to LLM serving use latency SLOs for evaluating acceptable deployment performance under certain workloads (Patel et al., 2024; Griggs et al., 2024).

Median improvement alone does not show that memory awareness has improved the main bottleneck of serving. In contrast, tail improvement shows that cache pressure and inefficient routing have been alleviated. That said, p95 latency and SLO satisfaction become essential for evaluating least queue routing compared to memory-aware routing.

Literature Gap

The literature examined here reveals four major insights. First, inference for transformers is highly sensitive to the management of key-value cache memory because of its impact on mem-

ory footprint and inference time (Kwon et al., 2023; Pope et al., 2023). Second, scheduling and batching become crucial considerations in LLM-serving due to the iterative nature of generation requests compared to one-shot inference (Yu et al., 2022). Third, prefix reuse becomes another consideration in terms of request costs when cached states can be reused from some previous instance (Gao et al., 2024; Zheng et al., 2023). Fourth, modern allocation schemes such as Mélange have shown that LLM-serving policies need to consider load characteristics and SLOs (Griggs et al., 2024).

The gap filled by the current project lies within the routing layer. While existing solutions account for the factors of memory, scheduling, reuse, and hardware allocation, a queue-aware router does not take into account the internal details of those processes beyond the queue visibility itself. The situation may lead to misleading results when, for example, the two instances have equal queues but different memory pressures due to the key-value cache or the possibility of reuse in prefix locality.

As a result, the project proposes to explore empirically the idea of memory-aware routing in multi-instance LLM-serving. There is no interest in creating a new inference engine or allocation strategy; the objective will be to examine the impact of key-value cache utilization, prefix locality, latency metrics, and migration cost on placement beyond a simple least-queue scheme.

CHAPTER 3

METHODOLOGY

Evaluation Method

An experimental systems method will be used to evaluate the project. The goal is to compare routing with no memory awareness against queue-based routing in a controllable workload setting. Since this approach does not account for anything other than the visible backlog, it will be referred to as least-queue routing. The proposed alternative will incorporate factors such as key-value cache memory pressure, prefix locality, observed latency, and the cost of offloading tasks from one server to another. This choice is inspired by existing literature which argues that key-value cache memory, scheduling, and latency tail are critical issues for LLM-serving infrastructure (Kwon et al., 2023; Yu et al., 2022; Dean and Barroso, 2013).

In the scope of this experiment, the artifacts will consist of a router, several server instances, and an experimental framework. In particular, for each request, the router identifies instances currently available in the system, chooses where to route the request based on the selected algorithm, and collects performance metrics. Afterward, the workloads are replicated using different routing strategies to establish causality. Therefore, least-queue routing will be considered the baseline case, and memory-based routing will be the experimental case.

The procedure described above will be carried out according to the methodology illustrated in Figure 5 below. Requests are sent to the experimental framework, an instance is selected based on the configuration, the instance executes the request, prompt plus generation, and finally, results are measured.

Metrics used as the outcome include p95 latency, p95 time to first token, p95 time per token, cache hit frequency, cached tokens reuse, and SLO fulfillment.

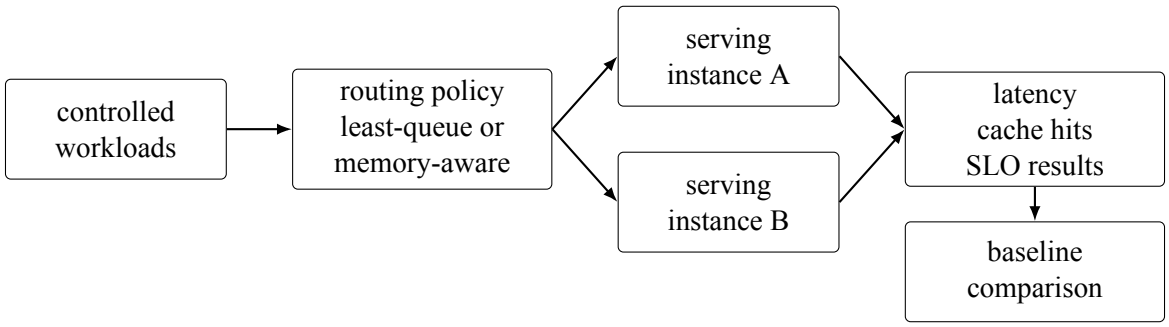

Figure 5. Methodology workflow for memory-aware routing evaluation. Source: adapted conceptually from LLM serving evaluation concerns in prior work (Kwon et al., 2023; Yu et al., 2022; Griggs et al., 2024).

System Architecture

For this evaluation, the project artifact used is a routing layer along with several serving instances. The routing layer will receive the inference request, maintain state regarding the availability of serving instances, implement the specified routing strategy, and collect measurements at the level of individual requests. Serving instances will reveal their state to the routing layer in the form of visible queue depth, active requests, state regarding key-value cache, prefill latency, decode latency, prefix cache operation, and ability to transfer tokens. The above-listed variables are necessary to determine whether memory-aware routing provides additional information beyond queuing.

Figure 6 represents the architecture of the system used in this evaluation. Solid lines represent requests sent and responses received by a routing layer and an asynchronous Kafka path used for transferring messages. Dashed lines stand for gathering state data and measurements for the experiment. This choice was intentional since routing policy relies on real-time state while the performance analysis is calculated post-factum.

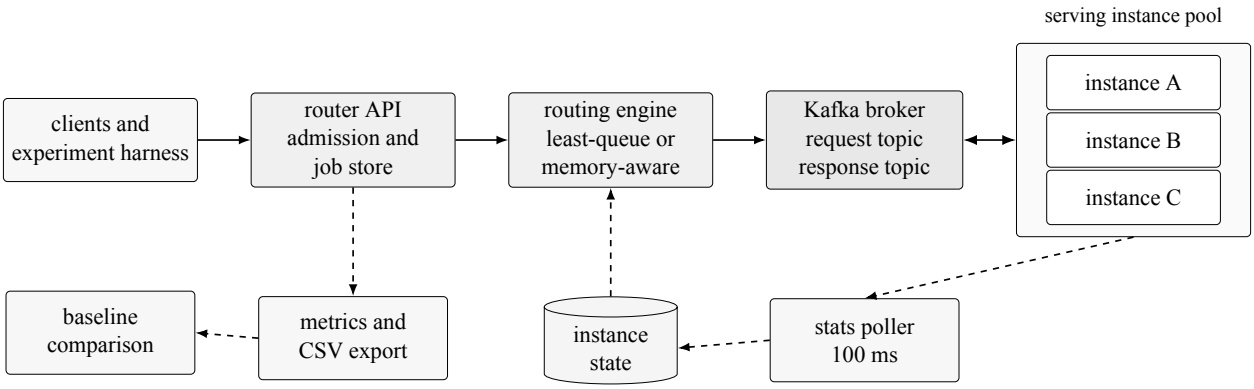

Figure 6. System architecture for memory-aware LLM routing evaluation. Source: adapted conceptually from the project implementation and LLM serving design concerns in prior work (Kwon et al., 2023; Yu et al., 2022; Griggs et al., 2024).

Prior to running each experiment, router asks serving instances to provide their state, which involves both visible load and memory-related load. Visible load corresponds to the number of requests in queue and active requests. Memory-related load will be defined by key-value cache usage, slot usage, number of cache hits, token usage, and key-value evictions. It is consistent with the findings discussed in Chapter 2, where key-value cache and scheduling were mentioned among the top factors affecting LLM serving performance (Kwon et al., 2023; Yu et al., 2022; Pope et al., 2023).

It was chosen because it enables repeating the same experiment with the same workload but using different routing policies, keeping the serving environment identical. Therefore, any difference in results may be safely attributed to the routing policy in question. This also makes comparison of the tail latency, cache hit/miss rates, and achieved service level objectives possible between various routing strategies.

Routing Policies Compared

Least-Queue Routing. The main baseline used in this experiment is the least-queue routing policy. Such policy chooses an instance based on observable queue pressure, as represented

by queued work and active work. Least-queue routing is adopted as the baseline policy because it uses simple queue information that is available at the moment of routing and is also well-justified by classical load-balancing literature (Mitzenmacher, 2001). Nonetheless, least-queue routing does not consider key-value cache pressure or prefix locality, the factors studied in this paper.

Memory-Aware Policies. Memory-aware policies include additional state from each instance that processes the requests. Intelligent routing considers queue pressure, prompt size, expected output size, prefill latency, memory pressure, prefix locality, and cached-token history. MELL bin-packing takes into consideration the need to re-route an incoming request away from a high-pressure instance when there is another instance with low key-value cache pressure. Bin-packing considers the capacity of an instance as the soft memory and compute budget and tries to place the request to the instance where it makes the best use of the remaining capacity. These policies are inspired by previous findings showing that memory management, request size, and SLO affect LLM serving (Kwon et al., 2023; Griggs et al., 2024).

The memory-aware placement score used in this paper is formulated with respect to the notation in the List of Symbols. For an incoming request i and a candidate instance g, the router calculates $S(g, i)$, where the smaller value indicates more preferred placement:

$$S(g,i) = \alpha Q_g + \beta p_i(1 - \rho_{g,i}) + \gamma o_i + \delta u_g + \eta \hat{P}_g + \theta \hat{D}_g o_i - \lambda \rho_{g,i}.$$

In this equation, Q_g represents queue pressure, $p_i(1 - \rho_{g,i})$ is estimated uncached prefill work, o_i is expected decode work, $u_g = M_g(t)/C_g$ is normalized key-value cache pressure, and $\hat{P}_g$ and $\hat{D}_g$ are estimated prefill and decode latencies, respectively. Finally, the last term rewards prefix locality by encouraging higher $\rho_{g,i}$, that is, the higher the portion of prompt is expected to be reusable on instance g.

MELL Bin-Packing Variant. The MELL bin-packing policy is formulated as a separate memory-aware routing policy because it considers placement as a resource packing problem, instead of a score minimization problem only. Bin-packing falls under the Mélange allocation

policy framework where each serving decision is considered a resource-aware placement with SLO constraints (Griggs et al., 2024). In this paper, MELL bin-packing tries to estimate whether the request fits in the remaining memory and compute budget of each instance. Request memory requirement is estimated as prompt size plus a portion of the output size, and request compute requirement is estimated as the sum of the prompt size and the output size. The policy prefers an instance that fits the request with the minimum wasted capacity, low queue pressure, and positive cached-token history.

This distinction is important because while least-queue routing equally distributes requests to all visible queues, MELL bin-packing purposefully maintains instances at productive memory utilization levels and refuses to route a request to an instance that does not have enough key-value cache budget. Put differently, the approach tries to prevent two possible failures: request distribution to useless caches and overload of a poorly-equipped instance. This choice is inspired by previous findings where it was concluded that key-value cache memory and serving capacity were the decisive factors for efficient processing of LLM requests (Kwon et al., 2023; Pope et al., 2023).

Algorithm 1 Memory-Aware Routing for Request i

Require: Candidate instance set G, request i, prompt tokens p_i, output tokens o_i
Ensure: Selected serving instance g^*

for all $g \in G$ **do**
    Read Q_g, $M_g(t)$, C_g, $\hat{P}_g$, and $\hat{D}_g$
    Compute $u_g \leftarrow M_g(t)/C_g$
    Estimate prefix reuse fraction $\rho_{g,i}$
    Compute $S(g, i)$ using the memory-aware score
end for
Select $g^* \leftarrow \arg\min_{g \in G} S(g, i)$
if u_{g^*} is high and a lower-pressure destination d exists **then**
    Estimate migration cost using source g^*, destination d, and transfer path $B_{g^*,d}$
    if estimated migration cost is lower than estimated queue wait **then**
        $g^* \leftarrow d$
    end if
end if
return g^*

Algorithm 2 MELL Bin-Packing Routing for Request i

Require: Candidate instance set G, request i, prompt tokens p_i, output tokens o_i
Ensure: Selected serving instance g^*

```
Estimate request memory demand from p_i and o_i
Estimate request compute demand from p_i and o_i
for all g ∈ G do
    Read Q_g, u_g, active work, and cached-token history
    Estimate remaining memory and compute capacity on g
    if request demand exceeds remaining capacity then
        Add overload penalty to the bin-packing score
    end if
    Estimate memory waste and compute waste after placement
    Add queue penalty from Q_g
    Subtract cache bonus when cached-token history indicates useful reuse
end for
Select g* from the lowest-score candidates
return g*
```

Algorithm 1 shows how the score is applied at routing time. The algorithm first evaluates all available instances in G, computes a score for each candidate, and chooses the instance with the lowest score. If the chosen instance is under high key-value pressure, routing of the request to a different location is determined by the cost-benefit analysis of whether it would make sense to wait. Only when the memory pressure of the chosen instance is high and there is an option to move to a less pressured instance, the routing operation is performed. This explains how the formulation maps to practical placement problem.

Algorithm 2 outlines MELL bin-packing variant in which a request fitting in the remaining memory and compute budget of the instance is preferred. If the request does not fit in the current instance, then it is penalized, and in case of inefficient fitting, there are additional costs. On the other hand, the policy gives a score reduction bonus to the instance if it has good cached-token history that implies the benefits from reusing cached tokens. Accordingly, the chosen instance is not necessarily the instance with the smallest capacity but one with the best fitting while taking into account queue pressure and prefix locality.

This comparison separates queue-visible routing from memory-aware routing. Least-queue routing policy relies on a queue-visible view of the system while the memory-aware poli-

cies take into account queue pressure, cache pressure, prefix locality, and prefill/decode latencies. The evaluation measures whether memory-aware policies improve p95 latency, time to first token, time per output token, cache hit rate, and SLO compliance.

Workload Construction

The experiments use controlled loads instead of uncontrollable production loads. It helps to make the comparison more reproducible and isolates the exact conditions that have been investigated. Three main conditions are tested with the help of specially created workloads. These conditions are: steady SLO-sensitive serving, key-value cache pressure, and migration benefits for repeated prefixes. They are in accordance with literature, which states that request size, cache reuse, and latency targets affect LLM serving performance (Gao et al., 2024; Griggs et al., 2024).

Steady load is used to generate both repeated and unique prompts for testing both cacheable and non-cacheable requests. Key-value pressure test load uses short, medium, and long repeated prefixes along with some unique prompts to create variation in prompt size and cache pressure. The migration benefit load uses mostly long repeated prefixes so as to provide the routing policy with the chance to take advantage of prefix information present in certain instances.

Each condition is run with the same sets of strategy, request number, number of concurrent requests, SLO thresholds, and eviction policy unless specified otherwise by the experiment. The default experimental settings are strategies round-robin, least-queue, intelligent, cache-affinity, MELL, and MELL bin-packing. All obtained data are then sorted according to strategy and eviction policy before computing statistical results.

Metrics Analysis

The evaluation considers metrics taken at both the request level and the strategy level. Metrics for requests include the selected instance, prompt type, number of prompt tokens, number of output tokens, latency, time-to-first-token, time-per-output-token, queue wait time, number of cached tokens, cache hit rate, key-value utilization, memory utilization, and if the request satisfied the desired SLO. These are then summarized into higher-level metrics.

Table 2. Methodology Variables and Measurements

Variable	Measurement Role
Routing Strategy	Independent variable used to compare least-queue routing with memory-aware routing strategies.
Prompt Class	Workload category used to distinguish repeated prefixes, unique prompts, short prompts, medium prompts, and long prompts.
p95 End-to-End Latency	Primary tail-latency outcome used to measure user-visible response behavior.
p95 Time to First Token	Interactive latency measure that captures the delay before the first generated token appears.
p95 Time per Output Token	Decode-performance measure used to compare generation speed under load.
SLO Fulfillment	Proportion of requests that satisfy the configured service latency objective.
Cache-Hit Rate	Proportion of successful requests that benefited from cached prefix state.
Cached Tokens	Number of prompt tokens reused from cache for a request.
Key-Value Utilization	Memory-pressure signal based on key-value cache usage on the selected instance.

Variable	Measurement Role
Instance Distribution	Allocation behavior showing how requests are spread across serving instances.

The main comparison is done against the least-queue baseline. Percentage change is used to represent improvement or degradation in terms of the relative performance of the memory-aware strategy regardless of the actual speed of the hardware. For latency, a positive percentage change means that the memory-aware strategy outperforms the least-queue routing strategy by reducing latency. With respect to cache-hit ratio and SLO satisfaction, a positive percentage change means that there was a higher rate of cache hits and SLO satisfaction, respectively. This evaluation is consistent with existing systems evaluation literature where tail latency and SLO satisfaction are essential components of serving quality (Dean and Barroso, 2013; Griggs et al., 2024).

Methodological Summary

The methodology compares routing policies under the same serving environment, workloads, and measurement process. Least-queue routing is used as the baseline because it represents a simple queue-visible policy. Memory-aware strategies are evaluated because LLM serving introduces hidden state in the form of key-value cache pressure, prefix locality, and observed latency behavior. The main purpose of the method is to determine whether those hidden signals improve placement decisions beyond visible queue length.

A memory-aware policy is considered useful if it improves p95 latency, time to first token, time per output token, cache reuse, or SLO fulfillment relative to least-queue routing. If

it does not improve those measurements, then the added complexity of memory-aware routing would not be justified for that workload.

CHAPTER 4

RESULTS

Results Overview

In turn, the workload under consideration explicitly covers the issue that this study considers–the issue of routing when facing the key-value cache pressure and repeat prefixes. The simple least-queue router can observe the amount of queued and processed requests. At the same time, this simple router cannot measure whether the chosen instance has the appropriate cache or whether the key-value cache is under pressure. In this way, this workload provides the main basis for comparison between the queue-based and memory-based approaches.

The Least-Queue routing algorithm is used as the baseline for all percentage-based comparisons in the current chapter. The latency gain will be measured by comparing latencies with those provided by least-queue routing. For example, a positive end-to-end p95 latency difference means that there was a latency decrease compared to the least-queue baseline. When considering the fulfillment of SLOs and cache hit ratio, the chapter describes two metrics–percentage and a percentage difference from the baseline. It corresponds to the evaluation process described in Chapter 3.

Pressure Results

The weighted key-value-pressure workload produced the best results in relation to MELL bin-packing. With least-queue routing, p95 end-to-end latency was equal to 3172.0 ms, but the application of MELL bin-packing allowed reducing it to 2526.2 ms, meaning a drop by 20.4%. Using the same approach, p95 time to first token was decreased from 274.6 ms to 249.6 ms (-9.1%), and p95 time per output token was reduced from 194.5 ms to 148.3 ms (-23.7%).

However, the reduction in median end-to-end latency was relatively smaller. Specifically, with least-queue routing, the median end-to-end latency was equal to 1351.9 ms, while using MELL bin-packing led to a median end-to-end latency of 1312.8 ms (-2.9%). It is worth noting the difference between median and p95 results, as it suggests that memory-aware routing had the strongest effect on slower requests rather than just moving the median value to lower numbers. The result matches the project hypothesis since the impact of key-value cache state is likely to be the most significant in case of high tail latency related to pressure on cache, repetitive prefill, or improper instance placement.

At the same time, MELL bin-packing can be seen as a particular type of routing in terms of choosing the appropriate solution since it took into account the capacity of instances in relation to both memory and CPU. That is why the effect on p95 time per output token was larger than that on median latency because requests were rerouted to appropriate instances and could avoid inefficient memory usage caused by improper cache management.

Latency Comparison

The results of latency for the weighted key-value-pressure scenario are summarized in Table 3. MELL bin-packing showed the highest improvement in p95 latency end-to-end, with a reduction of 20.4% when compared to the least-queue policy. In addition, MELL reduced p95 latency by 18.9%, and intelligent routing decreased latency by 17.1%. The round-robin method resulted in a 16.4% decrease in p95 latency in this scenario. However, it cannot be considered

a general superiority, because the round-robin approach is not aware of key-value cache state and does not benefit from the presence of prefix locality. As such, this improvement should be considered an anomaly, related only to the particular workload under consideration. All latencies provided in Table 3 are expressed in milliseconds.

Table 3. Latency Results for the Weighted Key-Value-Pressure Workload

Strategy	p50 E2E	p95 E2E	p95 TTFT	p95 TPOT	p95 E2E Chg.
Least-queue	1351.9	3172.0	274.6	194.5	0.0%
Cache-affinity	1301.8	2873.5	274.0	174.1	9.4%
Intelligent	1273.5	2630.6	229.5	153.4	17.1%
MELL	1331.5	2573.1	270.7	162.1	18.9%
MELL bin-packing	1312.8	2526.2	249.6	148.3	20.4%
Round-robin	1249.4	2653.3	272.0	149.8	16.4%

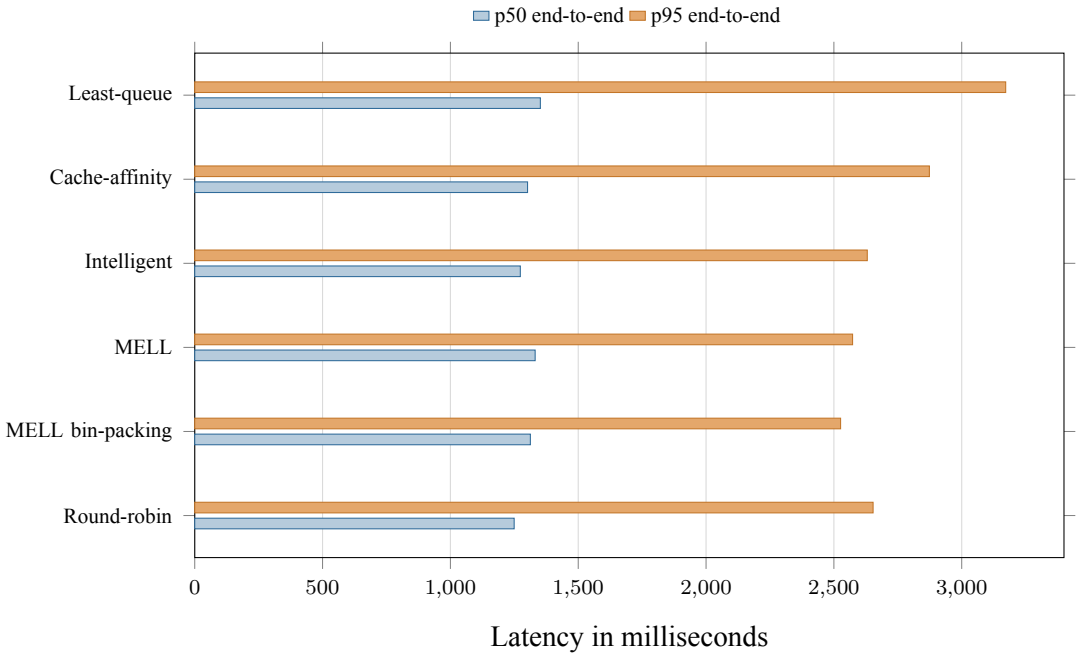

Figure 7. End-to-End Latency Profile by Routing Strategy

The analysis of the table clearly shows that memory-aware policies outperformed least-queue routing on the p95 latency end-to-end criterion. What is more important than this result is the fact that MELL bin-packing achieved the minimum p95 latency per output token of all

policies considered. It means that the proposed policy worked effectively not only during the decoding procedure but also during the placement of requests. As p95 time per token represents sustained generation process, the conclusion can be made that memory awareness benefits were present in sustained generation as well.

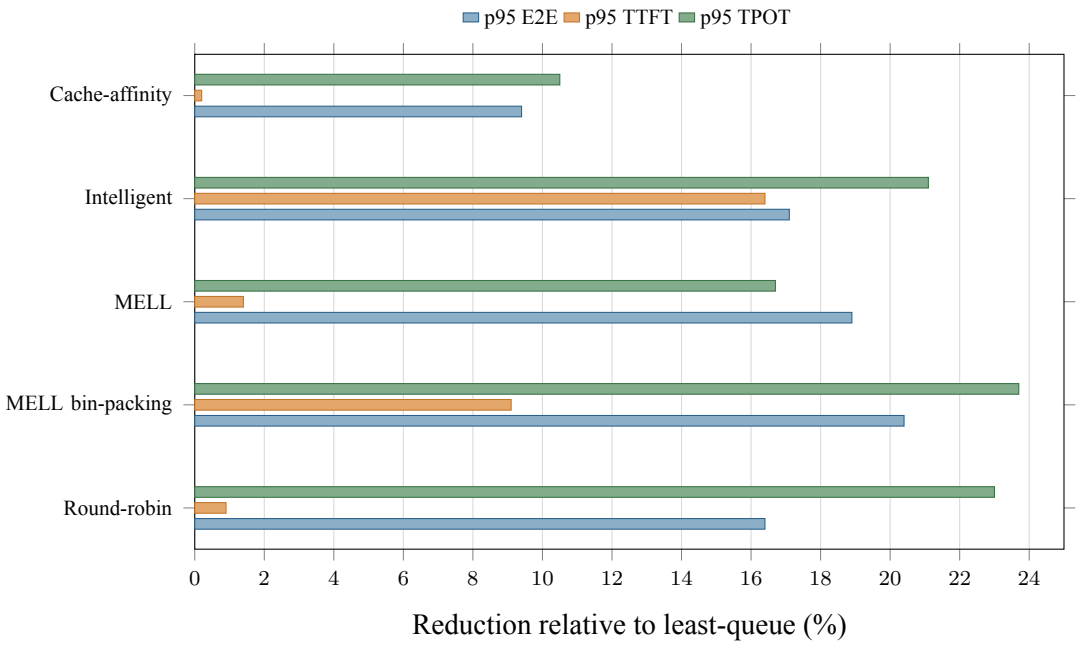

Figure 8. Tail-Latency Metric Reductions Relative to Least-Queue Routing

Cache and SLO Results

Table 4 contains the SLO and cache-related results obtained during the same experiment. Least-queue routing delivered 98.3% SLO satisfaction and 98.3% cache-hit rate. On the other hand, MELL bin-packing managed to deliver 100.0% SLO satisfaction and 100.0% cache-hit rate. Thus, the difference is equal to 1.7 percent points. On average, the number of cached tokens per request grew from 24.6 tokens using least-queue routing to 25.1 tokens using MELL bin-packing, which is a 2.2% increase. Average number of cached tokens per request is reported in Table 4.

Table 4. SLO and Cache Results for the Weighted Key-Value-Pressure Workload

Strategy	SLO Met	Cache Hit	Avg. Tokens	Token Change
Least-queue	98.3%	98.3%	24.6	0.0%
Cache-affinity	100.0%	100.0%	23.2	-5.7%
Intelligent	100.0%	100.0%	25.5	3.9%
MELL	100.0%	100.0%	23.5	-4.4%
MELL bin-packing	100.0%	100.0%	25.1	2.2%
Round-robin	100.0%	100.0%	25.3	2.8%

Thus, the results regarding SLO show that memory-aware placement improved performance of the workload aside from decreasing latencies. Even though the baseline worked effectively, it failed to fulfill one request out of 60 requests completed. However, in contrast, in the current test run, MELL bin-packing fulfilled all requests. Considering that the workload included repeating prefixes and key-value pressure, such results support the claim that cache state can affect chances of meeting a latency goal.

Interpretation of the cache results is rather tricky. The cache-hit rate reached 100.0% for all tested strategies except the baseline strategy. Meanwhile, average number of cached tokens was not always growing when compared to the baseline. In particular, cache affinity and MELL managed to reach a cache-hit rate of 100.0%, but reused fewer cached tokens on average than least-queue routing did. Therefore, a cache hit does not entirely reflect benefits of cache reuse. Specifically, a strategy can find a required cache token but then reuse fewer tokens from the prefix than expected. Therefore, a higher cache-hit rate and a higher number of cached tokens constitute a better result compared to the former metric alone.

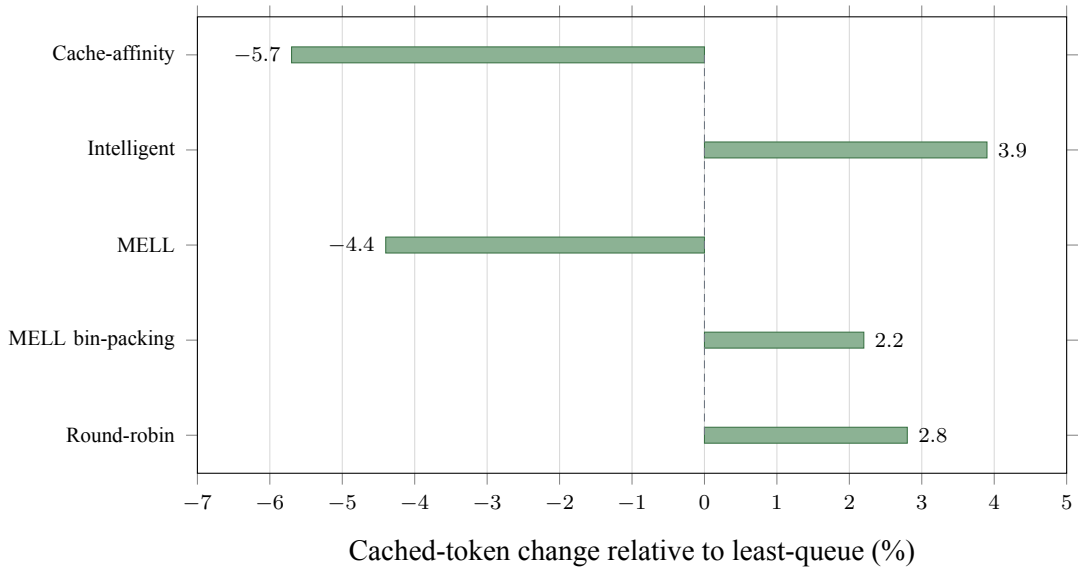

Figure 9. Cached-Token Reuse Change Relative to Least-Queue Routing

Interpretation of Findings

MELL bin-packing provides more benefit for tail requests compared to median requests. Specifically, end-to-end latency on p50 is reduced by only 2.9%, but p95 is cut down by 20.4%. This result shows that a memory-aware routing strategy does not just make typical requests faster; instead, it reduces tail request latencies. This matches the expected effect of cache pressure and inefficient placement on high-percentile requests.

The p95 time-to-first-token statistic also supports the usefulness of memory-awareness even though the reduction is much less dramatic compared to p95 time-per-output-token statistic. Time per output token is reduced by 23.7%, while time to first token is decreased by only 9.1%. Therefore, the effect occurs in both prefill and decode phases of inference, but more importantly for token generation itself. A request placed on a high-pressure instance can continue suffering latency problems during the decode phase after the first token is generated.

Finally, a comparison of different strategies of memory-awareness shows that a simple memory-awareness is not enough. While p95 end-to-end latency was decreased by 9.4% using a cache affinity technique, the number of cached tokens used turned out lower than that achieved

via least-queue routing. Also, MELL bin-packing reduced end-to-end latency by 18.9% while having an average cached-token reuse lower than the baseline value. MELL bin-packing significantly reduced p95 latency while showing higher cached-token reuse.

Summary of Results

The results obtained through the key-value-pressure test provide support for the cache-aware routing hypothesis. While in the least-queue routing policy, visible queue length was considered as a load signal, in MELL bin-packing algorithm, memory pressure, fit, queue pressure, and cache behavior parameters were used for making routing decisions. With the MELL bin-packing algorithm implemented in the least-queue routing policy, it was possible to achieve such improvements as a 20.4% improvement in p95 end-to-end latency, 9.1% improvement in p95 time to first token, 23.7% improvement in p95 time per output token, 2.9% improvement in median end-to-end latency, an increase in service level objective (SLO) fulfillment to 100% from 98.3%, an improvement in the cache-hit rate to 100% from 98.3%, and an increase in the mean number of cached tokens per request by 2.2%.

The obtained results confirm the research questions set forth in Chapter 1 regarding the key-value-pressure workload. In comparison with memory-oblivious routing, memory-aware routing has shown better results in p95 end-to-end latency, p95 time to first token, p95 time per output token, SLO fulfillment, and cache behavior. The improvements in the latter parameters were greater at the tails of the distribution compared to median values.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Summary of the Project

The purpose of this paper is to determine whether information from the key-value cache state might be used to guide routing decisions in multiple instances of LLM serving. The main issue here is that least-queue routing depends upon visibility of queue length but cannot see cache pressure, prefix reuse, and appropriateness of requests to a certain instance. Since transformer inference is significantly dependent on key-value cache memory and scheduling dynamics (Kwon et al., 2023; Yu et al., 2022; Pope et al., 2023), queue length alone might not give the entire picture of the load.

The research compares least-queue routing with memory-based routing approaches where queue pressure, key-value cache pressure, prefix locality, latency observed, and capacity fit were considered. In particular, the approach most relevant to this study was MELL bin-packing since this type of routing can be viewed as the task of resource placement. This idea is relevant in the context of modern literature devoted to LLM serving.

Research Question Findings

The findings validate the hypotheses posed. MELL bin-packing resulted in a 20.4% reduction in the p95 end-to-end latency compared to least-queue routing in the key-value-pressure

scenario. Furthermore, MELL bin-packing led to a 9.1% reduction in the p95 time to first token and a 23.7% reduction in the p95 time per output token. The higher difference in the p95 time per output token suggests that the memory-aware allocation facilitated the sustained production, not just the initial request acceptance. The findings also suggest that the improvement was more significant in the tail than the median; median end-to-end latency had a 2.9% decrease, while the p95 end-to-end latency had a 20.4% decrease.

Implications for LLM Serving

The takeaway is that queue size cannot be considered a universal metric for characterizing LLM serving traffic. Queue size is helpful but does not take into consideration key-value cache pressure and value of cached prefixes. It was shown that the routing plane may use state information provided by the serving plane, such as key-value cache information and prefix locality.

The results also show that cache-awareness needs to be considered together with fit and capacity aspects. Cache affinity had a 100.0% hit rate but did not improve token re-use over baseline. On the other hand, MELL bin-packing produced better results by lowering p95 latencies and increasing token re-use at the same time. Thus, queue size pressure, cache size pressure, and fit should be considered together during routing rather than one at a time.

Limitations of the Findings

These findings can be considered comparative in nature and achieved in a controlled experiment environment. The current research does not claim MELL bin-packing will improve latency in all possible scenarios for a particular workload, scale of models, hardware setup, or any

other factors involved. The greatest improvements could be achieved in the case of a weighted key-value-pressure workload aimed at cache pressure and repeated prefixes.

Another potential drawback is that this research focuses specifically on the routing component of the system and not on the whole system for inference tasks. Modern inference engines include such components as batching, prefill/decode scheduling, prefix re-use, offloading, and awareness of the hardware (Yu et al., 2022; Sheng et al., 2023; Patel et al., 2024; Griggs et al., 2024). As such, the findings can be viewed as demonstrating that routing can benefit from memory state.

Future Work

A next step would be testing the performance of the routing policy on more realistic and heterogenous traffic scenarios. The current paper uses simulated workloads to demonstrate the impact of key-value pressure, repeated prefixes, and SLO sensitivity. Another step would include testing the router with request traces that have bursty arrivals, different conversation lengths, RAG prompts, and varied output token distributions. This test will help us understand if the positive effect shown by the key-value-pressure workload scenario remains under more challenging circumstances.

Next, research in the area should expand the scope of the Mélange approach from heterogeneous allocation of GPUs to autoscaling of resources in terms of servers and GPUs. The work has proven that the workload size and SLOs define the best configuration of computing resources (Griggs et al., 2024). A reasonable follow-up is the creation of a system that controls both when to scale and how many and what type of servers and GPUs to use during scaling. At this point, memory awareness will become an important part of the overall resource management process that allocates requests to appropriate instances based on autoscaling information.

Final Conclusion

The experiment shows that memory-aware routing provides additional information to multi-instance LLM serving compared to queue length alone. Though the least queue is intuitive and easy to implement, it takes all known work as equal without any differentiation. On the other hand, inference of LLMs has different features, such as variance in prompt length, output length, prefix reuse, and pressure on key-value cache.

MELL bin-packing solves this problem by checking whether enough memory and computing power are available in the instance while also considering queue pressure and cache pressure. The largest improvement was shown in tail latency, where p95 end-to-end latency decreased by 20.4%. These results indicate that the routing layer becomes part of the LLM serving process when considering cache pressure, prefix locality, and capacity fit.

REFERENCES

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., & Ramjee, R. (2024). Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation* (pp. 117–134).

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., …Zhu, T. (2023). *Qwen technical report*. arXiv. `https://arxiv.org/abs/2309.16609`

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., …Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM, 56*(2), 74–80.

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., & Zuo, P. (2024). AttentionStore: Cost-effective attention reuse across multi-turn conversations in large language model serving. In *2024 USENIX Annual Technical Conference*.

Gerganov, G. (2023). *llama.cpp: LLM inference in C/C++*. GitHub. `https://github.com/ggml-org/llama.cpp`

Griggs, T., Liu, X., Yu, J., Kim, D., Chiang, W.-L., Cheung, A., & Stoica, I. (2024). *Mélange: Cost efficient large language model serving by exploiting GPU heterogeneity*. arXiv. `https://arxiv.org/abs/2404.14527`

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles* (pp. 611–626).

Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems, 12*(10), 1094–1104.

NVIDIA. (2023). *TensorRT-LLM*. GitHub. `https://github.com/NVIDIA/TensorRT-LLM`

Patel, P., Choukse, E., Zhang, C., Goiri, I., Shah, A., Maleki, S., & Bianchini, R. (2024). Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., & Dean, J. (2023). Efficiently scaling Transformer inference. In *Proceedings of Machine Learning and Systems, 5*, 606–624.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Re, C., Stoica, I., & Zhang, C. (2023). FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (pp. 31094–31116).

Sun, B., Huang, Z., Zhao, H., Xiao, W., Zhang, X., Li, Y., & Lin, W. (2024). Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Roziere, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and efficient foundation language models*. arXiv. `https://arxiv.org/abs/2302.13971`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems, 30*.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., & Chun, B.-G. (2022). Orca: A distributed serving system for Transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation* (pp. 521–538).

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., & Sheng, Y. (2023). *SGLang: Efficient execution of structured language model programs*. arXiv. `https://arxiv.org/abs/2312.07104`

APPENDIX A: PROJECT ARTIFACTS AND ACCESS LINKS

Appendix A lists the main artifacts used to implement and demonstrate the proposed memory aware LLM Routing algorithm. These artifacts support the project because the report presents an applied systems project implemented outside the report itself.

Table 5. Project Artifact Access Links

Artifact	Access Information	Purpose
Source Code Repository	https://github.com/Kartikiv/llm-inference-engine	Contains the router, serving-instance code, experiment harness, deployment assets, frontend interface, and supporting documentation.
Live Demo Recording	https://drive.google.com/file/d/12-_DVZ2cqtZEPAU4Vyh3BJLKAQESxMAF/view?usp=sharing	Provides a recorded demonstration of the running system, routing interface, and experiment workflow.
Public Test Endpoint	`http://35.222.234.243:30300/`	The project services are deployed and available for public testing. If the endpoint is unreachable from a university network, readers should try a personal or unrestricted network because some institutional networks block unknown public IP addresses or nonstandard ports.
Experiment Results	Project experiment output files	Contains the measured latency, cache, SLO, and strategy-comparison results used to prepare the results chapter.

APPENDIX B: PROJECT INTERFACE SCREENSHOTS

Screenshots of the deployed project interface are provided below. Figures capture the screens used to set up the experiment, check the health of the serving instances, perform live testing, and make chat requests through the routing layer.

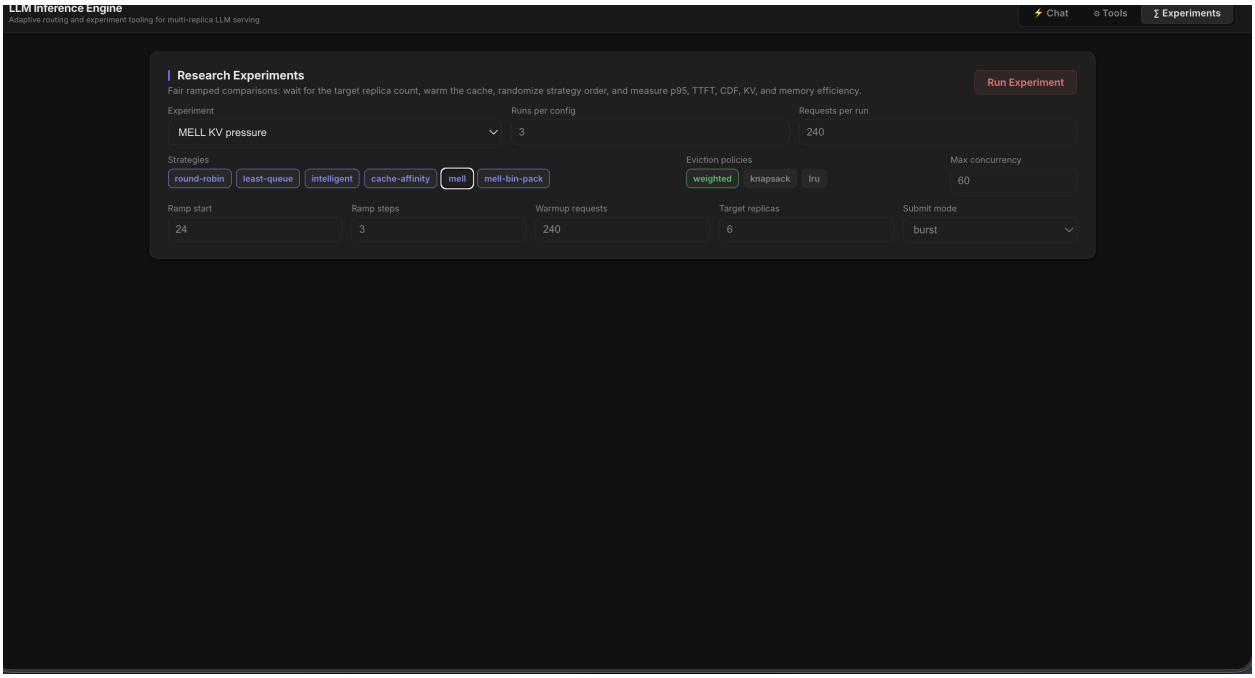

Figure 10. Experiment configuration screen for controlled routing studies. Source: Project deployment screenshot captured from the public demonstration interface.

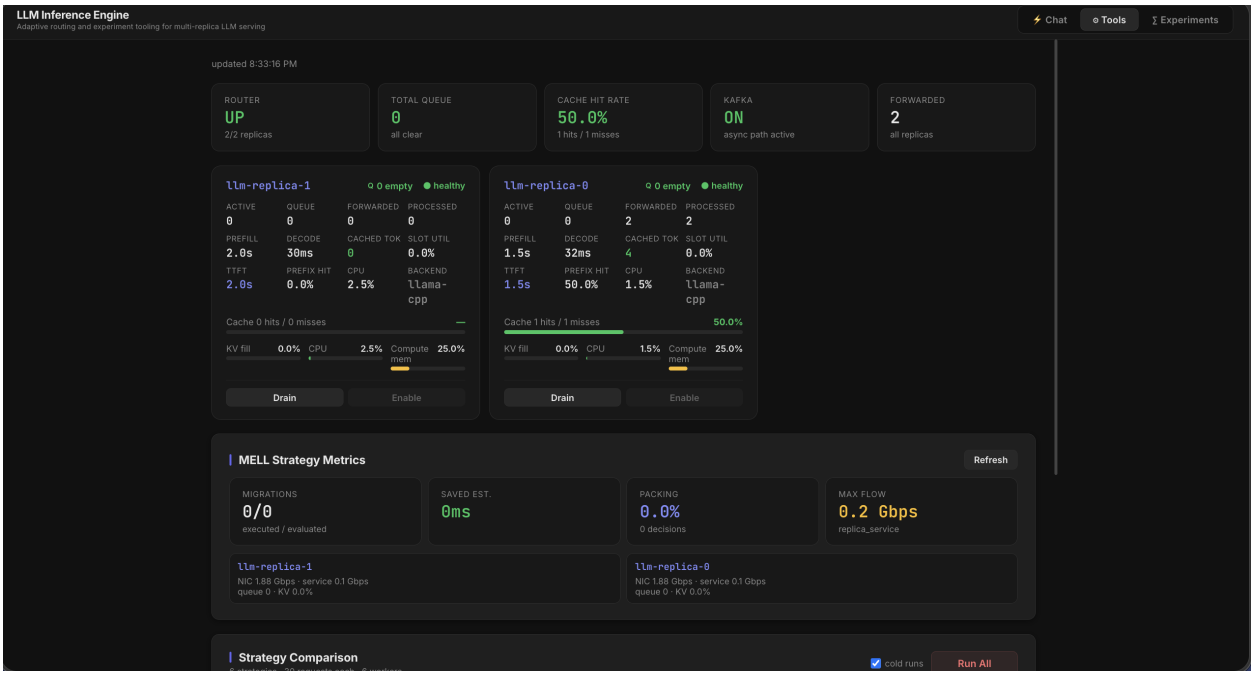

Figure 11. Tools dashboard with service metrics. Source: Project deployment screenshot captured from the public demonstration interface.

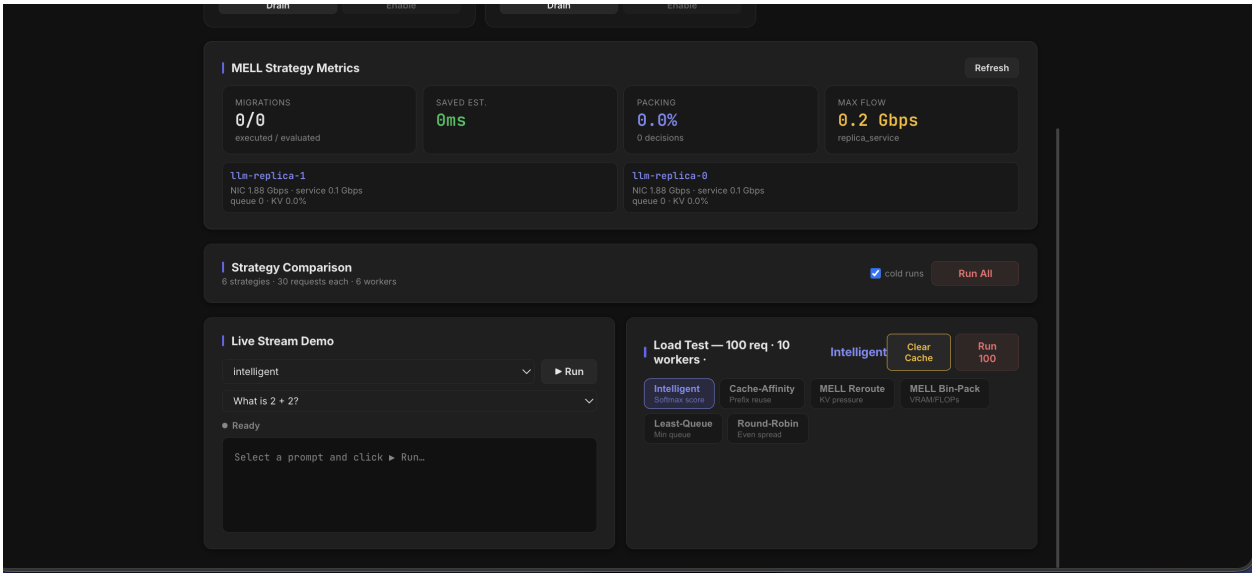

Figure 12. Tools dashboard with live testing controls. Source: Project deployment screenshot captured from the public demonstration interface.

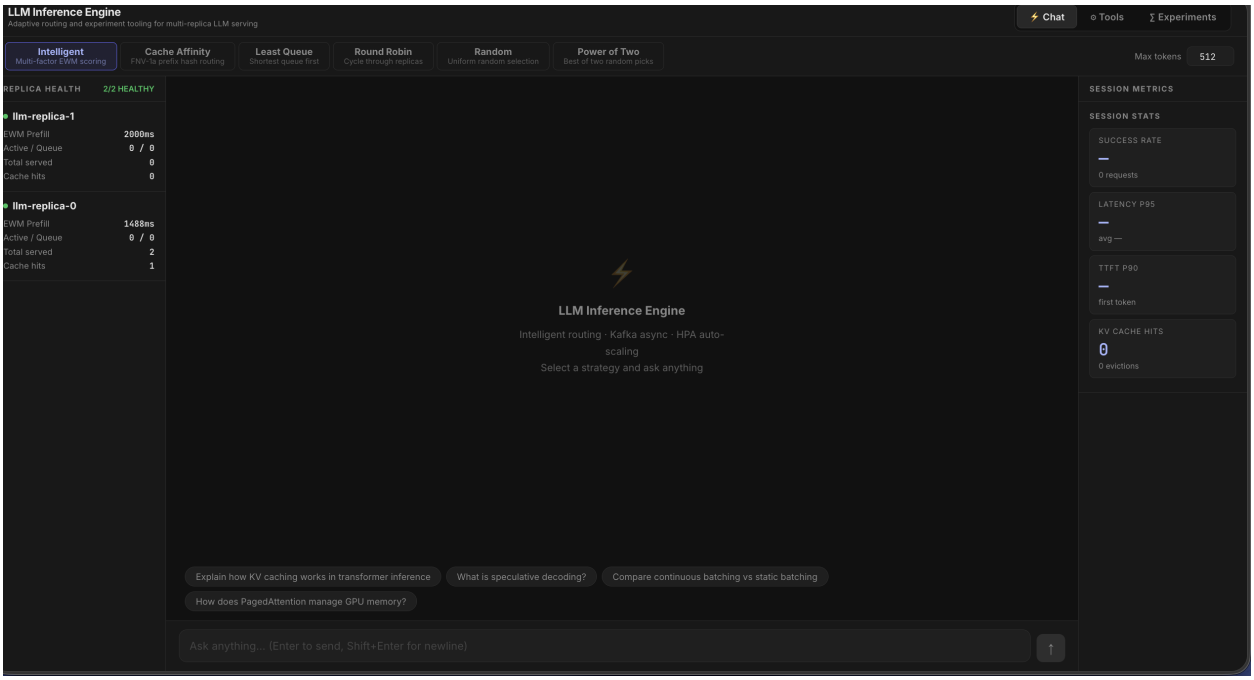

Figure 13. Chat interface with session metrics. Source: Project deployment screenshot captured from the public demonstration interface.

APPENDIX C: PROJECT DIRECTORY

The implementation is organized as a multi-module project. Table 6 summarizes the major directories and their purpose in the system.

Table 6. Project Directory Overview

Component	Directory or File	Purpose
Router service	`router/`	Contains the Spring Boot routing service, request admission endpoints, routing strategies, instance-state polling, Kafka gateway, migration coordination, and experiment API.
Serving-instance service	`replica-sim/`	Contains the serving-instance simulator and llama.cpp-backed inference path, including queue handling, key-value slot scheduling, cache statistics, and state reporting.

Component	Directory or File	Purpose
Experiment runner	`load-tester/`	Contains the workload execution utilities used to generate repeatable requests for routing-strategy comparison.
Frontend interface	`frontend/src/`	Contains React screens for chat, tools, live testing, experiments, and strategy metrics.
Deployment assets	`k8s/`, `docker-compose.yml`, `Dockerfile.router`, `Dockerfile.replica`	Contains Kubernetes and Docker assets used to deploy the router, serving instances, Kafka, and frontend interface.
Configuration files	`config/` and module-level `application*.yml` files	Contains environment-specific settings for local, Docker, Kubernetes, and stress-test execution.
Analysis scripts	`scripts/analyze.py` and `scripts/max_flow_analysis.py`	Contains supporting scripts for experiment analysis and strategy-comparison processing.

The directory organization separates implementation concerns so that the router, serving instances, frontend, experiment runner, and deployment files can be modified independently while remaining part of the same project repository.

APPENDIX D: CODE WALKTHROUGH

Appendix D maps the memory-aware routing experiment to the source files used in the project. This connection links the system design to its implementation in the source code repository.

Table 7. Core Implementation Walkthrough

Source Path	Implementation Role
`router/src/main/java/com/example/router/routing/RoutingEngine.java`	Selects the routing strategy for each incoming request, retrieves healthy serving-instance candidates, applies admission limits, and invokes migration coordination when a memory-aware strategy chooses a different target.
`router/src/main/java/com/example/router/routing/strategies/LeastQueueStrategy.java`	Implements the queue-only baseline by selecting the serving instance with the smallest observable queue pressure. This file provides the comparison point for the percentage-based results in Chapter 4.
`router/src/main/java/com/example/router/routing/strategies/IntelligentStrategy.java`	Implements a multi-factor routing score using queue pressure, estimated prefill cost, output-token cost, key-value cache pressure, observed latency, and cache reuse signals.

Source Path	Implementation Role
`router/src/main/java/com/example/router/routing/strategies/MELLStrategy.java`	Implements the MELL-style memory-aware strategy by combining intelligent selection with a cost-benefit decision for key-value migration when a lower pressure instance is available.
`router/src/main/java/com/example/router/routing/strategies/GreedyBinPackingStrategy.java`	Implements the MELL bin-packing variant. It estimates request memory and compute requirements, compares those requirements with remaining instance budgets, penalizes overload, and rewards useful cached-token history.
`router/src/main/java/com/example/router/replica/StatsPoller.java`	Polls serving-instance state so the router can observe queue depth, active requests, cache utilization, latency estimates, cached-token reuse, and health status before making placement decisions.
`router/src/main/java/com/example/router/controller/ExperimentController.java`	Exposes the experiment API used by the interface to launch controlled workload runs, retrieve job history, and export CSV measurements for analysis.
`replica-sim/src/main/java/com/example/replicasim/service/ReplicaSimService.java`	Processes inference requests through the llama.cpp-backed serving path, maintains queue and cache measurements, reports time-to-first-token and decode statistics, and exposes cache behavior to the router.

Source Path	Implementation Role
`replica-sim/src/main/java/com/example/replicasim/service/KVSlotScheduler.java`	Assigns requests to key-value cache slots, tracks prefix matches, supports cache-aware slot reuse, and applies the configured eviction policy.
`frontend/src/Experiments.jsx` and `frontend/src/Tools.jsx`	Provide the user-facing controls for experiment configuration, live testing, strategy comparison, and observation of router and serving-instance metrics.