CARRIER TO CARRY: MODELING KV CACHE DYNAMICS FOR PROVABLY STABLE ROUTING IN DISTRIBUTED LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

In distributed large language model (LLM) serving, key-value (KV) cache reuse reduces time-to-first-token (TTFT) by avoiding redundant prefill computation, yet exploiting shared prefixes remains challenging because caching affinity tends to concentrate requests on a few instances while pending requests alter the cached context available for subsequent reuse. To formalize this challenge, we introduce *CARRIER*, a mathematical model to represent request arrivals by affinity group and distributed LLM serving system with cache reuse, retention, and eviction across instances. Building on *CARRIER*, we develop *CARRY*, a routing algorithm that learns *serving potential*, a state function that balances immediate prefix reuse against instance load while accounting for future cache usefulness, thereby enabling a unified theoretical analysis that establishes system serving stability, proves robustness to bounded estimation errors, and quantifies the system serving capacity advantage under *CARRY* over baselines. Experiments across three benchmarks show that *CARRY* achieves improvements of up to 75.2% cache hit rate, 82.1% and 68.6% of TTFT reductions at P50 and P90, 77.8% and 68.6% of latency reductions at P50 and P90 over the state-of-the-art methods.

1 INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In large language model (LLM) serving, conversation, tool and agent, and long context applications repeatedly carry shared history, instructions, observations, or documents across requests (Ding et al., 2023; Qin et al., 2024; Liu et al., 2024a; Bai et al., 2024). Reusing their key value (KV) states reduces repeated prefill computation and therefore time to first token (TTFT) (Gim et al., 2024; Zheng et al., 2024; Ye et al., 2024; Juravsky et al., 2024).

At cluster scale, however, cache reuse is inseparable from load. Directing related requests to one instance improves cache hit but may create an affinity hotspot, whereas spreading them across instances improves load balance but repeats prefill. Production traces further show that KV reuse is skewed and varies across request classes (Wang et al., 2025a), and serving demand may be highly bursty (Wang et al., 2025c; Li et al., 2023). Meanwhile, retention and eviction continue to change cache contents while requests wait. Consequently, these dynamics make the cache observed at arrival and current instance load insufficient for routing, motivating a model that follows cache dynamics through the waiting queue and jointly evaluates immediate reuse, instance load, and future cache usefulness.

To address this gap, we introduce Cache Aware Representation of Requests and Instances under Eviction and Retention Dynamics (CARRIER), which propagates group structured requests through each instance's waiting queue to recover instance specific cache hits, effective workload, and serving workload, then formulates cache aware routing as a Markov decision process (MDP). Building on *CARRIER*, we develop Cache Aware Routing for Reuse Yield (CARRY), which learns a *serving potential* that balances reuse, instance load, and future cache usefulness. Through *CARRIER*, the serving stability margin decomposes into reuse yield and load imbalance, enabling us to derive conditions for system serving stability, show that *CARRY*'s serving capacity inner region contains

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

those of *Cache Greedy* and *Load Balance*, establish robustness to cache state and decode size errors, and prove that successive *CARRY* updates vanish almost surely. Across three benchmarks and four extreme cases, *CARRY* consistently outperforms state of the art methods in TTFT, latency, cache hit rate, and load balance.

1.2 Related Work

LLM serving systems improve continuous batching, KV memory allocation, phase separation, migration, and request scheduling (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024b; Zhong et al., 2024; Patel et al., 2024; Sun et al., 2024; Wu et al., 2026a). Other systems expose shared context for reuse (Gim et al., 2024; Zheng et al., 2024; Ye et al., 2024; Juravsky et al., 2024), move or persist KV states across the serving hierarchy (Liu et al., 2024b; Yao et al., 2025; Hu et al., 2024; Jin et al., 2025; Strati et al., 2024; Liu et al., 2025), or improve token retention and dynamic KV memory allocation within an instance (Zhang et al., 2023; Prabhu et al., 2025). *CARRIER* complements them by modeling the cache that requests encounter after retention and eviction in each waiting queue.

Cache aware routing combines locality with current or estimated load. *Min TTFT* minimizes estimated waiting and recomputation time (Qin et al., 2025), *Preble* coordinates cache reuse and load (Srivatsa et al., 2025), and *DualMap* adapts the power of two choices (Mitzenmacher, 2002) to this setting (Yuan et al., 2026). Randomized KV routing also studies routing and eviction jointly (Wu et al., 2026b). *CARRIER* instead advances cache state through earlier requests in the waiting queue to expose each choice's effective work and future cache usefulness.

Prior work on solving MDPs uses temporal difference learning with function approximation (Sutton et al., 2018; Tsitsiklis & Van Roy, 1996) and constrained stochastic approximation (Robbins & Monro, 1951; Borkar & Meyn, 2000). Stochastic network analysis provides tools for characterizing system stability (Tassiulas & Ephremides, 1990; Neely, 2010; Meyn & Tweedie, 2012). Model routing studies a related but distinct assignment problem by selecting among models with different cost and quality (Chen et al., 2023; Ong et al., 2025), whereas *CARRY* assigns requests among instances serving the same model.

1.3 Our Contributions

- **Modeling.** We introduce *CARRIER*, a mathematical model, to jointly represent group structured request arrivals and distributed serving under retention and eviction, making cache evolution through waiting queues tractable and enabling MDP based routing and system analysis.
- **Algorithm.** We develop *CARRY*, a routing algorithm built on *CARRIER*, using a learned serving potential to turn dynamic cache information into routing decisions that balance immediate reuse, instance load, and future cache usefulness.
- **Theory.** We establish a unified theory under *CARRIER*, characterizing system serving stability, quantifying *CARRY*'s serving capacity advantage over baselines, bounding the effects of cache state and decode size errors, and proving that successive *CARRY* updates vanish almost surely.
- **Simulation infrastructure.** We implement a nominal simulator from *CARRIER* and a cache aware practical simulator, validating *CARRIER*'s modeling consistency, *CARRY*'s performance, and the theoretical results.

2 Modeling and Formulation

CARRIER maps application context sharing to KV reuse and workload in distributed LLM serving. Figure 1 summarizes its request representation, arrival process, and serving system.

2.1 Request Arrival Model

Cache aware request representation. Requests with reusable context form a *cache affinity group*: conversation requests are grouped by session, tool and agent requests by recurring instructions or tool context, and long context requests by shared document (Figure 1). For groups $\mathcal{M} = \{1, \dots, M\}$, request ℓ in group m has request profile

$$J_{m,\ell} = (m, p_{m,\ell}, d_{m,\ell}, b_{m,\ell}, q_{m,\ell}), \quad 0 \le b_{m,\ell} \le p_{m,\ell}, \quad 0 \le q_{m,\ell} \le p_{m,\ell} + d_{m,\ell}, \tag{1}$$

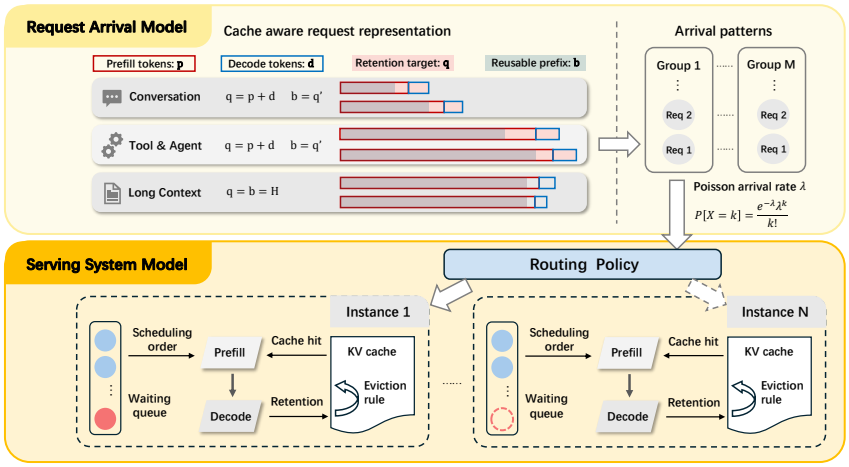

Figure 1: Overview of *CARRIER*. The request arrival model groups requests by reusable context, represents each request through its prefill tokens, decode tokens, retention target, and reusable prefix, then models the arrival times of each affinity group as a Poisson process. A routing policy assigns arriving requests to instances, where the waiting queue and scheduling order govern prefill and decode serving while cache hits, retention, and eviction update the KV cache available to later requests.

where the nonnegative KV block counts p and d denote prefill and decode work, b bounds reusable prefix blocks, and q specifies compatible context retained for later requests.

Arrival patterns. To isolate request timing while preserving correlations among request attributes, *CARRIER* models stochastic arrivals by a global Poisson process with request rate $\lambda > 0$. Let k index requests, let t_k be the arrival time of request k, and let J_k be its request profile. The empirical request set $\mathcal{J}_R = (J^{(1)}, \ldots, J^{(R)})$ contains R observed profiles indexed by r:

$$t_{k+1} - t_k \overset{\text{i.i.d.}}{\sim} \text{Exp}(\lambda), \qquad \widehat{F}_R = \frac{1}{R}\sum_{r=1}^{R} \delta_{J^{(r)}}, \tag{2}$$
$$J_k \overset{\text{i.i.d.}}{\sim} \widehat{F}_R, \qquad \{t_{k+1} - t_k\} \perp \{J_k\}.$$

Here δ_J denotes a unit point mass at request profile J. Sampling full profiles preserves the dependence among group, prefill, decode, reuse, and retention. Workloads derived from application scenarios preserve the recorded request order within each cache affinity group and use the timing pattern configured for that workload. Appendix A.2 specifies the empirical sampling construction and trace ordering.

2.2 Serving System Model

System components. A router dispatches each arrival to one of N homogeneous instances $\mathcal{N} = \{1, \ldots, N\}$. Instance n has waiting queue $\mathcal{Q}_n$, serving batch $\mathcal{B}_n$, and KV cache $U_n = (u_{n,m})_{m \in \mathcal{M}}$ of capacity C. We study serving batches that contain one request.

System dynamics. We study first come first served (FCFS) scheduling, under which an idle instance selects the oldest request in its waiting queue. Decode proceeds at block rate $\mu > 0$, while prefill proceeds at rate $\upsilon\mu$ for speed factor $\upsilon > 1$. For a request $J = (m, p, d, b, q)$ selected for serving on instance n, its cache hit size is $h_n = \min\{b, u_{n,m}\}$, leaving $p - h_n$ prefill blocks and d decode blocks to serve and giving serving time $[d + (p - h_n)/\upsilon]/\mu$. Cache retention and eviction follow rule E. After J completes, the instance cache retains up to q compatible blocks and evicts other blocks when necessary, with transition $(U_n^+, \xi_n^+) = E((U_n, \xi_n), J)$, where ξ_n is the eviction rule state. Appendix A.2 specifies E.

Serving objective and metrics. The cache available when a queued request reaches the serving batch depends on the requests ahead of it. *CARRIER* therefore propagates the cache state through

these requests. Let $\widehat{h}_{n,J}$ be the cache hit size for request J on instance n after this propagation. Its effective workload $G_{n,J}$ and serving workload $L_{n,J}$ are

$$G_{n,J} = p(J) + d(J) - \widehat{h}_{n,J}, \quad L_{n,J} = d(J) + \frac{p(J) - \widehat{h}_{n,J}}{\upsilon}. \tag{3}$$

Appendix A.2 details this FCFS cache state propagation. For residual prefill and decode blocks $\widetilde{p}_n$ and $\widetilde{d}_n$ in $\mathcal{B}_n$, the instance workload W_n and system unfinished request workload $B(t)$ at continuous serving time $t \geq 0$ are

$$W_n = \widetilde{p}_n + \widetilde{d}_n + \sum_{J \in \mathcal{Q}_n} G_{n,J}, \qquad B(t) = \sum_n \sum_{J \in \mathcal{Q}_n(t) \cup \mathcal{B}_n(t)} [p(J) + d(J)], \tag{4}$$

where residual blocks vanish for an empty batch. The system workload is $\mathbf{W} = (W_n)_{n \in \mathcal{N}}$. Let X_t denote the complete serving state at time t and define the unfinished request workload Lyapunov function $V_\theta^B(X) = \exp\{\theta B(X)\}$ for $\theta > 0$. Starting from an empty system, we define the serving system under policy π as stable if $\sup_{t \geq 0} \mathbb{E}_\pi[V_\theta^B(X_t)] < \infty$ for some $\theta > 0$. This condition follows from an auxiliary exponential Lyapunov function $\widetilde{V}_\theta$ that satisfies $\mathcal{L}_\pi \widetilde{V}_\theta(X) \leq -c\widetilde{V}_\theta(X) + b\mathbf{1}\{X \in \mathcal{C}\}$ for $c, b > 0$ and a bounded workload set $\mathcal{C}$, where $\mathcal{L}_\pi$ is the process generator under π; its negative drift outside $\mathcal{C}$, together with FCFS workload clearing, yields the stated bound on V_θ^B. For a fixed request distribution and serving configuration, the serving capacity region under π is $\mathcal{R}_\pi = \{\lambda \geq 0 : \text{the serving system is stable under } \pi\}$.

Let $A_k \in \mathcal{N}$ be the instance selected for request J_k, and let h_k^π be its realized cache hit under policy π. Let $\bar{s} = \mathbb{E}[p + d]$ denote mean request size and $\bar{\ell} = \mathbb{E}[d + p/\upsilon]$ for mean serving workload without reuse. For positive offered traffic and whenever the limits exist, define

$$\rho_{\pi,n}^L = \lim_{T \to \infty} \frac{1}{\mu T} \mathbb{E}_\pi \left[\sum_{k : t_k < T} \mathbf{1}\{A_k = n\} L_{n,J_k} \right], \tag{5}$$

$$\mathcal{Y}_\pi = \lim_{T \to \infty} \frac{\mathbb{E}_\pi[\sum_{k : t_k < T} h_k^\pi]}{\lambda T \bar{s}}, \tag{6}$$

$$\mathcal{I}_\pi^L = \max_n \rho_{\pi,n}^L - \frac{1}{N} \sum_n \rho_{\pi,n}^L, \qquad \varepsilon_\pi^L = 1 - \max_n \rho_{\pi,n}^L, \tag{7}$$

Here $\rho_{\pi,n}^L, \mathcal{Y}_\pi, \mathcal{I}_\pi^L$, and ε_π^L denote normalized per instance offered service load, reuse yield, normalized service load imbalance, and the serving stability margin.

2.3 MDP Formulation

State. At routing epoch k, define the complete state as

$$X_k = \left(J_k, \left\{ (\mathcal{Q}_{n,k}, \mathcal{B}_{n,k}, \widetilde{p}_{n,k}, \widetilde{d}_{n,k}, U_{n,k}, \xi_{n,k}) \right\}_{n \in \mathcal{N}} \right), \tag{8}$$

where the residual blocks are zero for an empty serving batch. The routing observation is $S_k = (\mathbf{W}_k, \chi_k, J_k)$, where the queue aware cache view χ_k contains each instance's cache and eviction state after cache state propagation through its waiting queue.

Action. Action $A_k = a \in \mathcal{N}$ assigns J_k to instance a. To evaluate this assignment, the queue aware action state $(\mathbf{W}_k^{(a)}, \chi_k^{(a)})$ adds G_{a,J_k} to W_a and propagates cache rule E through the updated waiting queue of instance a, leaving all other coordinates unchanged.

Transition. The request enters the selected instance's waiting queue. FCFS scheduling moves requests into serving batches, phase work decreases at rates $\upsilon\mu$ and μ, completions apply cache rule E, and the workload model produces the next arrival. These events define the transition kernel of X_k.

Objective. Let $B_k := B(t_k^-) = B(X_k)$ be the per step cost and let $\gamma \in (0, 1)$ be the discount factor. For any routing policy π, its action cost satisfies the Bellman expectation equation, while the

routing objective selects a policy minimizing discounted unfinished request workload:

$$Q^\pi(X,a) = B(X) + \gamma \mathbb{E}_\pi[Q^\pi(X',A') \mid X, A=a]\,,$$
$$\pi^\star \in \arg\min_\pi \; \mathbb{E}^\pi_{X_0}\left[\sum_{k\geq 0}\gamma^k B_k\right]. \tag{9}$$

Here X' and A' denote the next state and action. The Bellman equation uses the complete state X because observation S need not be Markov; *CARRY* approximates this action cost from S. Appendix A.2 specifies the continuous event evolution between routing epochs.

3 Algorithm

CARRY maps each action state from *CARRIER* to a learned cost that combines immediate prefix reuse, instance workload, and retained reusable context.

3.1 Value Approximation

CARRY approximates the policy action cost Q^π by mapping *CARRIER*'s workload and KV cache state to a learned cost.

Let the affinity group frequency ω_m be the fraction of requests in the empirical request set that belong to group m, with $\sum_m \omega_m = 1$, and let F_m be the conditional distribution of reusable prefix size R_m. Appendix A.2 specifies their estimates. To quantify retained context, let $V_n = (v_{n,m})_{m\in\mathcal{M}}$ be the queue aware cache state of instance n. We define its future cache usefulness $H(V_n)$ as

$$H(V_n) = \sum_{m\in\mathcal{M}} \omega_m \mathbb{E}_{R_m\sim F_m}[\min\{v_{n,m}, R_m\}]\,. \tag{10}$$

Each inner expectation is the expected cache hit size for a request from group m assigned to instance n. Using the system workload $\mathbf{W}$ and queue aware cache view χ from Section 2, and given positive scales $\sigma_W, \sigma_{W^2}, \sigma_H, \sigma_{H^2}$, the instance feature vector ϕ_n and system feature vector Φ are

$$\begin{aligned}\phi_n(W_n, V_n) &= \begin{bmatrix} W_n/\sigma_W & W_n^2/\sigma_{W^2} & H(V_n)/\sigma_H & H(V_n)^2/\sigma_{H^2}\end{bmatrix}^\top,\\ \Phi(\mathbf{W},\chi) &= \mathrm{col}_{n\in\mathcal{N}}\,\phi_n(W_n,V_n).\end{aligned} \tag{11}$$

The queue aware cache state V_n determines feature values, while its accompanying eviction policy state determines the transition.

To capture how an action changes both effective workload and retained reusable context, we define the serving potential $\Psi_w(\mathbf{W},\chi)$, parameterized by weight vector w, and the action cost approximation $\widehat{Q}(S,a;w)$ as

$$\Psi_w(\mathbf{W},\chi) = w^\top \Phi(\mathbf{W},\chi), \qquad \widehat{Q}(S,a;w) = \Psi_w(\mathbf{W}^{(a)},\chi^{(a)}). \tag{12}$$

For action a, the effective workload $G_a := G_{a,J_k}$ changes the workload coordinates through $W_a + G_a$, coupling saved prefill work with instance load. The H coordinates account for later reuse.

3.2 Weight Update

CARRY learns w from serving trajectories by on policy temporal difference learning. A softmax produces routing probabilities, and projection constrains the update.

The feasible weight set $\mathcal{W}$ is

$$\mathcal{W} = \{w : |w_{n,j}| \leq w_{\max}\;\forall n\in\mathcal{N},\; j\in\{1,2,3,4\}, \quad w_{n,2}\geq \epsilon_w > 0\;\forall n\in\mathcal{N}\}\,, \tag{13}$$

where the maximum weight magnitude $w_{\max}$ and positive quadratic weight lower bound ϵ_w define the parameter domain. With policy temperature $\iota > 0$, the routing policy π_w is

$$\pi_w(a \mid S) = \frac{\exp[-\widehat{Q}(S,a;w)/\iota]}{\sum_{n=1}^N \exp[-\widehat{Q}(S,n;w)/\iota]}. \tag{14}$$

At routing epoch k, define the action feature $\varphi_k(a) = \Phi(\mathbf{W}_k^{(a)}, \chi_k^{(a)})$ and immediate workload cost $c_k = B_k$. After selecting $A_{k+1} \sim \pi_{w_k}(\cdot \mid S_{k+1})$, the temporal difference error δ_k and updated weight vector w_{k+1} are

$$\begin{aligned} \delta_k &= c_k + \gamma w_k^\top \varphi_{k+1}(A_{k+1}) - w_k^\top \varphi_k(A_k), \\ w_{k+1} &= \Pi_{\mathcal{W}}(w_k + \alpha_k \delta_k \varphi_k(A_k)), \qquad \alpha_k = \frac{\alpha_0}{(k+1)^\nu}. \end{aligned} \tag{15}$$

Here the discount factor $\gamma \in (0,1)$ and learning rate α_k control the update, and the projection operator $\Pi_{\mathcal{W}}$ is Euclidean projection onto $\mathcal{W}$. Both value estimates in δ_k use the preupdate vector w_k.

3.3 Training and Deployment

CARRY learns w in the nominal simulator. Evaluation transfers the learned parameters to the practical simulator and deploys the learned policy without additional updates. Algorithm 1 summarizes the learning procedure.

Algorithm 1 *CARRY* policy learning

Require: Initial weights $w_0 \in \mathcal{W}$, updates K_{train}, discount γ, temperature ι, learning rates $\{\alpha_k\}$
1: Observe S_0 and construct $\varphi_0(a)$ for every $a \in \mathcal{N}$
2: Draw $A_0 \sim \pi_{w_0}(\cdot \mid S_0)$
3: **for** $k = 0, \ldots, K_{\text{train}} - 1$ **do**
4: Route request J_k to instance A_k and observe c_k and S_{k+1}
5: Construct $\varphi_{k+1}(a)$ for every $a \in \mathcal{N}$
6: Draw $A_{k+1} \sim \pi_{w_k}(\cdot \mid S_{k+1})$
7: Compute the TD error δ_k in (15)
8: Update w_{k+1} by (15)
9: **end for**
10: **return** Learned weights $w_{K_{\text{train}}}$

4 Theoretical Analysis

System serving stability, measured by unfinished request workload, captures whether routing prevents requests from accumulating, while serving capacity gives the sustainable request rates. *CARRIER* makes these system level properties tractable through a common representation of reuse and load.

4.1 Assumptions

Assumption 1 (Bounded request sizes)**.** *There is an $s_+ < \infty$ such that $p(J) + d(J) \le s_+$ almost surely.*

Assumption 2 (*CARRY* parameter domain)**.** *The set $\mathcal{W}$ in* (13) *is nonempty and compact, $0 < \epsilon_w \le w_{\max} < \infty$, and all feature scales and ι are positive and finite.*

Assumption 3 (*CARRY* learning conditions)**.** *The learning rates satisfy the Robbins–Monro conditions (Robbins & Monro, 1951), $\sum_k \alpha_k = \infty$ and $\sum_k \alpha_k^2 < \infty$, equivalently $\alpha_0 > 0$ and $1/2 < \nu \le 1$ for* (15)*. The update direction $g_k := \delta_k \varphi_k(A_k)$ satisfies $\sup_k \mathbb{E}\|g_k\|_2^2 < \infty$ along the training trajectory.*

4.2 Main Results

Theorem 1 (System Serving Stability and Capacity under Routing)**.** *Under Assumption 1, consider serving systems operated under routing policies π and π' for which the limits in* (5)–(7) *exist. Their serving stability margins ε_π^L and $\varepsilon_{\pi'}^L$ satisfy*

$$\varepsilon_\pi^L = 1 - \frac{\lambda}{N\mu}\left(\bar{\ell} - \frac{\bar{s}}{\upsilon}\mathcal{Y}_\pi\right) - \mathcal{I}_\pi^L, \tag{16}$$

$$\varepsilon_\pi^L - \varepsilon_{\pi'}^L = \frac{\lambda \bar{s}}{\upsilon N \mu}(\mathcal{Y}_\pi - \mathcal{Y}_{\pi'}) + \mathcal{I}_{\pi'}^L - \mathcal{I}_\pi^L. \tag{17}$$

Moreover, $\lambda \in \mathcal{R}_\pi$ implies $\varepsilon_\pi^L \geq 0$, meaning that at this request rate the serving system under π is stable as measured by unfinished request workload and has a nonnegative serving stability margin. Equivalently, $\varepsilon_\pi^L < 0$ implies $\lambda \notin \mathcal{R}_\pi$.

Theorem 1 gives a necessary outer boundary for system serving capacity.

Theorem 2 (Serving Capacity Inner Bound under *CARRY*)**.** *Under Assumptions 1–2, fix $w \in \mathcal{W}$ and let* C*,* CG*, and* LB *denote* CARRY*,* Cache Greedy*, and* Load Balance*. For the normalized serving potential cost $\Gamma_\pi(w)$ in Appendix A.3.2, define the serving capacity inner bound $\widehat{\mathcal{R}}_\pi(w) = \{\lambda \geq 0 : \lambda\Gamma_\pi(w) < 1\}$. For each $\pi \in \{\mathrm{C}, \mathrm{CG}, \mathrm{LB}\}$, the serving system under π is stable at every request rate in $\widehat{\mathcal{R}}_\pi(w)$, and*

$$\begin{aligned} \Gamma_{\mathrm{C}}(w) &\leq \min\{\Gamma_{\mathrm{CG}}(w), \Gamma_{\mathrm{LB}}(w)\}, \\ \widehat{\mathcal{R}}_{\mathrm{CG}}(w) \cup \widehat{\mathcal{R}}_{\mathrm{LB}}(w) \subseteq \widehat{\mathcal{R}}_{\mathrm{C}}(w), &\qquad \widehat{\mathcal{R}}_\pi(w) \subseteq \mathcal{R}_\pi. \end{aligned} \tag{18}$$

The first inclusion is strict relative to baseline $\pi_0 \in \{\mathrm{CG}, \mathrm{LB}\}$ whenever $\Gamma_{\mathrm{C}}(w) < \Gamma_{\pi_0}(w)$. When the long run limits in Theorem 1 exist, $\lambda \in \widehat{\mathcal{R}}_{\mathrm{C}}(w)$ and $\varepsilon_{\pi_0}^L < 0$ imply $\lambda \in \mathcal{R}_{\mathrm{C}} \setminus \mathcal{R}_{\pi_0}$.

Theorem 2 gives sufficient inner regions and compares them across routing methods. Its inner bound under *CARRY* contains those under both baselines. Combined with Theorem 1, the final implication further establishes an actual serving capacity separation whenever the *CARRY* inner condition holds and a baseline has negative serving stability margin: the system is stable under *CARRY* but cannot be stable under that baseline. Thus, the two results establish an unconditional certified inner bound advantage and, under the stated condition, an actual serving capacity advantage at that request rate.

Theorem 3 (*CARRY* Robustness to Cache and Decode Errors)**.** *Under Assumptions 1–2, fix $w \in \mathcal{W}$ and define the weighted service term $D_w(\mathbf{W}) = (\mu/\sigma_{W^2}) \sum_n w_{n,2} W_n$ and error span $\mathrm{span}_a(e) = \max_a e_a - \min_a e_a$. Consider each error separately.*

(i) Cache evaluation error. *Let q_a be* CARRY*'s action increment under* CARRIER *and $\widetilde{q}_a$ the increment used for routing. With $e_a^{\mathrm{c}} = \widetilde{q}_a - q_a$, suppose $\mathrm{span}_a(e^{\mathrm{c}}) \leq E_{\mathrm{c}} + \kappa_{\mathrm{c}} D_w(\mathbf{W})$ for finite E_{c} and $\kappa_{\mathrm{c}} \geq 0$.*

(ii) Decode size error. *Suppose* CARRY *evaluates each instance using one request level estimate $\widehat{d} = d + e_d$, where $\widehat{d} \geq 0$ and $|e_d| \leq \delta_d$, while the realized* CARRIER *dynamics use d.*

Let $\eta_{\mathrm{c}} = \kappa_{\mathrm{c}}/2$ and $\eta_{\mathrm{d}} = \delta_d/\mu$. For $e \in \{\mathrm{c}, \mathrm{d}\}$, the serving system with error e is stable at every request rate in the certified serving capacity inner region

$$\widehat{\mathcal{R}}_{\mathrm{C}}^{e}(w) = \{\lambda \geq 0 : \lambda[\Gamma_{\mathrm{C}}(w) + \eta_e] < 1\} \subseteq \widehat{\mathcal{R}}_{\mathrm{C}}(w). \tag{19}$$

For $\Gamma_{\mathrm{C}}(w) > 0$, its request rate boundary contracts from $1/\Gamma_{\mathrm{C}}(w)$ to $1/[\Gamma_{\mathrm{C}}(w) + \eta_e]$, a relative contraction of $\eta_e/[\Gamma_{\mathrm{C}}(w) + \eta_e]$.

Theorem 4 (*CARRY* Update Stabilization)**.** *Under Assumptions 2 and 3, if $w_0 \in \mathcal{W}$, then* CARRY*'s successive updates satisfy $\sum_k \|w_{k+1} - w_k\|_2^2 < \infty$ almost surely and hence $\|w_{k+1} - w_k\|_2 \to 0$ almost surely; projection keeps every $w_k \in \mathcal{W}$.*

5 Evaluation

We evaluate the modeling consistency of *CARRIER* by comparing its request arrival and serving system models with commonly used alternatives. The aligned trends show that *CARRIER* reasonably abstracts both request arrivals and distributed LLM serving. Furthermore, we compare *CARRY* with baselines across application scenarios, extreme cases, request rates, system scales, and LLM scales, where results on cache hit rate, TTFT, latency, and load balance show consistent performance advantages. Experiments on *CARRY*'s stability, convergence, and robustness validate the corresponding theoretical results. Ablations over features and training modes further demonstrate the effectiveness and necessity of *CARRY*'s components and training stages.

5.1 Experimental Setup

Testbed and model. The nominal simulator is derived entirely from *CARRIER* and directly implements its request arrival and serving system dynamics. To capture practical serving behavior at higher

fidelity, the practical simulator extends Vidur with KV cache retention, reuse, and eviction (Agrawal et al., 2024a). We serve Llama-3-8B and Llama-3-70B (Grattafiori et al., 2024) on 8 to 50 homogeneous A100 instances. *CARRY* is trained in the nominal simulator and transferred to the practical simulator without additional updates. All results average 3 or 4 random seeds. Appendix A.4 gives implementation details.

Datasets and workloads. Three real datasets cover conversation (ShareGPT (sha, 2023)), tool and agent (OpenHands (Wang et al., 2025b)), and long context (LooGLE (Li et al., 2024)). Synthetic datasets cover four extreme cases. Per instance cache capacity ranges from $5{,}000$ to $20{,}000$ blocks, while request rates range from 1 to 40 requests/s. These settings are adjusted to dataset size and system scale. Appendix A.4 gives the construction and parameter ranges.

Baselines and metrics. Baselines are *Cache Greedy*, *Load Balance*, and *Min TTFT* (Qin et al., 2025). We also compare with *DualMap*, the state of the art method adapted to affinity groups (Yuan et al., 2026). All methods use the same requests, cache capacity, request rates, and seeds. Metrics include TTFT, end to end latency, throughput, cache hit rate, and normalized service load imbalance (NSLI), corresponding to $\mathcal{I}_\pi^L$ in Equation (7).

5.2 Modeling Consistency

We evaluate the consistency of *CARRIER*'s request arrival and serving system models, with the results shown in Figure 2. For request arrival consistency, we compare the cache behavior induced by cache affinity groups with vLLM's token level cache management (Kwon et al., 2023). P50 and P90 TTFT differ by at most 0.6% across all algorithms, indicating that cache affinity groups reasonably abstract requests with reusable context. For serving system consistency, we compare CARRIER's proportional eviction rule (Appendix A.2) with least recently used (LRU). Their TTFT differs by at most 3.6%, showing that proportional eviction preserves the behavior of LRU while remaining amenable to theoretical analysis.

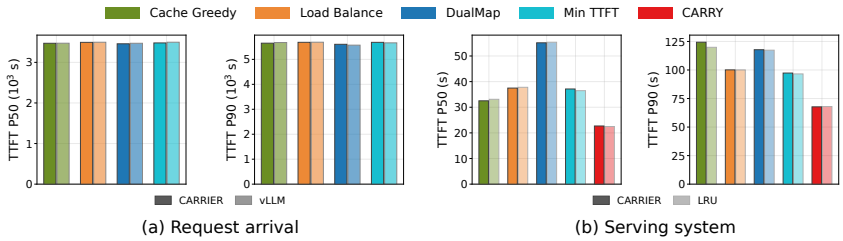

Figure 2: Performance comparison of *CARRIER* with alternative models.

5.3 Algorithm Performance

We compare *CARRY* with four baselines across three application scenarios, eight request rates, and four seeds in Figure 3, observing the largest gains under contention. Compared to *DualMap*, *CARRY* improves cache hit rate by up to 75.2% and reduces P50 TTFT and latency by 82.1% and 77.8%. Table 1 further shows the highest average cache hit rate and lowest P90–P99 TTFT and latency on all datasets, with P95 reductions of 5.5%–29.7% and 5.5%–29.6%, respectively. *CARRY* also achieves the lowest P50 latency on ShareGPT and LooGLE.

Table 1: Average performance. Hit is cache hit rate (%); TTFT and latency list P50/P95 (s).

	ShareGPT			**LooGLE**			**OpenHands**		
Method	Hit	TTFT	Latency	Hit	TTFT	Latency	Hit	TTFT	Latency
Load Balance	10.8%	351.6/582.7	354.5/586.3	3.2%	1521.6/2654.3	1521.6/2654.3	10.0%	997.5/3252.8	1001.1/3256.9
Cache Greedy	51.3%	344.9/616.2	348.0/619.5	16.6%	1235.0/2199.6	1235.1/2199.7	66.0%	**413.3**/2314.9	**417.3**/2318.2
Min TTFT	32.3%	350.5/586.2	353.4/589.5	5.9%	1487.6/2594.1	1487.7/2594.2	20.9%	904.8/3085.2	908.2/3089.3
DualMap	48.7%	350.1/586.6	353.1/589.7	13.6%	1305.5/2233.7	1305.6/2233.8	43.8%	696.3/2376.3	699.9/2380.6
CARRY	**60.0%**	**330.4/549.8**	**333.4/552.8**	**22.3%**	**1071.2/1794.4**	**1071.3/1794.4**	**66.7%**	452.0/**1627.4**	455.7/**1631.6**

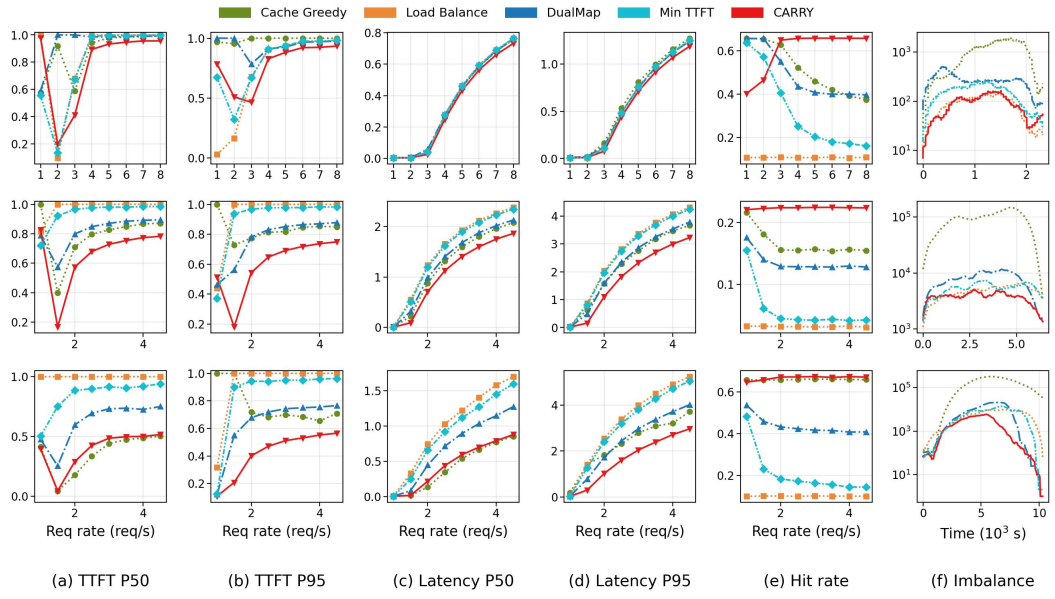

Figure 3: Performance comparison between *CARRY* and all baselines. From top to bottom, the rows are ShareGPT, Loogle, and OpenHands. (a) and (b) show the TTFT P50 and P95, each normalized by the largest value among the methods at the same request rate. (c) and (d) show the end-to-end latency P50 and P95 divided by 1000 in seconds.

On the 50-replica OpenHands grid, *CARRY* has the lowest P95 TTFT at every arrival rate, with a max-normalized value of 0.26–0.59, 20%–50% below the next-best method. Its throughput rises to 9.3 req/s, 17% above DualMap, the second best algorithm.

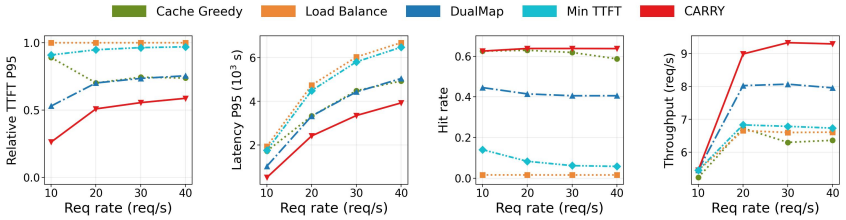

Figure 4: Comparison between *CARRY* and all baselines in a large scale simulation.

6 Conclusion

This paper addresses cache reuse and load balance under evolving cache states in distributed LLM serving. We introduced *CARRIER*, a mathematical model that captures group structured request arrivals together with cache reuse, retention, and eviction across instances, making cache evolution through waiting queues tractable. Building on *CARRIER*, *CARRY* learns a serving potential that balances immediate reuse, instance load, and future cache usefulness. Our analysis characterizes system serving stability, quantifies *CARRY*'s serving capacity advantage over baselines, establishes robustness to bounded cache state and decode size errors, and shows that successive updates vanish almost surely. Experiments with our nominal and practical simulators validate *CARRIER*'s modeling consistency, *CARRY*'s performance, and the theoretical results. Across three benchmarks, *CARRY* improves cache hit rate by up to 75.2% and reduces TTFT and latency by up to 82.1% and 77.8%. Extending the current homogeneous FCFS setting to heterogeneous clusters and broader scheduling policies remains future work.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the manuscript. We have not used generative AI tools for retrieval or discovery, research ideation or execution. or draft sections, and generating synthetic datasets or proving mathematical claims are not applicable to this work. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Sharegpt dataset., 2023. URL https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered.

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems*, 6:351–366, 2024a.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024b.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Vivek S Borkar and Sean P Meyn. The ode method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, 2023.

Douglas Down, Sean P Meyn, and Richard L Tweedie. Exponential and uniform ergodicity of markov processes. *The Annals of Probability*, 23(4):1671–1691, 1995.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27, 2025.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, 2024.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024a.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024b.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, et al. Lmcache: An efficient kv cache layer for enterprise-scale llm inference. *arXiv preprint arXiv:2510.09665*, 2025.

Sean P Meyn and Richard L Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE transactions on parallel and distributed systems*, 12(10):1094–1104, 2002.

Michael Neely. *Stochastic network optimization with application to communication and queueing systems*. Morgan & Claypool Publishers, 2010.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 1133–1150, 2025.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot. In *23rd USENIX conference on file and storage technologies (FAST 25)*, pp. 155–170, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. In *International conference on learning representations*, volume 2025, pp. 37057–37082, 2025.

Foteini Strati, Sara Mcallister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. D\'ej\avu: Kv-cache streaming for fast, fault-tolerant generative llm serving. *arXiv preprint arXiv:2403.01876*, 2024.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 173–191, 2024.

Richard S Sutton, Andrew G Barto, et al. Reinforcement learning: An introduction. mit press cambridge. *Massachusetts London, England*, 2018.

Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *29th IEEE Conference on Decision and Control*, pp. 2130–2132. IEEE, 1990.

John Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, 9, 1996.

Martin J Wainwright, Michael Irwin Jordan, et al. Graphical models, exponential families, and variational inference. 2008.

Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. {KVCache} cache in the wild: Characterizing and optimizing {KVCache} cache at a large cloud provider. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 465–482, 2025a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025b.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5831–5841, 2025c.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. {FastServe}:{Iteration-Level} preemptive scheduling for large language model inference. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 57–74, 2026a.

Fangzhou Wu, Sandeep Silwal, and Qiuyi Richard Zhang. Randomization boosts kv caching, learning balances query load: A joint perspective. In *International Conference on Learning Representations*, volume 2026, pp. 85754–85780, 2026b.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the twentieth European conference on computer systems*, pp. 94–109, 2025.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11608–11620, 2024.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Ying Yuan, Pengfei Zuo, Bo Wang, Zhangyu Chen, Zhipeng Tan, and Zhou Yu. Dualmap: Enabling both cache affinity and load balancing for distributed llm serving. *arXiv preprint arXiv:2602.06502*, 2026.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

A APPENDIX

A.1 NOTATION

Symbol	Meaning
CARRIER *model*	
$\mathcal{M} = \{1, \dots, M\}, m, m'; \mathcal{N} = \{1, \dots, N\}, n, a$	Cache affinity groups and serving instances; their indices
k, ℓ, i, r, j	Arrival epoch, within group request index, queue position, observed sample index, feature coordinate
$J_{m,\ell}, J_k, J^{(r)}, J^{\mathrm{srv}}, \mathbf{J}$	Group request, arriving request, observed request, serving request, ordered waiting requests
$J = (m, p, d, b, q)$	Group, prefill and decode blocks, reusable prefix bound, retention target
t, t_k, λ	Continuous serving time, request arrival time, and Poisson request rate
$\mathcal{J}_R, R, \widehat{F}_R, \delta_J$	Empirical request set, its size, empirical request distribution, point mass
ω_m, F_m, R_m	Affinity group frequency, reusable prefix distribution, associated random variable
$\mathcal{Q}_n, \mathcal{B}_n, U_n = (u_{n,m}), \xi_n, C$	Waiting queue, serving batch, cache state, eviction policy state, cache capacity
$E((U, \xi), J)$	Cache transition under retention and eviction rule E
$\mu, \upsilon, \mu^{\mathrm{p}}, \mu^{\mathrm{d}}$	Decode rate, prefill speed ratio, prefill rate $\mu^{\mathrm{p}} = \upsilon\mu$, decode rate $\mu^{\mathrm{d}} = \mu$
$h_n, \widehat{h}_{n,J}, G_{n,J}, L_{n,J}$	Immediate and queue aware cache hit sizes, effective workload, serving workload
$\widetilde{p}_n, \widetilde{d}_n, R_n^G, R_n^L$	Residual prefill and decode blocks, effective and serving workload residuals
$V_n = (v_{n,m}), \chi_k$	Queue aware cache state, its entries, and complete queue aware cache view
$W_n, \mathbf{W}$	Instance workload and system workload
$B(t), B^{\pi}(t), B(X), B_k, B_a$	Raw workload of unfinished requests, its policy indexed forms, per step cost, instance value
Z_n, Z	Auxiliary instance and total serving workloads
$A_k, A', \pi, \pi', \pi^{\star}, h_k^{\pi}$	Selected and next actions, routing, comparison, and optimal policies, realized cache hit
$\bar{s}, \bar{\ell}$	Mean request size and mean serving workload without reuse
$\rho_{\pi,n}^L, \mathcal{Y}_{\pi}, \mathcal{I}_{\pi}^L, \varepsilon_{\pi}^L$	Normalized per instance offered service load, reuse yield, normalized service load imbalance, serving stability margin
$\mathcal{R}_{\pi}, \theta$	Serving capacity region under π and exponential moment parameter
$X, X', X_t, X_k, X_0, S, S_k, \gamma$	Complete and next states, continuous and routing epoch states, initial state, routing observations, discount factor
$Q^{\pi}(X, a)$	Policy action cost satisfying the Bellman expectation equation
$(\mathbf{W}^{(a)}, \chi^{(a)})$	State after assigning the arriving request to instance a
$\overline{u}_{n,m}, g_{n,m}, e_n, e_{n,m'}$	Target cache allocation, cache growth, total and per group eviction
$U^{(i)}, \xi^{(i)}, h_i, G_i, K$	Cache propagation state, eviction state, cache hit, effective workload, and queue length
CARRY *algorithm*	
$H(V_n)$	Future cache usefulness
$\sigma_W, \sigma_{W^2}, \sigma_H, \sigma_{H^2}$	Feature normalization scales
$\phi_n, \Phi, \varphi_k(a)$	Instance, system, and action feature vectors
$w = (w_{n,j}), \mathcal{W}, w_{\max}, \epsilon_w$	Weight vector, parameter domain, magnitude limit, quadratic weight floor
$\Psi_w, \widehat{Q}(S, a; w), G_a$	Serving potential, approximated action cost, action effective workload
π_w, ι	*CARRY* softmax policy and temperature
c_k, δ_k	Immediate workload cost and temporal difference error
$\alpha_k, \alpha_0, \nu, \Pi_{\mathcal{W}}$	Learning rate, initial scale, decay exponent, Euclidean projection
K_{train}, g_k	Number of learning updates and stochastic update direction
Theory and proofs	
s_+	Almost sure request size bound
$\mathrm{C}, \mathrm{CG}, \mathrm{LB}, \pi_0$	*CARRY*, Cache Greedy, Load Balance, and a baseline policy
$\widehat{\mathcal{R}}_{\pi}(w), \Gamma_{\pi}(w)$	Serving capacity inner bound under π and normalized potential cost
$\widehat{\mathcal{R}}_{\mathrm{C}}^{e}(w), \eta_{\mathrm{c}}, \eta_{\mathrm{d}}$	Serving capacity inner region under error e and its cache and decode contraction terms
$D_w(\mathbf{W}), D_w^{\mathrm{svc}}(X)$	Workload service term and phase dependent service term
$q_a, \widetilde{q}_a$	*CARRY* serving potential increment and increment used by routing
$e_a^{\mathrm{c}}, E_{\mathrm{c}}, \kappa_{\mathrm{c}}$	Cache evaluation error, additive bound, workload dependent coefficient
$\widehat{d}, e_d, \delta_d, e_a^{\mathrm{d}}$	Decode estimate, decode error, error bound, induced action cost error

Symbol	Meaning
$\alpha_n, \beta_n, \eta_n, \zeta_n$	Coefficients of the four serving potential features
$\underline{\beta}, \overline{\beta}, \overline{\alpha}, \overline{\beta}_w$	Coefficient bounds used in the proofs
$\overline{H}_a, \Delta H_a, A, A_\pi, A_{\mathrm{C}}, A_{\mathrm{CG}}, A_{\mathrm{LB}}$	Cache usefulness, its action change, and random actions under the policies
$\ell_a(X, J), r_n(X), \mathcal{X}_+, c_\pi(X)$	Leading workload term, phase rate, positive workload prearrival states, expected leading request term
$\mathcal{L}_\pi, \mathcal{S}, \varpi$	Extended generator, serving operator, reference action distribution
$V_\theta^B, \widetilde{V}_\theta, \mathcal{C}$	Unfinished request workload and auxiliary Lyapunov functions, bounded workload set
ϱ_a, R_+	Bounded remainder of the action increment and its upper bound
$C_{\mathrm{ph}}, C_{w,\pi}$	Phase and policy dependent drift constants
R_w, C_*, c_1, c_2	Radial serving potential, shift, and workload comparison constants
J_R, C_R, C_e, c, b, R_0	Jump, Taylor, and exponential drift constants
$A^B(s,t]$	Raw work arriving during $(s,t]$
$\widetilde{\pi}_w, \mathcal{L}_{w,\mathrm{c}}, \widehat{G}_a, C_d$	Policy and generator under cache error, estimated effective workload, bound on the decode error remainder
Mathematical operators	
$\mathbb{E}, \Pr, \mathbf{1}\{\cdot\}, [x]^+$	Expectation, probability, indicator, positive part
$\mathrm{Exp}(\lambda), \overset{\text{i.i.d.}}{\sim}, \perp$	Exponential distribution, independent identical sampling, independence
$\|\cdot\|_1, \|\cdot\|_2, \mathrm{span}_a, \mathrm{col}$	Norms, action span, vector concatenation
Unif arg min, Unif arg max	Uniform tie breaking among optimizing actions

A.2 Modeling Specification

Markov property. Under *CARRIER*'s stochastic arrival model, the exponential interarrival time is memoryless and the next request profile is sampled independently. Given (X_k, A_k), FCFS scheduling, the phase rates, and cache rule E determine the distribution of every queue, serving batch, and cache transition before the next request. Therefore, for every state event $\mathcal{A}$,

$$\Pr(X_{k+1} \in \mathcal{A} \mid X_0, A_0, \ldots, X_k, A_k) = \Pr(X_{k+1} \in \mathcal{A} \mid X_k, A_k). \tag{20}$$

The complete state X_k is Markov, whereas feature observation S_k may omit information needed to determine the next state.

Empirical Sampling and Application Driven Arrivals. For empirical request set $\mathcal{J}_R$, the affinity group frequency ω_m and conditional reusable prefix distribution F_m are

$$\begin{aligned} \omega_m &= \frac{1}{R}\sum_{r=1}^{R} \mathbf{1}\{m(J^{(r)}) = m\}, \\ F_m(z) := \Pr(R_m = z) &= \frac{\sum_{r=1}^{R} \mathbf{1}\{m(J^{(r)}) = m, b(J^{(r)}) = z\}}{\sum_{r=1}^{R} \mathbf{1}\{m(J^{(r)}) = m\}}, \qquad \omega_m > 0. \end{aligned} \tag{21}$$

Full request profile sampling preserves attribute dependence. Workloads derived from application applications also preserve the recorded order within each group.

Integer Proportional Eviction. The cache state satisfies $u_{n,m} \in \mathbb{Z}_{\geq 0}$ and $\sum_m u_{n,m} \leq C$. After completing $J = (m, p, d, b, q)$, the target allocation $\bar{u}_{n,m}$, cache growth $g_{n,m}$, and eviction demand e_n are

$$\bar{u}_{n,m} = \max\{u_{n,m}, \min(q, C)\}, \quad g_{n,m} = \bar{u}_{n,m} - u_{n,m}, \quad e_n = \left[g_{n,m} - \left(C - \sum_{m'} u_{n,m'} \right) \right]^+. \tag{22}$$

CARRIER updates

$$u^+_{n,m'} = \begin{cases} \bar{u}_{n,m}, & m' = m, \\ u_{n,m'} - e_{n,m'}, & m' \neq m, \end{cases} \quad \sum_{m' \neq m} e_{n,m'} = e_n, \quad 0 \leq e_{n,m'} \leq u_{n,m'}. \tag{23}$$

Competing entries are evicted proportionally to occupancy; integer remainders are assigned uniformly without replacement. Cache state propagation preserves this rule.

Cache State Propagation under FCFS. For serving request J^{srv} on instance n, let $R_n^G = \widetilde{p}_n + \widetilde{d}_n$ and initialize the propagated cache state at completion as $(U^{(0)}, \xi^{(0)}) = E((U_n, \xi_n), J^{\mathrm{srv}})$. If the serving batch is empty, use $(U^{(0)}, \xi^{(0)}) = (U_n, \xi_n)$. For the ordered waiting queue $\mathbf{J} = (J_1, \ldots, J_K)$, recursively set

$$\begin{aligned} h_i &= \min\{b_i, u_{m_i}^{(i-1)}\}, \qquad G_i = p_i + d_i - h_i, \\ (U^{(i)}, \xi^{(i)}) &= E((U^{(i-1)}, \xi^{(i-1)}), J_i). \end{aligned} \tag{24}$$

The queue aware cache view is $V_n = U^{(K)}$ and $W_n = R_n^G + \sum_i G_i$. Under FCFS and shared eviction randomness, this recursion gives the cache states and cache hits encountered by waiting requests.

A.3 Theoretical Proofs

For the proofs, let $R_n^L = \widetilde{d}_n + \widetilde{p}_n/\upsilon$ and define $Z_n = R_n^L + \sum_{J \in \mathcal{Q}_n} L_{n,J}$ and $Z = \sum_n Z_n$. These auxiliary serving workloads satisfy $Z \le \|\mathbf{W}\|_1 \le \upsilon Z$.

A.3.1 Proof of System Serving Stability and Capacity under Routing

Proof of Theorem 1. FCFS cache state propagation uses the same eviction randomness as serving, so $\widehat{h}_{A_k,J_k} = h_k^\pi$ and hence $L_{A_k,J_k} = d_k + (p_k - h_k^\pi)/\upsilon$ for every routed request. Summing to time T, taking expectations, and using the compound Poisson means $\mathbb{E}[\sum_{k:t_k<T}(d_k + p_k/\upsilon)] = \lambda T \bar{\ell}$ and the definition of $\mathcal{Y}_\pi$ gives

$$\sum_n \rho_{\pi,n}^L = \frac{\lambda}{\mu}\left(\bar{\ell} - \frac{\bar{s}}{\upsilon}\mathcal{Y}_\pi\right).$$

Since $\max_n \rho_{\pi,n}^L = N^{-1}\sum_n \rho_{\pi,n}^L + \mathcal{I}_\pi^L$, the two identities follow. System serving stability implies $\sup_t \mathbb{E}B(t) < \infty$ because $x \le e^{\theta x}/\theta$ for $x \ge 0$. Moreover, $Z_n(t) \le B(t)$ and workload conservation with service rate μ give $\rho_{\pi,n}^L \le 1$; hence $\varepsilon_\pi^L \ge 0$ (Tassiulas & Ephremides, 1990). □

A.3.2 Proof of Serving Capacity Inner Bound under *CARRY*

Serving potential drift. Let the four serving potential coefficients be

$$\alpha_n = \frac{w_{n,1}}{\sigma_W}, \quad \beta_n = \frac{w_{n,2}}{\sigma_{W^2}} > 0, \quad \eta_n = \frac{w_{n,3}}{\sigma_H}, \quad \zeta_n = \frac{w_{n,4}}{\sigma_{H^2}}. \tag{25}$$

Thus $\underline{\beta} := \epsilon_w/\sigma_{W^2} \le \beta_n \le \overline{\beta} := w_{\max}/\sigma_{W^2}$. With $H_a := H(V_a)$ and $\Delta H_a := H(V_a^{(a)}) - H(V_a)$, let $q_a := \Psi_w(\mathbf{W}^{(a)}, \chi^{(a)}) - \Psi_w(\mathbf{W}, \chi)$ denote the serving potential increment caused by assigning the arriving request J to instance a. Then

$$\Psi_w(\mathbf{W}, \chi) = \sum_{n=1}^N \left[\alpha_n W_n + \beta_n W_n^2 + \eta_n H(V_n) + \zeta_n H(V_n)^2\right], \tag{26}$$

$$q_a = \alpha_a G_a + \beta_a(2W_a G_a + G_a^2) + \eta_a \Delta H_a + \zeta_a(2H_a \Delta H_a + \Delta H_a^2). \tag{27}$$

At prearrival state X, the baselines draw

$$A_{\mathrm{CG}} \sim \mathrm{Unifarg}\max_a \min\{b(J), u_{a,m(J)}\}, \qquad A_{\mathrm{LB}} \sim \mathrm{Unifarg}\min_a B_a. \tag{28}$$

where $B_a = \sum_{J' \in \mathcal{Q}_a \cup \mathcal{B}_a}[p(J') + d(J')]$ is the raw workload of unfinished requests at instance a. Here $G_a(X, J)$ is evaluated at the scheduled serving turn as in (3). For $w \in \mathcal{W}$, define

$$\ell_a(X, J) = \beta_a W_a G_a(X, J), \qquad D_w^{\mathrm{svc}}(X) = \sum_{n: W_n > 0} r_n(X)\beta_n W_n, \tag{29}$$

where $r_n(X) = \upsilon\mu$ during prefill and $r_n(X) = \mu$ during decode. Let $\mathcal{X}_+$ be the states with positive system workload in the closure of the attainable prearrival state set. For $\pi_0 \in \{\mathrm{CG}, \mathrm{LB}\}$, define

$$\begin{aligned} &c_{\mathrm{C}}(X) = \mathbb{E}_J \min_a \ell_a(X, J), \quad c_{\pi_0}(X) = \mathbb{E}_{J, A_{\pi_0}} \ell_{A_{\pi_0}}(X, J), \\ &\Gamma_\pi(w) = \sup_{X \in \mathcal{X}_+} \frac{c_\pi(X)}{D_w^{\mathrm{svc}}(X)}. \end{aligned} \tag{30}$$

Set $\Gamma_\pi(w) = 0$ if $\mathcal{X}_+ = \varnothing$; this case is trivial. The extended generator drift $\mathcal{L}_\pi \Psi_w(X)$ is

$$\mathcal{L}_\pi \Psi_w(X) = \lim_{u \downarrow 0} \frac{1}{u} \mathbb{E}_\pi[\Psi_w(X_{t+u}) - \Psi_w(X_t) \mid X_t = X]. \tag{31}$$

Only coordinate a changes, so $\widehat{Q}(S, a; w) = \Psi_w(\mathbf{W}, \chi) + q_a$. For any action distribution ϖ, the Gibbs identity (Wainwright et al., 2008) gives

$$\mathbb{E}_{\pi_w} q_A \le \mathbb{E}_\varpi q_A + \iota \log N, \tag{32}$$

Between arrivals the queue aware cache view is unchanged, and $R_n^L \le R_n^G \le \upsilon R_n^L$. During prefill, $(\dot{R}_n^G, \dot{R}_n^L) = (-\upsilon\mu, -\mu)$; during decode, both derivatives equal $-\mu$. Thus $r_n(X) \in \{\mu, \upsilon\mu\}$ and

$$\mathcal{S}\Psi_w(X) = -\sum_{n: W_n > 0} r_n(X)(\alpha_n + 2\beta_n W_n). \tag{33}$$

For some finite C_{ph},

$$\mathcal{S}\Psi_w(X) \le C_{\mathrm{ph}} - 2D_w^{\mathrm{svc}}(X). \tag{34}$$

Adding the compound Poisson arrival term gives

$$\mathcal{L}_\pi \Psi_w(X) = \mathcal{S}\Psi_w(X) + \lambda \mathbb{E}_{J, A_\pi}[q_{A_\pi}]. \tag{35}$$

Taking ϖ at a minimizing action gives

$$\mathbb{E}_{\pi_w} q_A \le \min_a q_a + \iota \log N.$$

Serving capacity inner region ordering. Decompose the action potential increment q_a as

$$q_a = 2\ell_a + \varrho_a(X, J), \qquad \varrho_a = \alpha_a G_a + \beta_a G_a^2 + \eta_a \Delta H_a + \zeta_a(2H_a \Delta H_a + \Delta H_a^2). \tag{36}$$

Bounded requests, cache states, and weights give finite $R_+ := \max\{0, \sup_{w \in \mathcal{W}, X, J, a} \varrho_a(X, J)\}$. The Gibbs bound with a point mass on a minimizing action gives

$$\mathbb{E}_{J, A_{\mathrm{C}}} q_{A_{\mathrm{C}}} \le \mathbb{E}_J \min_a q_a + \iota \log N \le 2c_{\mathrm{C}}(X) + R_+ + \iota \log N. \tag{37}$$

For either baseline π_0,

$$\mathbb{E}_{J, A_{\pi_0}} q_{A_{\pi_0}} \le 2c_{\pi_0}(X) + R_+. \tag{38}$$

Since $\min_a \ell_a \le \ell_{A_{\pi_0}}$,

$$\Gamma_{\mathrm{C}} \le \min\{\Gamma_{\mathrm{CG}}, \Gamma_{\mathrm{LB}}\}. \tag{39}$$

Also, $0 \le G_a \le s_+$ and $D_w^{\mathrm{svc}} \ge D_w(\mathbf{W})$, so $0 \le \Gamma_\pi(w) \le s_+/\mu < \infty$ for all three rules. Since $c_\pi(X) \le \Gamma_\pi(w) D_w^{\mathrm{svc}}(X)$, combining (37) or (38) with (34)–(35) therefore yields, for $\pi \in \{\mathrm{C}, \mathrm{CG}, \mathrm{LB}\}$,

$$\mathcal{L}_\pi \Psi_w(X) \le C_{w,\pi} - 2[1 - \lambda\Gamma_\pi(w)] D_w^{\mathrm{svc}}(X), \tag{40}$$

with

$$C_{w,\mathrm{C}} = C_{\mathrm{ph}} + \lambda(R_+ + \iota \log N), \qquad C_{w,\pi_0} = C_{\mathrm{ph}} + \lambda R_+. \tag{41}$$

The same intercept covers $D_w^{\mathrm{svc}} = 0$. If $\lambda\Gamma_\pi < 1$, the drift is negative outside a bounded workload set.

Workload moment bound. For any $\pi \in \{\mathrm{C}, \mathrm{CG}, \mathrm{LB}\}$ satisfying $\lambda\Gamma_\pi(w) < 1$, choose C_* so that the radial serving potential $R_w(\mathbf{W}, \chi)$ satisfies

$$R_w(\mathbf{W}, \chi) := \sqrt{C_* + \Psi_w(\mathbf{W}, \chi)} \ge 1. \tag{42}$$

Positive workload curvature and bounded cache terms give $0 < c_1 < c_2 < \infty$ such that

$$c_1(1 + \|\mathbf{W}\|_2) \le R_w(\mathbf{W}, \chi) \le c_2(1 + \|\mathbf{W}\|_2). \tag{43}$$

The tangent inequality for the square root and (40) give

$$\mathcal{L}_\pi R_w(X) \le \frac{C_{w,\pi} - 2[1 - \lambda\Gamma_\pi(w)] D_w^{\mathrm{svc}}(X)}{2R_w(\mathbf{W}, \chi)}.$$

By (43), $D_w^{\mathrm{svc}} \geq D_w$, and $\min_n \beta_n > 0$, this is bounded above by a negative constant outside a bounded workload set.

The map in (42) has bounded arrival jumps J_R. Set $\widetilde{V}_\theta(X) := e^{\theta R_w(X)}$. For $|u| \leq J_R$, $e^{\theta u} - 1 \leq \theta u + C_R\theta^2$ at sufficiently small θ. Applying this inequality and the chain rule gives, for finite C_e and small $\theta > 0$,

$$\mathcal{L}_\pi \widetilde{V}_\theta(X) \leq \widetilde{V}_\theta(X)\big[\theta \mathcal{L}_\pi R_w(X) + C_e\theta^2\big].$$

The negative constant drift of R_w outside a bounded set then yields finite constants $c, b > 0$ and the bounded workload set $\mathcal{C} := \{X : R_w(X) \leq R_0\}$ such that

$$\mathcal{L}_\pi \widetilde{V}_\theta(X) \leq -c\widetilde{V}_\theta(X) + b\mathbf{1}\{X \in \mathcal{C}\}.$$

Dynkin's formula, Gronwall's inequality, and (43) give a uniform exponential moment of $\|\mathbf{W}\|_1$ (Down et al., 1995; Meyn & Tweedie, 2012).

In serving workload units, the serving batch residual satisfies

$$\dot{R}_n^L = \begin{cases} -\mu^{\mathrm{p}}/\upsilon = -\mu, & \text{during prefill,} \\ -\mu^{\mathrm{d}} = -\mu, & \text{during decode.} \end{cases} \tag{44}$$

Let $A^B(t-\tau, t]$ be the raw work arriving over $(t-\tau, t]$. Since serving workload decreases at rate μ while an instance is busy, FCFS gives

$$\Pr(B(t) > x) \leq \Pr\Big(\max_n Z_n(t-\tau) > \mu\tau\Big) + \Pr(A^B(t-\tau, t] > x). \tag{45}$$

Later requests cannot delay FCFS work present at $t - \tau$. For $t \geq \tau$, take $\tau = \vartheta x$ with sufficiently small $\vartheta > 0$; the workload moment controls the first term and a compound Poisson Chernoff bound controls the second. When $t < \tau$, the empty initial state gives $B(t) \leq A^B(0, t]$, so the same arrival bound applies. Thus $\lambda\Gamma_\pi(w) < 1$ yields a uniform exponential moment of B, proving system serving stability and the inner region inclusions in (18). Finally, $\varepsilon_{\pi_0}^L < 0$ excludes $\lambda \in \mathcal{R}_{\pi_0}$ by Theorem 1, while $\lambda \in \widehat{\mathcal{R}}_{\mathrm{C}}(w)$ implies $\lambda \in \mathcal{R}_{\mathrm{C}}$.

A.3.3 Proof of Robustness to Bounded Errors

Cache evaluation error.

Proof of Theorem 3(i). Let $\widetilde{\pi}_w(a \mid S) \propto \exp[-(q_a + e_a^{\mathrm{c}})/\iota]$ be the softmax policy under cache evaluation error, and let $\mathcal{L}_{w,\mathrm{c}}$ denote the generator of the realized *CARRIER* process under this policy. The softmax induced by $q + e^{\mathrm{c}}$ minimizes its expected action cost minus entropy. For any reference selector ϖ, the Gibbs inequality gives

$$\mathbb{E}_{\widetilde{\pi}_w} q_A \leq \mathbb{E}_\varpi q_A + \iota \log N + \mathbb{E}_\varpi e_A^{\mathrm{c}} - \mathbb{E}_{\widetilde{\pi}_w} e_A^{\mathrm{c}} \leq \mathbb{E}_\varpi q_A + \iota \log N + \mathrm{span}_a(e^{\mathrm{c}}).$$

For $\lambda[\Gamma_{\mathrm{C}}(w) + \kappa_{\mathrm{c}}/2] < 1$, we have $1 - \lambda\Gamma_{\mathrm{C}}(w) > 0$. Using the cache error bound in Theorem 3(i), (40), and $D_w^{\mathrm{svc}}(X) \geq D_w(\mathbf{W})$ gives

$$\mathcal{L}_{w,\mathrm{c}}\Psi_w(X) \leq C_{w,\mathrm{C}} + \lambda E_{\mathrm{c}} - [2\{1 - \lambda\Gamma_{\mathrm{C}}(w)\} - \lambda\kappa_{\mathrm{c}}]\, D_w(\mathbf{W}),$$

The coefficient is positive under the stated condition; the square root potential and FCFS clearing argument from Theorem 2 then apply. □

Decode size error.

Proof of Theorem 3(ii). Let

$$\overline{\alpha} = \max_n |\alpha_n|, \qquad \overline{\beta}_w = \max_n \beta_n.$$

Since the same e_d is used for all instances and does not change the cache transition, $\widehat{G}_a = G_a + e_d$, which induces the action cost error

$$e_a^{\mathrm{d}} = q_a(\widehat{d}) - q_a(d) = 2e_d\beta_a W_a + e_d(\alpha_a + 2\beta_a G_a) + \beta_a e_d^2. \tag{46}$$

The first term satisfies

$$\mathrm{span}_a(2e_d\beta_a W_a) \le 2\delta_d \sum_n \beta_n W_n = \frac{2\delta_d}{\mu} D_w(\mathbf{W}).$$

Because $0 \le G_a \le s_+$, the absolute value of the remaining terms in (46) is at most

$$C_d := \delta_d(\overline{\alpha} + 2\overline{\beta}_w s_+) + \overline{\beta}_w \delta_d^2.$$

Therefore

$$\mathrm{span}_a(e^{\mathrm{d}}) \le 2C_d + \frac{2\delta_d}{\mu} D_w(\mathbf{W}). \tag{47}$$

Applying the preceding drift argument with $E_{\mathrm{c}} = 2C_d$ and $\kappa_{\mathrm{c}} = 2\delta_d/\mu$ gives $\eta_{\mathrm{d}} = \delta_d/\mu$. For either error, solving the corresponding condition for λ gives (19); dividing its request rate boundary by the error free boundary gives the stated relative contraction. □

A.3.4 Proof of *CARRY* Update Stabilization

Proof of Theorem 4. Projection gives $w_k \in \mathcal{W}$ by induction. Its nonexpansiveness yields

$$\|w_{k+1} - w_k\|_2 \le \alpha_k \|g_k\|_2.$$

By Tonelli's theorem and Assumption 3,

$$\mathbb{E}\left[\sum_{k\ge 0} \|w_{k+1} - w_k\|_2^2\right] \le \left(\sup_k \mathbb{E}\|g_k\|_2^2\right) \sum_{k\ge 0} \alpha_k^2 < \infty.$$

The nonnegative sum is therefore finite almost surely, so its terms vanish. □

A.4 Evaluation Details

Simulation Infrastructure. The practical simulator extends Vidur with *CARRIER*'s affinity group cache state and retention and eviction dynamics while preserving event based serving and hardware profiled phase times. The nominal simulator implements *CARRIER* with Poisson arrivals, empirical request profiles, first come first served scheduling, and constant phase rates calibrated from low load practical simulator traces. The same traces calibrate the feature scales. Thus, the nominal simulator retains the routing relevant arrival, waiting, and cache dynamics while abstracting kernel timing.

Application scenarios. The three application scenarios are conversation (ShareGPT), tool and agent (OpenHands), and long context (LooGLE). The main study uses eight A100 instances. ShareGPT uses 1000 groups, a 5000 block cache per instance, and request rates of 1–8 requests/s; LooGLE and OpenHands use 300 and 150 groups, 20000 blocks, and 1–4.5 requests/s. Across four seeds, these scenarios contain 6810–6887, 7803–8009, and 9487–9784 requests, respectively. ShareGPT is capped at 16384 tokens per request, LooGLE removes groups containing a request above 100000 prefill tokens, and OpenHands caps requests at 99374 tokens. The scale study uses 1000 OpenHands groups, 50 instances, and request rates of 10, 20, 30, and 40 requests/s.

Theory Validation. Figure 5 below verifies our theorems on stability, weight convergence and robustness.

For stability, results shown on subgraph (a) validates Theorem 2. Under the same system capacity, *CARRY* has a much lower system workload comparing with all baselines. Specifically, its workload peak value is 20.5%, 25.0%, 35.7% and 37.0% less then *Cache Greedy*, *Dualmap*, *Min TTFT* and *Load Balance* respectively. We can deduce that *CARRY* has a much larger serving stability margin which implies a larger system serving capacity and ensures the capability of keeping stable under larger request rates. Therefore, system using *CARRY* can serve a much higher request rate since *CARRY* has an obvious capacity advantage comparing with all baselines.

For robustness, in subgraphs (b) and (d), we compare *CARRY* and *CARRY* with observation error of cache and decode size separately. Empirical results show that within 20% cache error, the P50 TTFT and the cache hit rate stay within 6.2% and 4.2% of error-free *CARRY*; within 80% cache error, the

P50 TTFT and the cache hit rate stay within 7.2% and 1.3% of error-free *CARRY*. This verifies that within a bounded observation error, *CARRY* still enables the system to stay stable under a certain capacity range, validating Theorem 3. .

For convergence, we analyze the update norm of feature weights of *CARRY* under various request rates on OpenHands 1k. Results in subgraph (c) show that weight update vanish after limited update counts, validating Theorem 4.

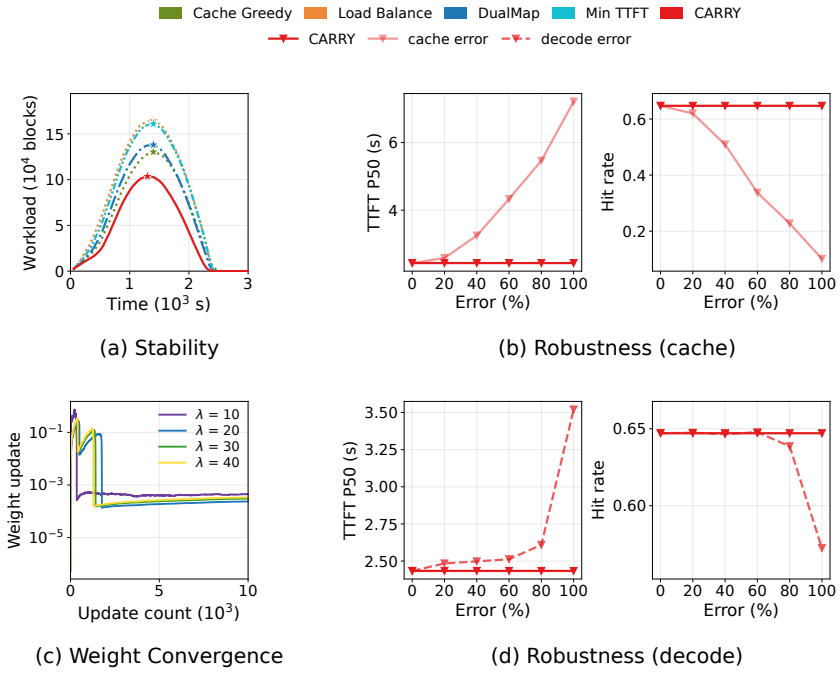

Figure 5: Theory validation on stability, weight convergence and robustness.

Extreme cases. Each full severity case uses four A100 instances, 5000 cache blocks per instance, 10000 requests, seeds 42–44, and request sizes from 4096 to 8192 tokens. Arrival overload uses 64 groups at 20 requests/s to represent a traffic surge. Pressure boundary uses 40 groups at 4 requests/s to place the reusable working set near aggregate cache capacity. Single hotspot activates one long lived group among 128 groups at 10 requests/s, representing a dominant session, whereas many hotspots activates 64 groups to represent many simultaneously popular sessions.

Figure 6 demonstrates the performance on TTFT P90 and cache hit rate of *CARRY* and all baselines under four extreme cases: arrival overload, many hotspots, pressure boundary and retention collapse. These four extreme cases are created under Vidur by changing relevant parameters. Arrival overload keeps the conversation scenario and orignal groups, but increases the Poisson rate to 20 req/s which is five times the baseline; Many hotspots allows 64 hot prefixes are live at once, so that the cache couldn't hold them together; Pressure boundary tests the behavior of all policies when reusable working sets pushing up against cache capacity boundary; Retention collapse changes the retention proportion of prefix when it's halfway through the trace. The results show that *CARRY* still outperform other baseline methods under all four extreme cases. Specifically, the TTFT P90 gap is largest at pressure boundary at which *CARRY* 18% lower than the second best method *Cache Greedy* and the cache hit rate gap is largest at arrival load where *CARRY* 38% higher than *Cache Greedy*.

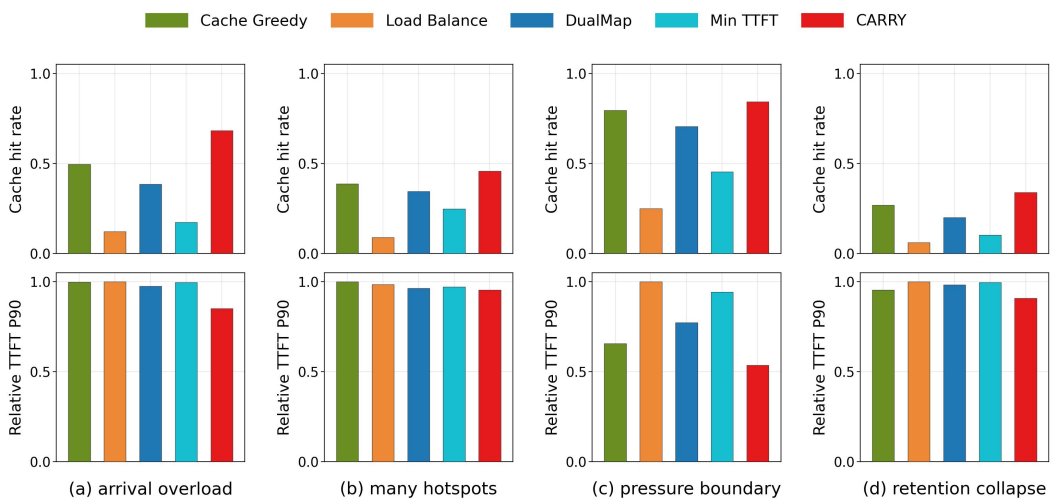

Figure 6: Performance comparison between *CARRY* and all baselines under four extreme cases: arrival overload, many hotspots, pressure boundary and retention collapse.

Training Modes. *CARRY* has three training mode variants: critic-only adaption, finetuning and training from scratch. In comparison with *CARRY*, critic mode allows value adaption on practical simulator and finetuning mode allows both value and policy adaptions on practical simulator. However, instead of training on nominal simulator, training from scratch runs on practical simulator directly and simultaneously adapt value and policy. Figure 7 shows that finetuning and critic mode have almost the same performance like *CARRY* and greatly outperform train from scratch mode. This result implies that pre-training experience on nominal simulator is necessary for well performance and later adaptions on practical simulator has ignorable affect on performance. Therefore, *CARRY* demonstrates a balance between well performance and algorithm complexity in comparing with all its variants.

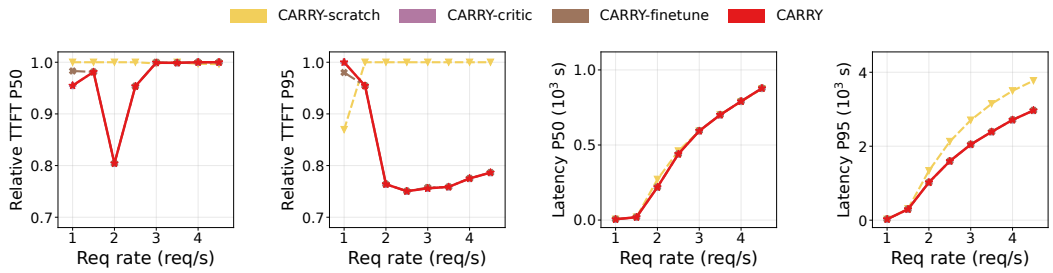

Figure 7: Different training modes of *CARRY* on OpenHands 150, containing finetuning, critic-only adaption, training from scratch and original *CARRY*.

Ablation Studies. Figure 8 shows the feature ablation of *CARRY* on OpenHands 150. Variants include dropping the workload feature, the effective-workload feature, or the quadratic workload feature. *CARRY* and its variants' performance are measured by TTFT P95, latency P95, imbalance at $\lambda = 1$ and throughput. Dropping the workload feature makes TTFT and latency the worst at every rate. Dropping effective workload is nearly as costly once the request arrival rate is increasing; its relative TTFT P95 rises from 0.25 at $\lambda = 1$ to 0.54-0.66 for $\lambda \geq 2$. Dropping only the quadratic workload term leaves relative TTFT P95 near 0.75 from $\lambda = 2$ onward. *CARRY* is 44% lower in P95 TTFT and 42% lower in P95 latency than the runner-up. The imbalance trace shows the same ranking; *CARRY* keeps the hotspots lowest and drains it fastest; without the quadratic term the peak is about an order of magnitude higher. This shows that the quadratic workload feature plays an important role in detecting extreme hotspots among servers. Throughput is seperated obviously after the arrival rate

increases. At $\lambda = 4$, *CARRY* reaches 1.55 req/s, against 1.21, 1.12, and 1.08 req/s for the quadratic, effective-workload, and workload ablations. Overall, the above results imply that the two workload features are important for *CARRY* to outperform against other methods.

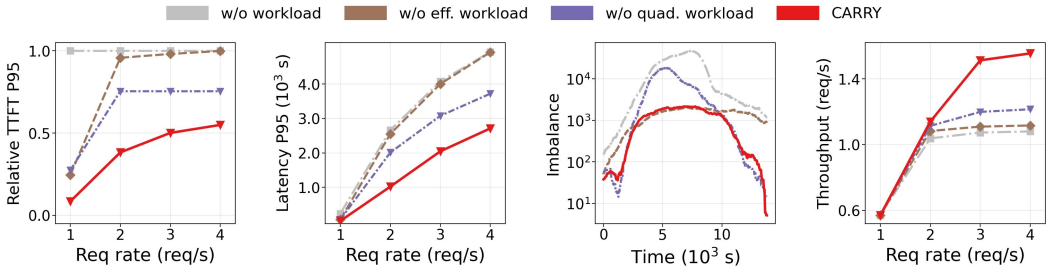

Figure 8: Feature ablations of *CARRY*. Variants drop the workload feature, the effective-workload feature, or the quadratic workload term.

LLM Scales. Figure 9 shows the results on comparing *CARRY* and all baselines on OpenHands 150 using Llama-3-70B. Similiar with the results using Llama-3-8B, *CARRY* leads on every metrics. Relative TTFT P95 for *CARRY* stays in 0.54-0.66, against 0.69-0.80 for *Cache Greedy* and 0.76-0.83 for *Dualmap*; *Load Balance* and *Min TTFT* holds the worst two TTFT P95. Latency P95 tracks the same gap: *CARRY* is 18% below the next-best method, *Cache Greedy*. The hit-rate panel accounts for the margin. *CARRY* holds a flat hit rate of 0.67 while *Cache Greedy* falls from 0.66 to 0.56 when request rate increases. *Dualmap* maintains a stable cache hit rate across the grid (stays near 0.40) and *Load Balance* and *Min TTFT* hold a cache hit rate not greater than 0.20. This implies that these two algorithms are less cache affinity than others. For throughput, *CARRY* sustains about 0.46 req/s at every rate, 18% higher than the runner-up. Overall, on simulation model Llama-3-70B, *CARRY* still outperforms all baselines across selected metrics.

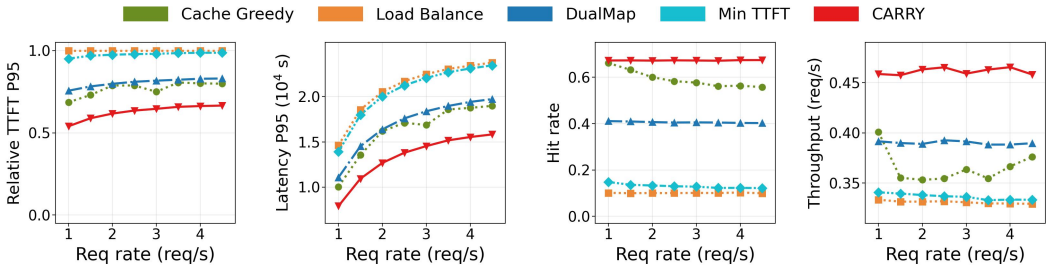

Figure 9: Performance comparison between *CARRY* and all baselines on Llama-3-70B.