
The Price of a Miss: Congestion-Priced KV Scheduling for Agentic LLM Serving

Jinhwan Suk¹

Abstract

An agentic LLM server keeps a program’s KV cache across tool calls, and every KV decision asks what one miss costs. For replicas with a prefill queue, a decode batch and one memory pool we derive the *price of a miss*, the time to first token it adds summed over all turns. Beyond its recompute, a miss delays every prefill queued behind it, a quadratic term that chunked prefill does not remove; the decode batch sees only mean work. One shadow price of memory decides eviction, offloading, placement and admission. With few sessions live the open-queue wait is an upper bound. On an NPU testbed at 50k-token contexts turns wait for KV blocks, not prefill compute, and the turns that miss are those that queued; at short contexts forced misses raise the prefill queue by an amount inside the price’s bracket, over half of it the head-of-line term.

1. Introduction

Response time is queueing delay plus service time, $T = W_q + S$. Hardware benchmarks measure S ; users experience T , and near saturation W_q dominates: in the simplest Markovian model, raising the load from $\rho = 0.90$ to 0.99 multiplies response time by ten. The serving question is therefore which control knob moves the load at which the time to first token (TTFT) crosses its service-level target.

Agentic workloads sharpen the problem. In production coding-agent traces the median session has 58 LLM turns over a context of 207k tokens, and almost every turn adds a tool result to a prompt the server has already processed and may still hold in memory (§4.1; [vLLM Team & Infract, 2026](#)). The unit of work is the *program*, a session with persistent KV state ([Kang et al., 2026](#)), whose KV occupies

Working draft v0.14. Of the measurements designed in Appendix E, the cost fit and the two replays are reported (§4.3); the other four are pending. ¹Rebellions Inc., Seongnam, Republic of Korea. Correspondence to: Jinhwan Suk <jinhwan.suk@rebellions.ai>.

memory while it waits for a tool.

Every KV decision in such a system asks what it costs to lose a paused program’s state. The obvious answer is the time to recompute it. In a queue that answer is incomplete: a miss is long work added to a shared server, every other request pays for it, and how much depends on the load and on whether the miss shares the batch or blocks it. Queueing models of LLM serving have so far asked when a server is stable and which schedulers are throughput-optimal, with output length as the source of variability ([Yang et al., 2024](#); [Dai et al., 2025](#); [Ao et al., 2025](#); [Nie et al., 2026](#)); agentic serving systems keep or evict the KV of paused programs by rules of thumb such as recency, time-to-live or idleness ([Li et al., 2025](#); [Xia et al., 2026](#); [Kang et al., 2026](#)). We connect the two. The variability that matters here is in the prefill work, and it has two sources: the size of the append, which the scheduler does not control, and the hit/miss mixture that the KV policy creates, which it does. The second can be priced per miss, and that price is the decision variable the policy needs.

This paper makes three contributions.

1. **A formulation and the price of a miss (§2).** Sessions arrive as a Poisson process and loop through turns and tools. A replica has a prefill queue that serves pending prefills in order, a decode batch that shares bandwidth, and one memory pool. A miss changes only the prefill work, and we derive its price at the prefill queue as the delay one longer job adds to itself and to the jobs behind it (Proposition 1); the decode batch depends on the work only through its mean, so a miss leaves it unchanged (Proposition 3). With N sessions live a waiting session cannot submit its next turn, and for exponential work we prove that the usual open-queue wait at the same utilisation overstates the true wait (Proposition 2).
2. **Congestion-priced scheduling (§3).** The scheduler compares one number per paused program, its expected price per byte of memory per second held, with one shadow price of memory. That comparison drives eviction, offloading, placement and admission. Eviction is a covering knapsack whose standard solutions we cite. Any eviction rule that ignores resume probabilities can

be made arbitrarily bad, and shortest-first is the special case in which the price is the recompute alone (Proposition 4); in the traces we measured it is the pause time, not the resume probability, that separates programs.

- Results (§4).** Production traces give the workload side: the append alone has a prefill-work CV^2 of 32.0, resume probabilities vary little and pause times vary widely. In the synthetic simulation, the admission cap changes hit rate and latency much more than the choice between priced and shortest-first eviction, whose differences stay within seed noise. On replayed sessions under vLLM’s engine rules the open price overstates the rise from forced misses by 3 to 12 times. On an NPU testbed (§4.3) the fitted prefill cost is quadratic past 30k tokens; a replay of a coding-agent trace at 50k-token contexts shows turns waiting for KV blocks, which the cap on live sessions controls, and the misses among the turns that queued; and a short-context replay, where the prefill queue binds, puts both wait formulas within a factor of two of the observed wait and the rise from forced misses inside the bracket of Proposition 1, over half of it the head-of-line term and over half of it borne by turns that hit. The remaining experiments (Appendix E) are not yet run.

Propositions are our results, proved in Appendix A; standard results are cited, not re-proved; other papers are quoted by section and figure, and we have not re-run their experiments.

2. Problem Formulation

Background. An LLM request is served in two phases. Prefill reads the whole prompt in one pass and produces the first token; the wait for it to start plus its duration is the time to first token (TTFT). Decode then produces one token per step. Both phases keep the attention state of every prompt token, the KV cache, in device memory; when a later request starts with the same tokens, prefix caching reuses that state (a *hit*) instead of recomputing it (a *miss*). An engine such as vLLM batches the decode steps of many requests and, with chunked prefill, feeds pending prefills into those steps in fixed-size chunks (Agrawal et al., 2024). An agent program is a session of many requests, each appending a tool result to the previous prompt, so nearly every request could be a hit if its state survived the tool call. The memory held by paused sessions is what the KV policy manages.

Table 1 is the model on one page: sessions arrive and loop through turns and tool calls (§2.1); each turn is served by one of J replicas, each a prefill queue, a decode batch and a memory pool (§2.2); the KV policy decides which turns hit and so what the prefill queue serves (§2.3); §2.4 states the decision problem and derives the one quantity every de-

Table 1. The model at a glance.

System	J replicas, each with a memory pool of M bytes; optionally a slower shared KV store that any replica can read.
Sessions	Poisson arrivals at rate Λ ; a session loops prefill $\rightarrow$ decode $\rightarrow$ tool; a paused program i resumes with probability p_i after an expected τ_i seconds; turns reach the replicas at rate λ .
Turn i	context K_i , n_i new tokens, o_i output tokens. Demands: prefill P (1), decode D_i (2), and $c_i = \kappa K_i$ bytes of memory while the state is resident.
Prefill	FIFO queue served from the token budget the decode batch leaves; wait $W_q = W_q^P + W_q^M$, the queueing wait of (3) plus the wait for blocks; TTFT = $W_q + S$; a miss adds ΔS_i to the work.
Decode	processor sharing with capacity $\varphi(m)$ for m turns in the batch; occupancy depends on the work only through its mean.
Decisions	keep, offload or drop each paused state (a dropped or remote state makes the next turn a miss or a transfer); place each resuming turn; admit sessions.
Objective	$L = L_P + L_D$, the mean number of turns in the prefill stage and in the batch, s.t. resident bytes $\leq M$ per replica; θ is the shadow price of a byte-second; Φ_i (4) is the TTFT one miss of program i adds over all turns.

cision in it needs.

2.1. Sessions, Turns and Tools

An agent program, or session, cycles through prefill $\rightarrow$ decode $\rightarrow$ tool $\rightarrow$ prefill $\rightarrow \dots$. Sessions arrive in a Poisson process of rate Λ ; inside a session the loop is closed: a turn is served, its tool call takes a random time, the *think time* of an interactive network (Lazowska et al., 1984), and with probability p_i the session issues another turn. Turns reach the replicas at rate $\lambda = \Lambda \mathbb{E}[\text{turns per session}]$. For a network of processor-sharing stations and delay stations (stations without queueing, such as the tool call) the BCMP theorem gives a product-form stationary law (Baskett et al., 1975): each station behaves as if fed by Poisson arrivals at its own rate, whatever the tool-time law. The decode stage is such a station. The FIFO prefill stage is not, and we use the same Poisson reading for it as an approximation, tested in §4.2 and §4.3.

2.2. A Replica: Two Resources and a Memory Pool

A turn with context K_i appends n_i tokens and decodes o_i . It makes three demands on its replica.

- Prefill compute.* Prefilling n tokens onto a resident prefix of K costs

$$P(n, K) = a n + b n (K + n/2), \quad (1)$$

a linear term for the dense layers and an attention term over the prefix. A hit costs $P(n_i, K_i)$; a miss re-prefills the context, $P(K_i + n_i, 0)$, quadratic in K_i , as Thunder-Agent’s Lemma 4.1 also assumes (Kang et al., 2026).

- *Decode bandwidth.* Each decode step reads the weights once for the batch and the KV of every turn in it, so decoding o_i tokens takes

$$D_i = o_i(\beta K_i + \omega/m) \quad (2)$$

seconds of bandwidth, with m the batch size, β the time to read one token’s KV and ω the time to read the weights. Which term dominates depends on the hardware (on our testbed $\beta \approx 0$, §4.2); neither differs between a hit and a miss.

- *Memory.* $c_i = \kappa K_i$ bytes for as long as the state is resident, running or paused, with κ the KV bytes per token.

The hit/miss distinction lives entirely in the first demand.

Decode stage. Continuous batching advances every turn in the batch at each iteration, so decoding turns share the device. We model the decode stage as a processor-sharing (PS) station: a batch of m turns completes $\varphi(m)$ seconds of demand (2) per second, and every turn in it advances at the same rate; batching makes φ increase with m and memory makes it saturate. The stationary law of the batch size is $\pi(m) \propto \rho_D^m / \prod_{k \leq m} \varphi(k)$ with the decode load $\rho_D = \lambda \mathbb{E}[D]$, and it is *insensitive*: it depends on the demand only through its mean (Baskett et al., 1975; Kelly, 1979). This takes φ as fixed by the engine’s step rule, which serves the batch at every step (next paragraph), so that the prefill load does not enter it. An engine that instead gives a prefill a step of its own stops its decodes for that step, and its decode capacity falls with the prefill load; in that model a miss adds no decode demand but shrinks the capacity, and a smaller share of the capacity at the same demand leaves more turns in the batch. Proposition 3 is about the first kind of engine.

Prefill stage. With chunked prefill each step first serves the decode batch, then spends the tokens left in its budget on pending prefills (Agrawal et al., 2024). Chunking protects the decoding turns from a long prefill; it does not let one prefill overtake another, and by default the schedulers we know of serve pending prefills in arrival order. vLLM serves its running requests in admission order rather than decode first; the two orders coincide as long as no per-request cap limits a prefill’s tokens in a step, because a prefill that does not finish leaves no budget for the requests behind it. We therefore model the prefill stage as a FIFO queue whose server is available a fraction $1 - \rho_D$ of the time; we write S_i for the work P_i of turn i measured in that server’s own time and $\rho = \lambda \mathbb{E}[S]$ for the prefill load. For Poisson arrivals to a FIFO server with load $\rho < 1$ and

$\mathbb{E}[S^2] < \infty$ (M/G/1) the Pollaczek–Khinchine (PK) formula gives the mean wait

$$\mathbb{E}[W_q] = \frac{\lambda \mathbb{E}[S^2]}{2(1 - \rho)} \quad (3)$$

(Pollaczek, 1930; Khinchine, 1932). Session feedback makes arrivals non-Poisson and the budget fluctuates, so (3) is an approximation here. With N sessions live the queue is a finite-source system, and a session that is waiting cannot generate its next turn: Proposition 2 shows that the open formula then bounds the Markovian wait from above. The two stages thus differ in what they need from the cache: the decode stage needs only mean demands, which the cache does not change; the prefill stage needs the second moment of its work, which the cache sets.

Memory pool. The batch and the paused states share one pool of M bytes, so a replica admits turns into the batch until the next one does not fit. The batch that fits is a property of the footprint law and the admission rule, not of the mean footprint (a more variable footprint can shrink the batch or enlarge it), so φ is to be calibrated (Appendix E), not derived. A turn whose KV does not fit waits for blocks before its prefill starts; that wait W_q^M adds to the queueing wait W_q^P of (3) and is set by the pool, not by the queue. Once resident KV exceeds the pool the policy chooses what to drop, which sets the hit rate h . A shared KV store (Qiao et al., 2026) adds a slower tier every replica can read, so a state that is not resident is fetched instead of recomputed.

2.3. Prefill Work under KV Reuse

With hit probability h the prefill work of a turn is the mixture $S = S^{\text{hit}} = P(n_i, K_i)$ with probability h and $S = S^{\text{miss}} = P(K_i + n_i, 0)$ otherwise, so the KV policy sets $\mathbb{E}[S]$, $\mathbb{E}[S^2]$ and ρ . At the FIFO prefill stage, (3) makes the second moment a first-class quantity: of two work laws with the same mean, the one with the larger variance waits longer at every stable load, and raising the hit rate lowers both moments at once, so the PK wait is non-increasing in h .

We do not assume that agentic workloads have intrinsically high variance; we ask where it comes from. Table 2 splits $\text{Var}[S]$ into the spread of the appends within hits and within misses and the hit/miss mixture between them. With every turn a hit, as the prefix hashes of the production traces say 98.8% of their tokens would be, the append alone gives $\text{CV}^2 = 32.0$ under a linear cost model (tool outputs range from a line to a file). Forcing 1 to 10% of turns to miss adds the mixture, which then supplies 47 to 58% of the variance under the linear model and 23 to 35% once attention dominates; the mixture rivals the appends only once eviction is active (§4.2). What the price of a miss must capture is there-

Table 2. Where the variance of prefill work comes from, on 26 439 production Claude Code requests (§4.1). Work is (1) with the crossover context $K_c = a/b$ at which the attention term equals the dense term ($K_c = \infty$: the linear model). Miss/hit: mean miss work over mean hit work. All hit: CV^2 of the work when every turn hits as the prefix hashes say it would. Then the CV^2 of the hit/miss mixture at hit rate h , with the share of $\text{Var}[S]$ that the mixture contributes in parentheses; the rest is the spread of the appends.

K_c	miss/hit	all hit	$h=0.90$	$h=0.96$	$h=0.99$
∞	81	32	12 (58)	25 (57)	41 (47)
100k	78	55	20 (35)	39 (35)	66 (28)
30k	78	64	23 (31)	45 (30)	76 (25)
10k	77	68	24 (29)	47 (29)	80 (23)

fore not a ratio of CV^2 but the rise of $\lambda \mathbb{E}[S^2]/(2(1-\rho))$ that one more miss causes.

2.4. The Price of a Miss

At a decision epoch each replica holds a set of paused programs. Program i holds c_i bytes, resumes with probability p_i after an expected further τ_i seconds, and its next turn needs prefill S_i^{hit} if the state is resident on the replica that serves it and $S_i^{\text{miss}} = S_i^{\text{hit}} + \Delta S_i$ if it must be recomputed; with a shared store a remote state costs a transfer instead. The controller keeps, offloads or drops each state, places each resuming turn, and admits sessions, subject to resident bytes at most M per replica, to minimise $L = L_P + L_D$ of Table 1, which by Little’s law is the delay accumulated per unit time. Relaxing the memory constraint with a multiplier $\theta \geq 0$, the shadow price of a byte-second, gives an additive cost: keeping program i costs $\theta c_i \tau_i$, and dropping it costs the price of the miss it may cause. A single eviction is a transient event. In the stationary model we price it as a small change of policy: a fraction q_i of arrivals, those of program i , turn from hits into misses, and we ask how L_P responds. Applied to one eviction this is an approximation; §4.2 and §4.3 test the stationary form.

Proposition 1 (The price of a miss). *Model the prefill stage as an M/G/1 FIFO server with arrival rate $\lambda \geq 0$, load $\rho < 1$ and mean wait $W = \mathbb{E}[W_q^P]$ from (3). Turn a fraction $q_i \geq 0$ of arrivals from hits of work $S_i^{\text{hit}} \geq 0$ into misses of work $S_i^{\text{miss}} \geq S_i^{\text{hit}}$ such that the new load $\rho' = \rho + \lambda \sum_i q_i \Delta S_i$ is below one, and let*

$$\Phi_i = \Delta S_i + \frac{\lambda((S_i^{\text{miss}})^2 - (S_i^{\text{hit}})^2)}{2(1-\rho)} + \frac{\lambda W \Delta S_i}{1-\rho}. \quad (4)$$

Then the change in L_P satisfies $\lambda \sum_i q_i \Phi_i \leq \Delta L_P \leq \frac{1-\rho}{1-\rho'} \lambda \sum_i q_i \Phi_i$.

Φ_i is in seconds: the TTFT, summed over all turns, that one more miss of program i adds; misses of program i occur at rate λq_i , so $\lambda \sum_i q_i \Phi_i$ is a rise in the mean number in

the prefill stage by Little’s law. Its three terms are three ways a miss hurts. The miss itself takes longer to prefill. Turns that arrive while it is being prefilled wait for all of it, and that wait grows with the square of the miss less the square of the hit, as the second moment in (3) does; this is the externality of the economics of queues (Naor, 1969; Mendelson & Whang, 1990), applied to a job whose size changes. Its extra work raises the load, which stretches every wait by $1/(1-\rho)$. The sum of these prices never overstates the damage and understates it only by the factor $(1-\rho)/(1-\rho')$. Which term dominates depends on the load: the second term is $\lambda(S_i^{\text{miss}} + S_i^{\text{hit}})/(2(1-\rho))$ times the first and $(S_i^{\text{miss}} + S_i^{\text{hit}})/(2W)$ times the third, so when few turns arrive during a miss the price is close to ΔS_i , when many arrive the turns queued behind it dominate, and once the mean wait exceeds half the sum of the miss and hit work the load term takes over and diverges as the stage approaches capacity.

Within a replica, at one load and equal append, Φ_i is non-decreasing in the context K_i , since $\Delta S_i = aK_i + bK_i^2/2$ and every term of (4) grows with it, so the load alone never reorders programs of equal append by Φ_i . It does reorder them by the expected price per byte $p_i \Phi_i / c_i$ once p_i differs: with $K_c = 50\text{k}$ tokens, a 200k-token program resuming with probability 0.3 and a 20k-token one resuming with probability 0.9 change order between an idle queue and $\rho = 0.5$, which no fixed key reproduces.

The M/G/1 model assumes an unbounded population. A pool of a few million tokens holds about ten sessions of the workload of §4.1, and a session that is waiting for prefill cannot submit its next turn, so fewer turns arrive while the queue is long. For exponential work the finite-source model captures this exactly.

Proposition 2 (Finite population). *Let $N \geq 1$ sessions each think for a time of mean $Z > 0$ and then submit a turn of exponential work with mean $1/\mu$ to one FIFO server. The mean number at the server with n sessions is given exactly by mean value analysis (Lazowska et al., 1984, §6) as $Q_0 = 0$, $Q_n = n(1 + Q_{n-1})/(\mu Z + 1 + Q_{n-1})$; with N sessions the utilisation is $\rho_N = N/(\mu Z + 1 + Q_{N-1}) < 1$ and the mean wait in queue is $W_N = Q_{N-1}/\mu$.*

- (i) Q_n is nondecreasing in n , $Q_n \leq n$, and Q_n is nonincreasing in μZ .
- (ii) $W_N \leq \rho_N/(\mu(1-\rho_N))$, the open M/M/1 wait at the same utilisation.

The think-time law enters only through Z , since the delay station is insensitive. Part (ii) says that the wait W in the price of Proposition 1 is at least the Markovian finite-source wait at the same utilisation; with two sessions and $Z = 2/\mu$ the open wait is 4.5 times it, and Table 10 shows the ratio falling to 1.09 at $N = 64$. Since at most N turns exist, a policy change with N live sessions raises L_P by at most

$N - L_P$, whatever the work law, and the last part of (i) gives a finite-source price of longer work, the rise of Q_N when the mean work grows; §4.2 and §4.3 compare both prices with the rise a real workload shows.

Proposition 3 (The decode stage prices only work). *Consider the PS decode stage of §2.2 with any nondecreasing capacity φ and load $\rho_D < \sup_m \varphi(m)$.*

- (i) L_D is nondecreasing in ρ_D . Hence, at every load, a policy whose turns add no more expected demand than another’s leaves no more turns in the batch.
- (ii) For constant capacity C and added demand $\lambda \sum_i q_i \Delta D_i$ raising the load to $\rho'_D < C$, the change in L_D is exactly $\frac{C-\rho_D}{C-\rho'_D} \lambda \sum_i q_i \Phi_i^D$ with $\Phi_i^D = C \Delta D_i / (C - \rho_D)^2$.

A miss adds no decode demand, so to first order the decode stage is untouched by it; the load factor of part (ii) is what a decision that adds decode demand, such as admitting a session, pays. The prefill queue is where a miss is paid for, with a head-of-line term that batching does not remove, and the decode batch is where the memory a kept state occupies would otherwise buy capacity. We define the expected price of dropping program i ’s state, its price per byte and its price per byte-second held as

$$w_i = p_i \Phi_i, \quad v_i = \frac{w_i}{c_i}, \quad u_i = \frac{w_i}{c_i \tau_i}. \quad (5)$$

3. Congestion-Priced Scheduling

The formulation leaves the scheduler two numbers. Each paused program has a price per byte-second u_i from (5), and each replica has a shadow price θ for a byte-second of its memory. Every KV decision compares the two. Algorithm 1 states the scheduler; this section derives its steps.

The inputs are estimates: ΔS_i from the cost model (1) calibrated once per replica (§4.3), p_i and τ_i from the program’s phase, tool and gap so far (§3.1), and the prefill queue’s rate, load and wait as running averages. The shadow price is the u_i of the marginal resident block, zero while the pool has room. Prices change with load, so a program’s rank can change while it waits, which is the point of pricing at decision time.

3.1. Eviction

When memory runs out the controller must free ΔC bytes at the least price,

$$\min_S \sum_{i \in S} w_i \quad \text{s.t.} \quad \sum_{i \in S} c_i \geq \Delta C, \quad (6)$$

a covering knapsack, NP-hard in general and approximable within a constant factor (Csirik et al., 1991; Carnes &

Algorithm 1 Congestion-priced scheduling at one replica

- 1: **Observe** the prefill arrival rate λ , load ρ and mean wait W , and the memory ΔC that must be freed.
- 2: **Price** each paused program: Φ_i from (4) with ΔS_i from (1), $w_i = p_i \Phi_i$, $u_i = w_i / (c_i \tau_i)$.
- 3: **Set** θ to the smallest value at which the blocks of the programs with $u_i \leq \theta$ free ΔC ; $\theta = 0$ if nothing must be freed.
- 4: **Evict** the blocks of the programs with $u_i \leq \theta$, or whole programs by the guarded greedy (§3.1).
- 5: **Keep, offload or drop** each remaining state by the least of $\theta c_i \tau_i$, the priced fetch and w_i (§3.3).
- 6: **Place** a resuming turn at the replica j minimising $W_j + S_j + R_j + F_j$ (wait, service, move cost, future cost at j , §3.2), with $R_j = \Phi_i$ at replica j ’s load, or the priced fetch if the state is in a shared store.
- 7: **Admit** sessions up to the count at which the expected resident KV of §3.3 fills the pool; hold the rest in an entry queue.

Shmoys, 2008). Engines evict blocks rather than programs, but from the tail of a context, and by (1) recomputing the last m tokens of a context of K costs $am + bm(K - m/2)$, concave in m : a context’s deeper blocks cost less per byte than its tail, and freeing tokens from one context costs no more than freeing as many from two. The block problem is therefore no fractional knapsack, and choosing programs remains its core. Its fractional relaxation, in which a program can be given up in part at its average price per byte, is solved by price per byte (Dantzig, 1957): for any $\theta \geq 0$ the states with $v_i \leq \theta$ cost no more than any other set that frees as much memory, and taking programs in increasing v_i until the target is reached costs at most any feasible set plus one program. The same holds per byte-second, with $c_i \tau_i$ in place of c_i and a target in byte-seconds, which is the order Algorithm 1 uses; its target is the bytes to free now, so ordering by u_i against a byte target is a heuristic mapping of that rule, and the simulation tests it (§4.2). The price needs the end of a program: a finished program resumes with $p_i = 0$, but an engine that keeps its blocks, as vLLM’s does, prices them as those of a paused one (§4.2). Where programs must be evicted whole, price-per-byte order can fail by any factor; the standard repair, a greedy guarded by a guess of the dearest evicted program, costs at most twice the optimum (Csirik et al., 1991) for any prices w_i .

Fixed keys. Deployed systems evict by a key fixed in advance, the shortest context first (Kang et al., 2026), the least recently used, the longest idle or the expired time-to-live (Li et al., 2025; Xia et al., 2026): Algorithm 1 with u_i replaced by a quantity that does not depend on p_i .

Proposition 4 (Price-blind keys). *Consider (6) with $w_i =$*

$p_i \Phi_i$.

- (i) Let a rule choose which of two programs to evict from any information except their resume probabilities. For every R there are resume probabilities in $(0, 1]$ under which the program it evicts costs more than R times the other, while either program alone frees the target. No such rule has a constant approximation ratio for (6).
- (ii) If every program resumes and its price is its recompute work alone, so that $w_i \propto c_i^2$ by (1), then shortest-first is the price-per-byte order: it costs at most twice any feasible set, and the factor 2 is tight.

The proposition takes programs as the units of eviction. Part (i) holds as well when an engine evicts blocks: its two candidates can be the tail blocks of two programs that differ only in p_i , since the argument needs nothing but a common positive recompute cost. Part (ii) is about whole programs.

Part (ii) is the regime in which shortest-first is defensible: light prefill load, where the price is close to ΔS_i , and programs that all resume. Even there it is not optimal: with contexts 4, 5, 6 and a target of 6 it evicts $\{4, 5\}$ at cost 41 while $\{6\}$ is feasible at cost 36, contrary to the optimality theorem of Kang et al. (2026, Def. 4.1, App. F.3). Outside that regime part (i) applies to every key that ignores p_i . How much p_i and τ_i vary is a workload question (Table 16, Appendix C): in the production traces the probability of another turn within ten minutes is 0.85 to 0.99 however one conditions, while the expected gap varies from 4 to 65 s. There the byte-second denominator of u_i , not p_i , separates programs, so a time-to-live key is closer to the price than shortest-first; part (i) says what a key cannot see, not how often it is fooled.

3.2. Placement

A resuming turn can be served where its state is or elsewhere. The router charges each replica j the wait and service there, the cost R_j of moving or recomputing the state, and the expected future cost F_j of the program’s remaining turns. With replica 1 holding the state ($R_1 = 0$), the rule $\min_j W_j + S_j + R_j + F_j$ keeps the turn there iff

$$(W_1 + S_1) - (W_2 + S_2) < R_2 + (F_2 - F_1). \quad (7)$$

The price of a miss supplies R_j : a move turns the next turn into a miss served at replica j , so R_j is Φ_i at that replica’s prefill load, small at an idle replica even for a long context. Session affinity, the rule that never moves a program, sets $R_j = \infty$ elsewhere. Against an idle replica with the same service rate μ for whole turns, moving costs $1/\mu + R + F$, and the M/M/1 response time of the affinity replica exceeds it exactly when its utilisation exceeds

$$\rho^* = \frac{\mu(R + F)}{1 + \mu(R + F)}, \quad (8)$$

which is nondecreasing in R and F . The idle alternative gives the earliest inversion, so ρ^* is a lower bound on the load at which affinity loses to any real alternative. With a shared KV store R_j becomes the priced fetch of §3.3 and that load falls: affinity and a store are alternatives rather than complements.

3.3. Admission and Offloading

The pool holds both the batch’s working set and the paused states, and the two feed each other: a miss stays longer and re-inserts its context, which evicts paused state and causes more misses. The loop amplifies misses at moderate load (§4.3) and thrashes the replica at high load (§4.2), as Ao et al. (2026) find without reuse. Admission keeps the pool out of that regime. Let a session live for a time T_s and hold $K(t)$ tokens at age t . Treating live sessions as an M/G/ ∞ system (each session its own server, none queues), Campbell’s theorem (Kingman, 1993) gives the mean resident KV as $\Lambda \mathbb{E}[\int_0^{T_s} K(t) dt]$, which exceeds the naive $\Lambda \mathbb{E}[T_s] \mathbb{E}[K]$ whenever long contexts belong to long sessions (by $1.15\times$ in the production traces and $0.98\times$ in our SWE-bench Pro runs). The admission step caps live sessions where this estimate fills the pool; §4.2 shows that at higher load the cap changes hit rate and latency much more than the choice between priced and shortest-first eviction. The cap has a price of its own, the wait at the entry queue, and a session is worth admitting while the memory it will hold, at θ per byte-second, costs less than that wait.

A state is kept, offloaded to a slower tier or dropped by the least of $\theta c_i \tau_i$, the priced fetch and w_i , where the priced fetch is Φ_i with the transfer time in place of ΔS_i when the fetch holds the prefill stage, plus the tier’s wait once it saturates. Since offloading only enlarges the choice set, enabling it never raises the optimal cost, while always offloading can be worse than never offloading, so the collapse of an always-offloading stack reported by Kang et al. (2026, App. A.2) is a property of that policy, not of the mechanism. Every rule in use replaces one price by a constant: shortest-first sets $w_i \propto c_i^2$, recency and time-to-live keys set p_i to a constant, always offloading sets the priced fetch to zero, strict affinity sets $R_j = \infty$, and admitting every session sets $\theta = 0$.

4. Results

Production traces fix the workload side of the model (§4.1). A simulated replica serving those sessions tests the propositions where an assumption fails (§4.2). An NPU testbed gives the measurements (§4.3; the remaining experiments are designed in Appendix E).

4.1. Real-World Traces: the Workload

Two trace sets give the workload: a public corpus of 183 production Claude Code sessions with 26 439 main-agent requests recorded at an API proxy (SemiAnalysis, 2026), with prefix-block hashes, output lengths, the gap before each request and the TTFT the proxy observed (Table 14, Appendix C), and our own 99 Claude Code runs on SWE-bench Pro tasks against an open-weight model, with per-call token counts. Under a never-evicting cache 98.8 % of prompt tokens would be prefix hits; the 2.1 % of requests that reuse less than half their prompt are context compactions, the workload’s own misses. The append is the variable part: its CV^2 is 32.0 here and 6.8 in the SWE-bench Pro runs, before any eviction, and Table 2 gives the split. The gap before a request (tool time plus the user’s pauses) has a median of 3.8 s and a p90 of 88 s; 13 % of gaps exceed a minute, the memory pressure of §3.3. The production cache and load are unobserved; TTFT at the proxy is not explained by the append or the context ($R^2 = 0.01$ and 0.06 in log-log, Table 15), as the model predicts when almost every turn hits.

4.2. Simulation: Replayed Production Sessions

The simulator replays the production sessions of §4.1 against a simulated replica that follows the vLLM v1 engine’s rules. Every iteration hands a token budget to the running turns in admission order, one token to each decoding turn and the rest to the prefill at the head, so prefill runs on the compute the batch leaves, as in §2.2. A waiting turn is admitted at the start of an iteration once its whole prompt fits; its KV then grows chunk by chunk, and a growth that does not fit preempts the latest admitted turn. KV is cached in 16-token blocks and evicted from the tail of a context, so an evicted turn re-prefills, with the work (1), only the part it lost. A finished session’s blocks stay cached, since the engine cannot tell that the session ended. The workload side is measured and the cost model is the testbed’s (§4.3): the prefill terms of the cost fit and a decode iteration of $\omega = 0.057$ s, the inter-token latency of the lightest long-context replay, which did not vary with the context ($\beta = 0$); no number here is a measurement of a serving system. Before the replay, every proposition is checked as a seeded test under its own assumptions and with one assumption dropped (Appendix B; 39 of 39 checks pass). Three of those checks matter later. On a replica with vLLM’s engine rules, forced misses land at the upper end of the bracket of Proposition 1 and change the number in the decode batch by at most 0.3 %: a miss’s prefill lengthens the iterations the decoding turns share. On open sessions at the vLLM-rule replica with the testbed’s cost model (Table 7), the priced and shortest-first eviction orders are indistinguishable within seed noise. Least recently used matches them at the lower load and pays 1.8 times their time to first token

at the higher load. At the higher load, increasing the admission cap reduces the hit rate and raises time to first token (Table 8). The price needs the end of a session: when a finished session’s blocks stay, as vLLM’s engine keeps them, the byte-second order cannot tell them from a paused session’s, and at the lower load its hit rate falls from 0.97 to 0.54 and its time to first token rises 2.4 times. The load at which moving beats affinity follows ρ^* of (8).

Replayed production sessions. Table 12 (Appendix B) plays each real session’s appends, outputs, think times and length on the replica; a gap above ten minutes starts a new session whose first prompt is the accumulated context (Table 13 varies the threshold). With no eviction every follow-up turn hits and its prefill work still has $CV^2 = 27\text{--}38$ from the appends alone, while the mean TTFT of 63–208 s is set by the cold first prompts. Once the pool is finite the cap on live sessions binds and the replica is a closed system of N sessions. At the smaller N of each pool a follow-up turn reuses on average at least 0.92 of its reusable prefix, although only a share of 0.60 or more reuses all of it: eviction takes the tail of a context, so most misses are partial and cheap, and the hit/miss mixture supplies only 6–22 % of the variance of follow-up prefill work. At the larger N the replica thrashes (hit rate at most 0.65, mean TTFT above 964 s). The PK wait from the measured λ , $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ is above the observed wait in every cell, as Proposition 2 says it must be for exponential work: by 7 to 15 times in the 6 cells whose prefill load stays below one in every seed, and without bound in the other 2, where a seed’s load reaches one. For the price itself, Table 11 forces a share δ of follow-up turns to reuse nothing, as a nonce at the head of the prompt would, and compares the rise of the number in the prefill stage with the bracket of Proposition 1 and with the finite-source price of Proposition 2. With 2 to 9 sessions live, the open bracket’s lower end overstates the rise by 3 to 12 times. At the higher rate and $\delta = 10\%$ the added work puts the open queue past its pole while the capped replica stays finite. The Markovian finite-source price, which ignores the second moment of the work, understates the rise by 2 to 18 times. The cap $N - L_P$ holds in every cell.

4.3. Testbed Measurements

The testbed is one replica group of MiniMax-M2.7 (fp8 weights) on four RBLN-CR13 NPUs, data parallel with expert parallelism, one rank per NPU, served by vLLM with 4096-token KV blocks, sub-block prefix caching, chunked prefill (512-token budget) and at most eight sequences per rank; a miss is a recompute, and sessions are pinned to a rank.

Cost fit. Single cold prompts and appends onto a cached prefix on the idle group (Table 17, Appendix D) give

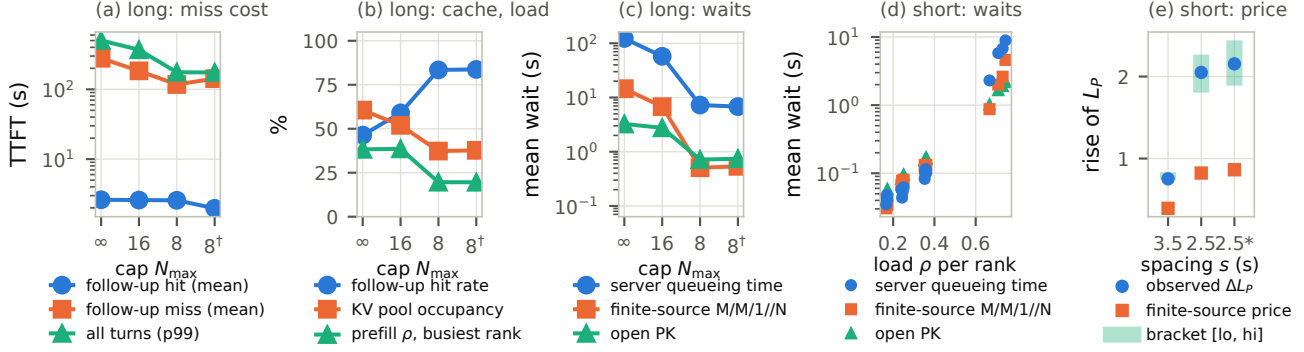


Figure 1. The testbed replays (Appendix D). Long-context replay, one session every 20 s (†: every 10 s), at most $N_{\max}$ live: (a) mean TTFT of follow-up hits and misses, p99 over all turns; (b) follow-up hit rate, KV-pool occupancy, prefill load of the busiest rank; (c) the server’s mean queueing time against the open PK wait and the finite-source wait of Proposition 2. Short-context replay: (d) per rank and run, the same three waits against the rank’s prefill load; (e) the rise of the prefill stage under forced misses against the bracket of Proposition 1 and the finite-source price (*: second seed).

$a = 0.194$ ms per token, $b = 6.51$ ns per token² and a constant of 0.044 s in (1), with a mean absolute error of 3.4 % (3.7 % when the cold probes alone predict the appends). The crossover context is $K_c = a/b \approx 30$ k tokens, below the traces’ median context of 207k: a cold 64k prompt takes 2.7 times a cold 32k prompt.

Long-context replay. Figure 1 and Table 18 replay a coding-agent trace of 37 sessions with ten turns each (median context 50k tokens, mean output 1.1k tokens, gaps capped at 30 s, mean 7 s), one session every 20 s, with the admission cap of Algorithm 1 at ∞ , 16 and 8 live sessions, and once more at 8 with a session every 10 s. A rank’s pool holds 51 allocatable blocks of 4096 tokens, two to four such contexts; the batch cap never binds.

Misses are the turns that queued. A follow-up turn whose prefix is resident has a TTFT of 2.0–2.6 s at every cap, the service time of its append. A turn whose prefix was partly evicted waits 17–80 s, one whose prefix is gone 117–279 s, 46 to 107 times longer than a hit. That ratio is not the price of a miss: 92–100 % of the misses but 2–4 % of the hits arrived to a queue on their rank, after the same think times (5.0–7.2 s and 5.9–6.6 s): admissions ahead of a waiting turn, in arrival order, evict the least recently used prefixes, possibly its own.

The wait is for KV blocks, not prefill compute. The per-rank prefill utilisation stays at 0.03–0.39, the server’s prefill time is 1.07–1.10 times the $\mathbb{E}[S]$ of the fitted cost model, and the PK wait it implies is 0.7–3.2 s. The server’s mean queueing time is 6.8–122.6 s, spent while the pool was 78–80 % full, with fewer free blocks than the 13 a 50k-token miss needs. The finite-source wait of Proposition 2, with the measured decode plus think time as Z , is 0.5–14.5 s. The queueing time is 9 to 38 times the PK wait and 8 to 14 times the finite-source wait: it is the block wait W_q^M of §2.2, which neither

prices. Running turns hold their blocks through decodes that are slower than on an idle server; the ranks step in lockstep, and some rank prefills 31–67 % of the time.

The cap decides, and it moves the wait rather than removing it. At 8 live sessions p99 TTFT falls from 503 s to 175 s while sessions wait 772 s at the entry queue; the sojourn including that wait still falls from 2583 s to 1344 s and throughput rises from 0.070 to 0.092 turns per second. Doubling the arrival rate at that cap changes only the entry wait: the pool binds before the prefill compute does, and the open regime of (3) is not reached.

Short-context replay and the price test. To reach that regime we replay a trace derived from the same sessions: nine copies with contexts scaled to 2k–9k tokens, outputs cut to four tokens so that decode is negligible, and the source’s think times (Table 19), one session every 2.5 to 5 s. The prefill queue is then the only wait. A rank serves 98–99 % of follow-ups from its cache at a load of 0.17–0.36, and its queueing time of 0.04–0.11 s is predicted within a factor of two by both forms: the PK wait is 1.3 to 1.6 times it and the finite-source wait of Proposition 2 0.7 to 1.8 times it per rank (Figure 1d); against the residual TTFT beyond the model’s service (prefill 1.09–1.17 times the model, 0.04–0.07 s outside the engine) the PK wait is 0.5–0.7. Forcing 10 % of follow-up turns to miss adds 163 to 251 misses in the window and, at the higher load, 55 more through the pool, after the 30 s gap: the feedback of §3.3. Priced together from the baseline’s λ , ρ and PK wait (Table 20, Figure 1e), the misses the change caused give a bracket containing the observed rise of L_P in all 3 tests (two loads, two seeds), 1.01 to 1.14 times its lower end (0.86 to 1.36 per rank); the head-of-line term is 55–64 % of the price, the queue’s part of the rise is 1.4 to 2.9 times the added service, and 49–59 % of the rise is borne by turns that did not miss. The finite-source price, with no

second moment, understates the rise 1.9 to 2.5 times. One session every 1.5 s saturates the replica: the batch cap of eight binds in 25 % of the samples, and 25 % of follow-ups miss, about half after the 30 s gap and most after waiting in a queue, so the misses feed the load, the instability of §3.3 on hardware.

5. Related Work

Charging a job for the delay it imposes on others is classical in the economics of queues (Naor, 1969; Mendelson & Whang, 1990); the size-squared term of (4) is the PK second moment of that externality. We claim the application (a job whose size changes with a cache decision), the two-stage separation, the finite-population bound and the one price across four decisions, not the formula. LLM queueing analyses take the variability from output length (Mitzenmacher & Shahout, 2025; Yang et al., 2024; Ozbas & Bastopcu, 2026; Shahout et al., 2024), and Ao et al. (2026) show the equilibrium of a memory-constrained replica unstable; none models KV reuse across the turns of a session, so none prices a miss; online KV scheduling has competitive policies and hardness results (Jaillet et al., 2025; Feng et al., 2026; Wang et al., 2025; Dexter et al., 2025).

Among systems that keep the KV of paused programs (Kang et al., 2026; Abhyankar et al., 2024), Continuum (Li et al., 2025) sets a time-to-live from the reload cost and the queueing delay an eviction would induce, our price without its head-of-line term and load factor; MORI (Xia et al., 2026) offloads by idleness; others predict reuse or trade a retained prefix against batch memory (Bian et al., 2025; Pan et al., 2025; Zhang et al., 2026; Ni et al., 2026); KVLearn (Liu et al., 2026a) learns retention from the inputs of u_i and θ . Chunked prefill (Agrawal et al., 2024) leaves head-of-line blocking among prefills (Hsieh et al., 2026; Liu et al., 2026b; Lyu et al., 2025), the term our price keeps.

6. Limitations

The propositions are about the model, not the world, and of the measurements of Appendix E only the cost fit and the two replays have been run (one stack, one derived workload, one forced share, four ranks, two seeds). (3) is an approximation: session feedback makes arrivals non-Poisson, and on our stack a prefill step is exclusive and the ranks step in lockstep, so decode gets what prefill leaves. The propositions concern means and a stationary fraction of misses, not tails, one eviction or block waits. It is falsified by a prefill wait off (3) with the pool slack, or a blind key beating the price.

Impact Statement

This work provides analytical tools for capacity planning and scheduling in LLM serving systems. Better utilisation of serving hardware lowers the energy and cost of deploying language models; we see no direct negative societal consequences beyond those generally associated with more efficient LLM deployment.

References

- Abhyankar, R., He, Z., Srivatsa, V., Zhang, H., and Zhang, Y. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. UNVERIFIED: authors, venue and the description of its preserve/discard/swap policy are from memory; read before submission.
- Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve, 2024. Abstract read 2026-09-23: “chunked-prefills which splits a prefill request into near equal sized chunks” and “stall-free schedules that adds new requests in a batch without pausing ongoing decodes”.
- Ao, R., Luo, G., Simchi-Levi, D., and Wang, X. Optimizing LLM inference: Fluid-guided online scheduling with memory constraints, 2025. Abstract read 2026-09-23 (v4).
- Ao, R., Dong, J., Luo, G., and Simchi-Levi, D. Service-induced congestion in memory-constrained LLM serving, 2026. Abstract read 2026-09-23 (v1).
- Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22 (2):248–260, 1975. doi: 10.1145/321879.321887. Bibliographic details verified against Crossref 2026-09-24.
- Bian, Z., Wu, F., Li, Z., Ma, T., and Zhuo, Y. TokenCake: A KV-cache-centric serving framework for LLM-based multi-agent applications, 2025. Abstract read 2026-09-23 (v4). Comments field: “Accepted at EuroSys ’27”.
- Carnes, T. and Shmoys, D. B. Primal-dual schema for capacitated covering problems. In *Integer Programming and Combinatorial Optimization (IPCO)*, pp. 288–302, 2008.
- Csirik, J., Frenk, J. B. G., Labbé, M., and Zhang, S. Heuristics for the 0-1 min-knapsack problem. *Acta Cybernetica*, 10(1–2):15–20, 1991.

- Dai, J. G., Deng, T., Li, Y., and Peng, T. Throughput-optimal scheduling algorithms for LLM inference and AI agents, 2025. Abstract read 2026-09-23 (v3).
- Dantzig, G. B. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957. doi: 10.1287/opre.5.2.266. Bibliographic details verified against Crossref 2026-09-24.
- Dexter, G., Tang, S., Fatahi Baarzi, A., Song, Q., Dharamsi, T., and Gupta, A. LLM query scheduling with prefix reuse and latency constraints, 2025. Abstract read 2026-09-23 (v3).
- Feng, Y., Yang, Z., and Zhang, Y. Competitive non-clairvoyant KV-cache scheduling for LLM inference, 2026. Abstract read 2026-09-23 (v1).
- Hsieh, C.-c., Zong, Z., Chen, X., Li, J., Zhai, J., and Wen, L. FlowPrefill: Decoupling preemption from prefill scheduling granularity to mitigate head-of-line blocking in LLM serving, 2026. Abstract read 2026-09-23 (v1), paraphrased by the fetch tool; re-read before quoting.
- Jaillet, P., Jiang, J., Mellou, K., Molinaro, M., Podimata, C., and Zhou, Z. Online scheduling for LLM inference with KV cache constraints, 2025. Abstract read 2026-09-23 (v5).
- Kang, H., Li, Z., Xu, W., Yang, X., Chen, Y., Wang, J., Chen, B., Krishna, T., Xu, C., and Arora, S. Thunder-agent: A simple, fast and program-aware agentic inference system, 2026. v3, 30 June 2026.
- Kelly, F. P. *Reversibility and Stochastic Networks*. Wiley, 1979. Bibliographic details (Wiley, 1979, 230 pp.) confirmed via Crossref-indexed reviews 2026-09-24; cited for symmetric queues with state-dependent capacity.
- Khinchine, A. Mathematical theory of a stationary queue. *Matematicheskii Sbornik*, 39(4):73–84, 1932.
- Kingman, J. F. C. *Poisson Processes*. Oxford University Press, 1993. ISBN 9780198536932. Oxford Studies in Probability 3; Campbell’s theorem is Chapter 3 (sums over Poisson processes). Bibliographic details verified against Crossref 2026-09-24.
- Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.
- Li, H., He, R., Mang, Q., Zhang, Q., Mao, H., Chen, X., Zhou, H., Zhang, H., Cheung, A., Gonzalez, J., and Stoica, I. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live, 2025. Abstract read 2026-09-23 (v7); listed as an ICLR 2026 workshop poster, cite as arXiv.
- Liu, D., Yu, Y., Jiang, E., Wang, S., and Wu, Y. N. To keep or not to keep: Learning KV cache retention in disaggregated LLM serving systems. In *Proceedings of the ACM International Systems and Storage Conference (SYSTOR)*, 2026a. doi: 10.1145/3793230.3837769. Project README read 2026-09-23 (github.com/FastLM/KVLearn); paper text not read. README describes a reuse predictor, a “Cost-Aware Retention Score” weighing “recompute, transfer, and storage costs” and an “Adaptive Threshold Controller” driven by “closed-loop feedback from observed hit rates and memory pressure”; TTFT reduced “by up to 56% vs. No-Cache”.
- Liu, H., Wang, J., Xu, Y., and Li, R. Fairness-aware and latency-controllable scheduling for chunked-prefill LLM serving, 2026b. Abstract read 2026-09-23 (v1).
- Lyu, H., Liu, B., Wu, M., and Chen, H. FairBatching: Fairness-aware batch formation for LLM inference, 2025. Abstract read 2026-09-23 (v1).
- Mendelson, H. and Whang, S. Optimal incentive-compatible priority pricing for the M/M/1 queue. *Operations Research*, 38(5):870–883, 1990. doi: 10.1287/opre.38.5.870. Bibliographic details verified against Crossref 2026-09-24; cited for priority pricing at the M/M/1 queue only.
- Mitzenmacher, M. and Shahout, R. Queueing, predictions, and LLMs: Challenges and open problems, 2025. Abstract read 2026-09-23 (v1).
- Naor, P. The regulation of queue size by levying tolls. *Econometrica*, 37(1):15–24, 1969. doi: 10.2307/1909200. Bibliographic details verified against Crossref 2026-09-24.
- Ni, H., Tian, H., Zhang, C., Li, G., and Tan, H. TOPAS: Workflow-aware prefix-state scheduling for multi-agent LLM serving, 2026. Abstract read 2026-09-23 (v1).
- Nie, C., Si, N., and Zhou, Z. A queueing-theoretic framework for stability analysis of LLM inference with KV cache memory constraints, 2026. Abstract read 2026-09-23 (v1). Comments field: “Accepted in ICML 2026”.
- Ozbas, E. and Bastopcu, M. Queueing-aware optimization of reasoning tokens for accuracy-latency trade-offs in LLM servers, 2026. Abstract read 2026-09-23 (v1).
- Pan, Z., Patel, A., Hu, Z., Shen, Y., Guan, Y., Li, W.-L., Qin, L., Wang, Y., and Ding, Y. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows, 2025. Abstract read 2026-09-23 (v1). UN-VERIFIED: NeurIPS 2025 venue seen only in a search hit.

Pollaczek, F. Über eine aufgabe der wahrscheinlichkeitstheorie. *Mathematische Zeitschrift*, 32:64–100, 1930.

Qiao, Y., Le, T. D., Shen, A., Li, Z., and Wang, B. Serving agentic workloads at scale with vLLM x Mooncake. vLLM Blog, <https://vllm.ai/blog/2026-05-06-mooncake-store>, May 2026.

SemiAnalysis. CC traces – Weka, with subagents, v7 only (Jun 13 2026). Hugging Face dataset, <https://huggingface.co/datasets/semianalysisai/cc-traces-weka-061326>, June 2026. WekaTrace corpus of 183 Claude Code proxy sessions (26,648 main-agent requests, 652 sub-agent groups); per-request prefix-block hashes, input/output tokens, api_time, ttft, think_time. Verified 2026-09-23 against the dataset README and the file itself; statistics in paper/traces/ are computed by scripts/trace_stats_weka.py.

Shahout, R., Malach, E., Liu, C., Jiang, W., Yu, M., and Mitzenmacher, M. Don’t stop me now: Embedding based scheduling for LLMs, 2024. Abstract read 2026-09-23 (v1). UNVERIFIED: ICLR 2025 venue seen only in a search hit.

vLLM Team and Inferact. vLLM x AgentX: Optimizing for real-world agentic serving. vLLM Blog, <https://vllm.ai/blog/2026-09-08-vllm-agentx>, September 2026.

Wang, M., Ye, Y., and Zhou, Z. LLM serving optimization with variable prefill and decode lengths, 2025. Abstract read 2026-09-23 (v4).

Xia, T., Li, H., Li, Z., Chen, X., Kang, H., Qiao, Y., Xu, Y., and Stoica, I. Idleness is relative: Exploiting tool-call idle windows for offloading in agentic systems with MORI, 2026. Abstract read 2026-09-23 (v1).

Yang, Y., Xu, Y., and Jiao, L. A queueing theoretic perspective on low-latency LLM inference with variable token length, 2024. Abstract read 2026-09-23 (v2). UNVERIFIED: conference publication seen only in a search hit.

Zhang, R., Kim, C., Feng, S., Du, K., Liu, Y., Zhong, Y., Ching, C.-W., Jiang, J., and Hu, L. Learning agent execution for KV-cache management in agentic serving, 2026. Abstract read 2026-09-23 (v1).

A. Proofs

Every proposition and every derived number in the paper has also been formalised and machine-checked in Lean 4 with Mathlib; the development is in the supplementary material. The proofs below are written for human readers and

do not follow the formal proofs step by step. The known results quoted in §3.1 (the threshold rule, the block prefix and the guarded greedy) and the closed form (8) are verified there as well and not re-proved here. For Proposition 2 the mean value recursion is taken as the model, as the arrival theorem gives it; the inequalities are proved from it.

Proof of Proposition 3. (i) Write $a_m = 1/\prod_{k \leq m} \varphi(k)$, so $L(\rho) = \sum_m m a_m \rho^m / \sum_m a_m \rho^m$. For $\rho_1 \leq \rho_2$,

$$L(\rho_2) - L(\rho_1) = \frac{\sum_{m,j} m a_m a_j (\rho_2^m \rho_1^j - \rho_1^m \rho_2^j)}{P(\rho_1)P(\rho_2)},$$

with $P(\rho) = \sum_m a_m \rho^m$. Exchanging m and j and averaging, the numerator is half of $\sum_{m,j} a_m a_j (m-j)(\rho_2^m \rho_1^j - \rho_1^m \rho_2^j)$. For $m \geq j$ the last factor is $\rho_1^j \rho_2^j (\rho_2^{m-j} - \rho_1^{m-j}) \geq 0$, and symmetrically for $m < j$ both factors are non-positive, so every term is non-negative. This holds for every truncation of the sums at a finite M , and L is the limit of the truncated means, so it holds for L . By insensitivity, L depends on the policy only through ρ , and a policy that adds expected demand A per arrival moves the load to $\rho + \lambda A$.

(ii) With constant capacity π is geometric and $L = \rho/(C - \rho)$. With $\rho' = \rho + \lambda A$ and $A = \sum_i q_i \Delta D_i$,

$$L(\rho') - L(\rho) = \frac{C \lambda A}{(C - \rho)(C - \rho')} = \frac{C - \rho}{C - \rho'} \cdot \frac{C \lambda A}{(C - \rho)^2},$$

and the last factor is $\lambda \sum_i q_i \Phi_i^D$. Since $\rho' \geq \rho$ the prefactor is at least one. $\square$

Proof of Proposition 2. Write $c = \mu Z$ and $g(q) = (1 + q)/(c + 1 + q)$, so $Q_n = n g(Q_{n-1})$. The recursion is mean value analysis for the closed network of one FIFO server and one delay station: by the arrival theorem a turn arriving with n sessions in the system sees the stationary state with $n-1$, so its response time is $R_n = (1 + Q_{n-1})/\mu$, the throughput is $X_n = n/(Z + R_n)$ and $Q_n = X_n R_n$. Dividing, $\rho_n = X_n/\mu = n/(c + 1 + Q_{n-1})$ and $Q_n = \rho_n(1 + Q_{n-1})$.

(i) g is nonnegative and nondecreasing on $q \geq 0$, and $g \leq 1$. Hence $0 \leq Q_n \leq n$ by induction, and $Q_{n+1} = (n+1)g(Q_n) \geq n g(Q_{n-1}) = Q_n$ once $Q_n \geq Q_{n-1}$, which holds at $n = 1$.

$\rho_n < 1$: it is equivalent to $Q_{n-1} > n - 1 - c$, and $Q_n > n - c$ holds for all n by induction, since $Q_{n+1} - (n+1 - c) = c(Q_n + c - n)/(c + 1 + Q_n) > 0$.

Q_n is nonincreasing in c : g is nonincreasing in c at fixed q , so if $Q_{n-1}(c') \leq Q_{n-1}(c)$ for $c \leq c'$ then $Q_n(c') = n g_{c'}(Q_{n-1}(c')) \leq n g_c(Q_{n-1}(c)) \leq n g_c(Q_{n-1}(c)) = Q_n(c)$.

(ii) $W_N \leq \rho_N/(\mu(1 - \rho_N))$ is $Q_{N-1}(1 - \rho_N) \leq \rho_N$, that is $Q_{N-1} \leq \rho_N/(1 - \rho_N) = \rho_N$, which is (i).

The cap stated after the proposition: the mean number at the server of a closed system of N sessions lies in $[0, N]$ under any policy, so $L'_P - L_P \leq N - L_P$. $\square$

Proof of Proposition 1 and of the remarks after it.

Write $u = 1 - \rho$, $m_2 = \mathbb{E}[S^2]$, $A = \sum_i q_i \Delta S_i$ and $B = \sum_i q_i ((S_i^{\text{miss}})^2 - (S_i^{\text{hit}})^2)$. The change raises $\mathbb{E}[S]$ by A and $\mathbb{E}[S^2]$ by B , so the load becomes $\rho' = \rho + \lambda A$ and $1 - \rho' = u - \lambda A > 0$. By (3), $L = \lambda^2 m_2 / (2u) + \rho$, hence

$$\Delta L = \lambda A + \frac{\lambda^2}{2} \left(\frac{m_2 + B}{u - \lambda A} - \frac{m_2}{u} \right) = \lambda A + \frac{Y}{u(u - \lambda A)},$$

with $Y = \lambda^2(uB + \lambda A m_2)/2 \geq 0$. Summing (4) with $W = \lambda m_2 / (2u)$ gives $\lambda \sum_i q_i \Phi_i = \lambda A + Y/u^2$.

For the bracket, since $0 < u - \lambda A \leq u$, $Y/(u(u - \lambda A)) \geq Y/u^2$, which is the lower bound. For the upper bound,

$$\frac{u}{u - \lambda A} \left(\lambda A + \frac{Y}{u^2} \right) - \Delta L = \frac{(\lambda A)^2}{u - \lambda A} \geq 0.$$

For the ratios stated after the proposition, $(S^{\text{miss}})^2 - (S^{\text{hit}})^2 = (S^{\text{miss}} + S^{\text{hit}})\Delta S$, so the second term of Φ_i is $\lambda(S^{\text{miss}} + S^{\text{hit}})/(2u)$ times ΔS_i , and the third is $\lambda W \Delta S_i / u$. Dividing gives both ratios.

For the divergence, let $m_1 = \mathbb{E}[S]$ and $\lambda = (1 - \varepsilon)/m_1$ with $0 < \varepsilon \leq 1/2$, so $\rho = 1 - \varepsilon$ and $\lambda \geq 1/(2m_1)$. The first two terms of Φ_i are non-negative, and the third is $\lambda^2 m_2 \Delta S_i / (2\varepsilon^2) \geq m_2 \Delta S_i / (8m_1^2 \varepsilon^2)$, which exceeds any bound for small ε . $\square$

Proof of Proposition 4. (i) Take two programs of the same size $c > 0$ that agree in everything the rule observes, and a target $\Delta C = c$, so that either program alone is feasible. The rule sees the same input whatever the resume probabilities are, so it evicts a fixed one of the two, say e , and let o be the other. Set $p_e = 1$ and $p_o = 1/(|R| + 2)$, both in $(0, 1]$. The evicted program costs $p_e c^2 = c^2$, while o costs $c^2/(|R| + 2)$, and $R \cdot c^2/(|R| + 2) < c^2$ since $R < |R| + 2$. The optimum is at most the cost of o , so the rule's ratio exceeds R .

(ii) By (1) the recompute work of a context of c tokens is $ac + bc^2/2$; the proposition takes the quadratic term as the price, $w_i = c_i^2$, so $w_i/c_i = c_i$ and price-per-byte order is shortest-first (SF). SF frees ΔC whenever the pool holds it: it stops only at a program at least as long as the remaining target or when the pool is exhausted. For the ratio, let SF take $t_1 \leq \dots \leq t_m$, the first m programs in sorted order. Write $\ell = t_m$ and $\sigma = t_1 + \dots + t_{m-1}$, so $\sigma < \Delta C$ (SF did not stop earlier). Let S be any feasible set.

Step 1: $\ell^2 \leq \text{cost}(S)$. Every program with $c_i < \ell$ is among the first $m - 1$ in sorted order, and these free only $\sigma < \Delta C$. So S contains a program with $c_i \geq \ell$, whose cost alone is at least ℓ^2 .

Step 2: a lower bound on $\text{cost}(S)$. Let $f(x) = x(\ell - x)$ for $x < \ell$ and $f(x) = 0$ otherwise, so $f \geq 0$ on $[0, \infty)$ and $x^2 \geq \ell x - f(x)$ for all $x \geq 0$ (with equality for $x < \ell$, and $x^2 \geq \ell x$ for $x \geq \ell$). Summing over S and using $\sum_{i \in S} c_i \geq \Delta C$,

$$\text{cost}(S) \geq \ell \Delta C - \sum_{i \in S} f(c_i) \geq \ell \Delta C - \sum_{j < m} t_j(\ell - t_j),$$

because only programs with $c_i < \ell$ contribute to the sum, all of them are among $t_1, \dots, t_{m-1}$, and $t_j(\ell - t_j) \geq 0$ for $j < m$. The right side equals $\ell(\Delta C - \sigma) + \sum_{j < m} t_j^2$.

Step 3. Since $\Delta C - \sigma > 0$,

$$\begin{aligned} \text{cost}(\text{SF}) &= \sum_{j < m} t_j^2 + \ell^2 \\ &\leq \text{cost}(S) - \ell(\Delta C - \sigma) + \ell^2 \\ &\leq \text{cost}(S) + \ell^2, \end{aligned}$$

and Step 1 bounds the last term by $\text{cost}(S)$.

Tightness: on $K, K + 1$ with $\Delta C = K + 1$, SF takes K (leaving 1), then $K + 1$, at cost $K^2 + (K + 1)^2$, while $\{K + 1\}$ costs $(K + 1)^2$. $\square$

Block eviction (§3.1). The proof of (i) uses only that the two candidates cost the same positive amount before the resume probabilities weigh them, so it applies unchanged to the tail blocks of two programs that agree in everything but p_i . For the tail of one context, let $f(m) = am + bm(K - m/2)$ be the recompute of its last m tokens. Then $f(m+d) - f(m) = ad + bd(K - m - d/2)$ is nonincreasing in m for $b, d \geq 0$, so deeper blocks cost no more per byte, and $f(m_1 + m_2) - f(m_1) - f(m_2) = -bm_1 m_2 \leq 0$.

B. Simulation Tables

The tables below are the ones §4.2 summarises. The validation package runs serQ programs and turns their results into the tables below. Its checks of the propositions run the queues the propositions are stated for (M/G/1, PS); everything that involves the replica's memory runs the vLLM v1 engine's rules, below. The synthetic checks use $a = 2 \cdot 10^{-5}$, $b = 2 \cdot 10^{-9}$, $\omega = 2 \cdot 10^{-4}$ and $\beta = 2 \cdot 10^{-9}$ seconds per token unless a caption says otherwise. Where a miss is paid (rows 8–9 of Table 4), the eviction and admission experiment (Tables 7 and 8, Figure 4) and the replays of production sessions (Tables 12, 11 and 13) use the cost model calibrated on the testbed (§4.3) and run the replica as programs of serQ, a language for serving deployments

whose interpreter reproduces the upstream vLLM v1 scheduler request for request (first-token and last-token steps, preemptions and cached tokens) on recorded scenarios and on a full trace replay. Every number is generated by the simulator from fixed seeds and is a property of the simulated model, not a measurement; the best value in each comparison is in bold.

Table 3. Simulation under each proposition’s assumptions (in-model). M/M/1: $\mu = 10$. PK: $\rho = 0.7$. Cache: $\lambda = 1.8$. ΔL (the prefill queue as an M/G/1 FIFO server): rise in the mean number in system when a fraction δ of turns changes from hit (0.05 s) to miss (0.5 s) at hit rate 0.8, $\rho = 0.6$, against the bracket $[\lambda\delta\Phi, \frac{1-\rho}{1-\rho'}\lambda\delta\Phi]$. SF/OPT: worst of the 9000 instances of the first three rows of Table 6. $\pm$ is a 95 % batch-means half-width. Times in seconds.

Quantity	Result	Theory	Simulated
M/M/1 W , $\lambda = 9$	§1	1.000	1.001 ± 0.034
PK W_q , workload B	Eq. (3)	4.002	3.989 ± 0.059
W_q^B / W_q^A , $\rho = 0.6$	§2.3	3.43	3.38
ρ at $h = 0.8$	§2.3	0.252	0.252
$W_q / M/M/1$, $CV^2=15.3$	§2.3	8.15	8.12
ΔL , $\delta = 0.05$	Prop. 1	[0.67, 0.88]	0.849 ± 0.009
max SF/OPT	Prop. 4	≤ 2	1.791

Table 4. Batching server under its assumptions (in-model): mean number of turns at the replica, $L = \lambda R$. Rows 1–4: Poisson turns at $\rho = 0.7$, work deterministic or hit/miss (0.05 s w.p. 0.8, else 0.5 s; both mean 0.14 s); PS theory $\rho/(1-\rho)$, FIFO theory $\lambda W_q + \rho$. Rows 5–6: sessions arrive Poisson, each turn is followed by a tool call (mean 2 s, $CV^2 = 4$) and another turn w.p. 0.8; theory $\sum_n n\pi(n)$ at $\lambda = \Lambda/(1-p)$ ($\rho = 0.7$ for $\phi \equiv 1$, $\rho = 3$ for $\phi(n) = n/(1+\beta(n-1))$). Row 7: rise of L at a PS server when a fraction δ of turns becomes misses, hit rate 0.8, $\rho = 0.6$, against $[\lambda\delta\Phi_{PS}, \frac{C-\rho}{C-\rho'}\lambda\delta\Phi_{PS}]$. Rows 8–9 (beyond the model): the same misses on a replica with the vLLM v1 engine’s rules and the testbed’s cost model (prefill of 512 or 5120 tokens in the budget a decode batch of 8 tokens per turn leaves, mean availability $\bar{r} = 0.997$, $\lambda\mathbb{E}[P] = 0.6$): rise of the number in the prefill stage $L_P = \lambda TFFT$ against the FIFO bracket at service $P/\bar{r}$, and of the number in the decode stage L_D , whose demand does not depend on hit or miss but whose iterations a miss’s prefill lengthens. $\pm$ is a 95 % batch-means half-width.

Quantity	Result	Theory	Simulated
PS, D	§2.2	2.333	2.338 ± 0.021
PS, hit/miss	§2.2	2.333	2.315 ± 0.033
FIFO, D	Eq. (3)	1.517	1.519 ± 0.011
FIFO, hit/miss	Eq. (3)	2.867	2.844 ± 0.039
Loop, $\phi \equiv 1$	§2.1	2.33	2.40 ± 0.08
Loop, sat. $\phi(\beta=0.25)$	§2.1	12.00	12.02 ± 0.33
PS ΔL , $\delta=0.05$	Prop. 3	[0.60, 0.79]	0.787 ± 0.021
vLLM ΔL_P , $\delta=0.05$	Prop. 1	[0.70, 0.94]	0.896 ± 0.027
vLLM ΔL_D , $\delta=0.05$	Prop. 3	0	0.002 ± 0.000

Table 5. Batch cap $B = 8$: exact limited PS (at most B turns share $\phi(n) = n/(1 + 0.1(n-1))$, FIFO beyond) against PS with ϕ flattened at B , the model’s approximation. Poisson turns, work of mean 1 with the given law (CV^2 in brackets), load $u = \rho/\phi(B)$. Mean number at the replica; last column: relative error of the approximation for LPS, in %. Mean response time has the same relative error (Little).

Work	u	Formula	PS	LPS	Err.
D (0)	0.5	3.10	3.10 ± 0.00	3.08 ± 0.00	−1
D (0)	0.8	7.33	7.33 ± 0.06	6.60 ± 0.03	−10
D (0)	0.9	12.67	12.81 ± 0.44	9.90 ± 0.22	−22
Exp (1)	0.5	3.10	3.09 ± 0.02	3.10 ± 0.02	−0
Exp (1)	0.8	7.33	7.36 ± 0.12	7.37 ± 0.12	+1
Exp (1)	0.9	12.67	12.06 ± 0.44	12.06 ± 0.45	−5
H2 (4)	0.5	3.10	3.09 ± 0.03	3.11 ± 0.04	+0
H2 (4)	0.8	7.33	7.33 ± 0.17	9.11 ± 0.38	+24
H2 (4)	0.9	12.67	12.58 ± 0.85	20.57 ± 2.14	+62

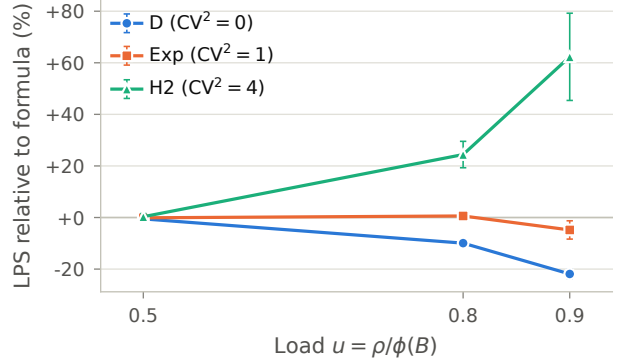


Figure 2. Error of the saturating- ϕ formula against an exact batch cap ($B = 8$) as a function of load, one line per work law. The sign follows the work CV^2 . Synthetic, uncalibrated.

Table 6. Offline eviction: cost relative to the exact optimum of (6), mean / max over 3000 random instances per row (12 programs, $c_i \in \{1, \dots, 200\}$, ΔC a random 10–60 % of the total); the lowest mean and the lowest max per row in bold. First three rows: costs $p_i c_i^2$. Last row: weights w_i log-uniform on $[10^{-4}, 10^5]$, independent of c_i ; shortest-first does not see w_i there, and its ratio is the price of ignoring the weights. Guarded: the guarded density greedy of §3.1.

Costs	SF	Density	Guarded
equal p_i	1.108 / 1.67	1.108 / 1.67	1.038 / 1.36
$p_i \sim U[0.05, 1]$	1.902 / 10.05	1.136 / 2.86	1.032 / 1.34
p_i falls with c_i	2.037 / 8.60	1.146 / 2.67	1.030 / 1.87
arbitrary w_i	$6.7 \times 10^6 / 1.0 \times 10^9$	1.091 / 6.71	1.005 / 1.27

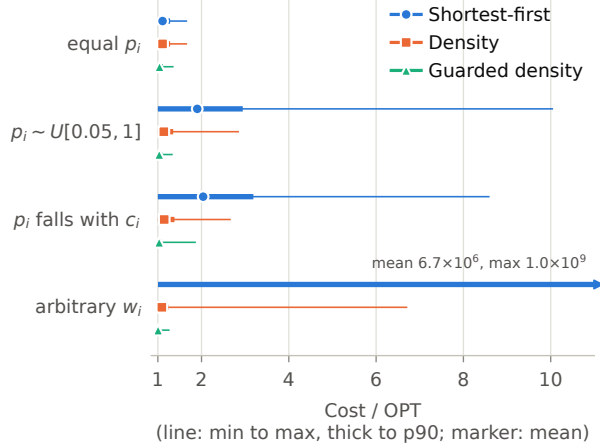


Figure 3. Offline eviction cost relative to the exact optimum by rule and regime (mean, with the worst instance marked). Synthetic, uncalibrated.

Table 7. Eviction with open sessions on a replica with the vLLM v1 engine’s rules (those of Table 12) and the testbed’s cost model. Sessions arrive Poisson at rate Λ (per s): half agents ($p_i = 0.95$, 4–8k initial tokens, tool calls Exp(6 s)), half one-shot documents ($p_i = 0.3$, 15–30k tokens, think time Exp(2 s)); 600k-token KV, batch cap 8, at most 24 live sessions. Every policy evicts 16-token blocks from the tail of a context, sessions in a tool call before waiting ones (except LRU), and drops a finished session’s blocks; rows marked * keep them, as vLLM’s engine does, which cannot tell a finished session from one in a tool call. Throughput (turns/s), follow-up hit rate (the whole reusable prefix reused), mean and p99 time to first token (s), mean turn response (s; its half-width is in the data file), and the number of the 20 seeds that thrashed (follow-up turns reusing less than half of their reusable prefix); $\pm$ is a 95 % half-width over seeds; the best throughput, hit rate, TTFT and p99 per load in bold. Priced: $p_i\Phi_i/c_i$ with the FIFO price of Prop. 1 from online estimates at the engine; Priced/ τ : per byte-second, $p_i\Phi_i/(c_i\tau_i)$ (the threshold rule of §3.1); Blocks: the same price of the tail block. The priced rules take p_i and τ_i from the generating class (an oracle); the fixed keys see neither.

Λ	Policy	X	Hit	TTFT	p99	R	Thr.
0.03	SF	0.31	0.97	7.01 $\pm$ 3.71	35.6	25.5	0
0.03	Density	0.31	0.97	7.02 $\pm$ 3.72	35.6	25.5	0
0.03	Priced	0.31	0.97	7.01 $\pm$ 3.72	35.7	25.5	0
0.03	Priced/ τ	0.31	0.97	7.01 $\pm$ 3.72	35.9	25.5	0
0.03	Blocks	0.31	0.97	7.01 $\pm$ 3.72	35.7	25.5	0
0.03	LRU	0.29	0.96	11.80 $\pm$ 8.35	68.6	30.5	0
0.03	LRU*	0.29	0.94	14.13 $\pm$ 10.60	69.9	33.1	0
0.03	Priced/ τ *	0.30	0.54	16.83 $\pm$ 5.98	59.5	36.6	0
0.05	SF	0.35	0.75	40.89 $\pm$ 2.95	95.2	60.9	0
0.05	Density	0.35	0.75	40.92 $\pm$ 2.94	96.3	60.9	0
0.05	Priced	0.35	0.75	40.93 $\pm$ 2.97	94.5	60.9	0
0.05	Priced/ τ	0.35	0.75	40.89 $\pm$ 2.94	96.2	60.9	0
0.05	Blocks	0.35	0.75	40.91 $\pm$ 2.95	94.8	60.9	0
0.05	LRU	0.24	0.69	75.45 $\pm$ 12.56	255.8	97.3	2
0.05	LRU*	0.21	0.57	91.24 $\pm$ 12.98	256.2	114.0	6
0.05	Priced/ τ *	0.32	0.42	46.31 $\pm$ 2.43	96.1	67.1	0

Table 8. Admission-cap sweep on the vLLM-rule replica: the scenario of Table 7 with at most 16, 24 or 32 live sessions; later arrivals wait in an entry queue. Throughput (turns/s), follow-up hit rate, mean and p99 time to first token (s), mean turn response (s), mean entry-queue wait of a new session (s), and the number of the 20 seeds that thrashed. Means over seeds; the best throughput, hit rate, TTFT and p99 per (Λ , cap) in bold. The 95 % half-widths over seeds are the whiskers of Figure 4; Table 7 states them for the rows it shares. An entry wait of hundreds of seconds means the replica does not carry the offered turn rate and the entry queue grows over the run; such values depend on the horizon (20 000 s). Every policy drops a finished session’s blocks.

Λ	Cap	Policy	X	Hit	TTFT	p99	R	Entry	Thr.
0.03	16	SF	0.31	1.00	4.42	23.2	22.81	18.2	0
0.03	16	Density	0.31	1.00	4.42	23.2	22.81	18.2	0
0.03	16	Priced/ τ	0.31	1.00	4.42	23.2	22.81	18.2	0
0.05	16	SF	0.40	0.99	14.99	35.3	33.72	2266.6	0
0.05	16	Density	0.40	0.99	14.98	35.3	33.72	2267.0	0
0.05	16	Priced/ τ	0.40	0.99	14.99	35.3	33.72	2266.6	0
0.03	24	SF	0.31	0.97	7.01	35.6	25.53	21.2	0
0.03	24	Density	0.31	0.97	7.02	35.6	25.54	20.9	0
0.03	24	Priced/ τ	0.31	0.97	7.01	35.9	25.53	21.4	0
0.05	24	SF	0.35	0.75	40.89	95.2	60.86	2590.1	0
0.05	24	Density	0.35	0.75	40.92	96.3	60.91	2589.9	0
0.05	24	Priced/ τ	0.35	0.75	40.89	96.2	60.89	2591.3	0
0.03	32	SF	0.31	0.96	9.06	40.9	27.65	27.2	0
0.03	32	Density	0.31	0.96	9.06	41.0	27.64	26.3	0
0.03	32	Priced/ τ	0.31	0.96	9.06	40.5	27.65	27.1	0
0.05	32	SF	0.29	0.45	79.12	168.8	100.79	2953.7	3
0.05	32	Density	0.29	0.44	79.30	169.7	100.99	2959.3	3
0.05	32	Priced/ τ	0.29	0.45	79.04	168.2	100.71	2950.4	3

Table 9. Placement over 4 replicas with half of the new programs placed on replica 0 (the affinity node), follow-up turns of 5–15k-token contexts, state moved over one shared link of bandwidth B (tokens/s) when that is cheaper than recomputing it. x : mean move cost over mean service; $\rho^* = x/(1+x)$, the inversion utilisation of Eq. (8) for one node and an idle alternative; Inv.: the lowest program rate (per s, swept in steps of 0.05) at which always moving to the least-loaded replica beats affinity in mean turn response beyond both confidence intervals; ρ_0 : utilisation of the affinity node at that rate; ρ_L : utilisation of the shared link under always-move there; mean turn response (s) of affinity, always-move and the lookahead rule of Eq. (7). The idle alternative gives the earliest inversion, so ρ^* is a lower bound for any real alternative.

B	x	ρ^*	Inv.	ρ_0	ρ_L	Affinity	Move	Lookahead
20000k	0.01	0.01	≤ 0.20	0.13	0.00	0.081	0.072	0.072
2000k	0.08	0.08	≤ 0.20	0.13	0.00	0.081	0.074	0.072
500k	0.33	0.25	0.40	0.25	0.05	0.095	0.088	0.075
125k	1.31	0.57	1.60	0.99	0.96	2.878	0.961	0.120

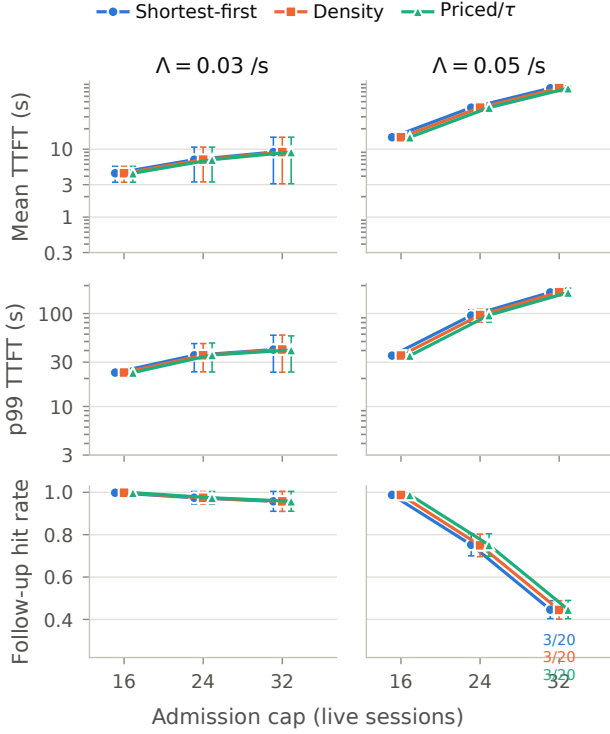


Figure 4. Admission-cap sweep on the vLLM-rule replica with the testbed’s cost model: mean and p99 time to first token (log scale) and hit rate by cap and eviction rule at two session arrival rates; annotations give the number of seeds (of 20) that thrashed. Synthetic workload.

Table 10. A FIFO prefill queue fed by N live sessions, each thinking for an exponential time of mean Z (s) and then submitting a turn of exponential work with mean 1 s, with Z chosen so that the server utilisation is $\rho = 0.6$ for every N . Mean wait in queue (s): exact finite-source value (M/M/1// N), simulated (95 % batch-means half-width), and the open M/M/1 wait $\rho/(\mu(1 - \rho))$ that the PK formula gives from the measured rate and moments; last column their ratio.

N	Z	Exact	Simulated	Open	Open/Exact
2	2.0	0.333	0.332 ± 0.007	1.500	4.50
4	5.0	0.646	0.649 ± 0.005	1.500	2.32
8	11.4	0.914	0.922 ± 0.010	1.500	1.64
16	24.5	1.126	1.129 ± 0.005	1.500	1.33
64	104.3	1.375	1.370 ± 0.009	1.500	1.09

Table 11. Forced misses on the replayed production sessions with no pool limit (open, Poisson sessions at rate Λ per s, at most 24 live, 60000 s of measurement after 6000 s of warm-up: the configuration of the first rows of Table 12). A share δ of follow-up turns is forced to miss: the turn reuses nothing, as a nonce at the head of its prompt would make it, and its old blocks stay cached. $\bar{N}$, $\bar{N}'$: mean live sessions in the baseline and in the forced run; ρ : prefill load of the baseline; ρ' : the load the added work implies; L_P : baseline mean number in the prefill stage; ΔL_P : its rise in the forced run, mean and 95 % half-width over 20 seeds; [lo, hi]: the bracket of Proposition 1 from the baseline’s measured λ , ρ , W and each forced turn’s own S^{hit} , S^{miss} ($\rho' \geq 1$: the open queue would be unstable); Finite: the rise of Proposition 2’s recursion for the added mean work, with $N = \bar{N}$ rounded and the corpus mean think time; Fin.: that finite-source rise; cap: $\bar{N}' - L_P$.

Λ	δ	$\bar{N}$	$\bar{N}'$	ρ	ρ'	L_P	ΔL_P	lo	hi	Fin.	cap
0.0005	0.01	2.2	2.5	0.32	0.39	0.7	0.2 ± 0.1	2.0	2.3	0.1	1.7
0.0005	0.03	2.2	3.2	0.32	0.52	0.7	0.8 ± 0.3	5.3	7.5	0.2	2.5
0.0005	0.10	2.2	6.1	0.32	0.77	0.7	3.3 ± 1.3	10.8	32.2	0.4	5.4
0.001	0.01	8.8	10.5	0.62	0.72	5.0	1.7 ± 0.9	19.5	27.3	0.1	5.5
0.001	0.03	8.8	12.6	0.62	0.88	5.0	3.8 ± 1.0	45.2	149.3	0.3	7.6
0.001	0.10	8.8	16.3	0.62	1.11	5.0	7.5 ± 1.1	90.4	$\rho' \geq 1$	0.4	11.2

Table 12. Replayed production sessions on a replica with the vLLM v1 engine’s rules: 761 Claude Code sessions (26395 turns, mean final context 391k tokens, mean think time 23 s; a gap above 10 min starts a new session) from the corpus of §4.1, each played turn by turn. The engine admits a waiting turn at the start of an iteration with the token budget its residents leave, once its whole prompt fits, and grows its KV chunk by chunk (a growth that does not fit preempts the latest admitted turn); KV is cached in 16-token blocks, evicted from the tail of a context by price per byte-second where the pool is finite; a finished session’s blocks stay. Cost model with $K_c = a/b \approx 30$ k tokens (calibrated on the testbed), batch cap 8. With no pool limit the replica is open (Poisson sessions at rate Λ per s, at most 24 live; the regime column gives Λ). With a finite pool the cap on live sessions binds throughout the run, so the replica is a closed system of N sessions and the arrival rate is immaterial; the entry queue grows for the whole horizon. Mean live sessions $\bar{N}$ and throughput X (turns/s); follow-up hit rate (the whole reusable prefix reused) and the mean share of the reusable prefix reused (Reuse); the share of $\text{Var}[S]$ of follow-up prefill work due to the hit/miss mixture (the rest is the spread of the appends and of partial misses) and the CV^2 of that work; prefill load ρ in stage time; observed mean prefill wait W_q and the PK wait from the measured λ , $\mathbb{E}[S]$, $\mathbb{E}[S^2]$ (s); mean TTFT (s). Means over 20 seeds; the half-widths are in the data file.

Pool	Regime	$\bar{N}$	X	Hit	Reuse	Mix	CV^2	ρ	W_q	PK	TTFT
∞	$\Lambda=0.0005$	2.2	0.012	1.00	1.00	0.00	27.0	0.32	35	299	63
∞	$\Lambda=0.001$	8.8	0.024	1.00	1.00	0.00	38.4	0.62	182	1267	208
4M	$N=8$	6.1	0.019	0.72	0.96	0.06	15.1	0.69	140	2011	186
4M	$N=16$	9.4	0.015	0.65	0.83	0.10	13.0	0.73	872	inf	964
2M	$N=4$	3.7	0.012	0.60	0.92	0.10	11.4	0.70	259	3754	397
2M	$N=8$	7.1	0.006	0.41	0.62	0.16	6.9	0.89	1224	inf	1465
1M	$N=2$	2.0	0.007	0.67	0.94	0.22	12.3	0.63	131	1464	268
1M	$N=4$	3.9	0.004	0.26	0.51	0.14	2.6	0.90	851	6657	1182

Table 13. Sensitivity of the open-pool replay ($\Lambda = 0.0005$, no pool limit) to the rule that starts a new session at a gap longer than the threshold: sessions in the corpus, mean turns and mean think time (s) per session, then mean live sessions, throughput (turns/s), follow-up prefill-work CV^2 , load, observed and PK prefill wait (s) and mean TTFT (s).

Split	Sess.	Turns	Think	$\bar{N}$	X	CV^2	ρ	W_q	PK	TTFT
10 min	761	34.7	23	2.2	0.012	27.0	0.32	35	299	63
30 min	451	58.6	36	3.3	0.022	22.2	0.29	16	164	30

Table 14. Production Claude Code traces (SemiAnalysis, 2026): main-agent requests from the second turn on. Think time is the gap before a request (tool plus user).

	median	mean	p90	CV^2
turns per session	58	144	289	4.53
context K (tokens)	207 040	275 135	600 704	0.60
appended tokens n	1 344	3 413	5 056	32.01
output tokens o	501	1 132	2 795	2.75
think time (s)	3.8	214.4	88.2	304.36
TTFT (s)	3.54	4.64	8.24	0.95

C. Trace Tables

Tables 14, 15 and 16 give the workload statistics of the production Claude Code traces that §4.1 summarises: the per-request workload, the proxy’s TTFT against the append and the context, and the resume probabilities and pause times behind the eviction key.

D. Testbed Tables

Table 17 and Figure 5 give the probes behind the cost-model fit of §4.3; every request was sent alone on an idle replica group, and append probes count only when the server’s cached prefix length equals K . Table 18 holds the numbers behind Figure 1; with the fitted cost model the hit/miss mixture supplies 58–81 % of the variance of follow-up prefill work in those runs, the measured counterpart of the split of Table 2. Tables 19 and 20 hold the numbers behind Figure 1(d,e). The short-context trace keeps the source’s appends scaled by a constant, its output lengths capped and its think times; every session carries a nonce so that no two share a prefix, and a forced miss puts a second nonce at the head of that turn’s prompt and of every later prompt of the session, so the turn recomputes its context once and hits again afterwards.

E. Experimental Design for a Real System

Table 21 lists the experiments that will decide whether the propositions carry over to a rack-scale NPU testbed running a serving stack with prefix caching, KV offloading and a pluggable router, on replayed coding-agent traces and a synthetic chat workload. Decision accuracy is the

Table 15. Production Claude Code traces (SemiAnalysis, 2026): TTFT at the proxy by appended tokens, main-agent requests from the second turn on. The serving stack and load behind the proxy are unknown.

appended tokens n	requests	TTFT median (s)	p90 (s)
< 1K	11 100	4.06	8.76
1–4K	11 857	3.02	7.44
4–16K	2 676	3.70	8.59
16–64K	497	3.82	9.28
$\geq 64K$	126	5.26	15.86

Table 16. Resume probability and pause time as a scheduler could estimate them, production Claude Code traces (SemiAnalysis, 2026): p is the probability that a turn is followed by another within ten minutes and τ the gap before that next turn, by turn index and by the gap that preceded the turn. The scheduler uses such conditional means as its estimate of the remaining pause.

	turns	p	τ mean (s)	τ median (s)
turn 1	183	0.99	4	0.0
turns 2–5	728	0.99	19	1.4
turns 6–20	2 639	0.97	23	1.8
turns 21–50	3 816	0.96	25	2.5
turn ≥ 51	19 073	0.97	22	4.3
gap 0–2 s	7 661	0.99	12	1.4
gap 2–10 s	12 831	0.98	21	4.6
gap 10–60 s	2 478	0.98	27	8.1
gap 60–600 s	2 664	0.85	65	9.0

primary metric: whether the model picks the same policy as the real optimum and ranks policies in the same order (Kendall τ over policy sets), reported alongside the error of TTFT, TPOT and throughput on held-out workloads. The *cost fit* fits (1), (2) and $\varphi(m)$ and measures the batch that fits under the trace’s footprint law. The *replays* test whether the prefill wait tracks $\lambda \mathbb{E}[S^2]/(2(1-\rho))$, report the number of live sessions next to the load, and force misses on a controlled fraction of turns to compare the rise in summed TTFT with Proposition 1. The *offloading test* asks whether the priced keep/offload/drop choice is never below never-offload while always-offload falls below it once the tier saturates. The *eviction replay* replays eviction events: shortest-first within a factor 2 of the optimum when resume probabilities are uniform, the byte-second key ahead when pause times differ, and the admission cap swept end to end, since the simulation says it decides whether eviction matters at all. The *placement test* measures the move cost and the load at which affinity loses, against ρ^* of (8). The *faithfulness* scoring scores the model and the calibrated simulator against the testbed. The result-table layouts are in the research plan distributed with the code; the cost fit and the replays are reported in §4.3 and Appendix D.

Table 17. Cost fit: prefill probes on the testbed (single requests on an idle replica group): cached prefix K , new tokens n , measured TTFT (mean $\pm$ half-range over repeats) and the fitted cost model $P(n, K) = c_0 + an + bn(K + n/2)$.

K	n	TTFT (s)	$P(n, K)$ (s)	error
0	1k	0.22 ± 0.00	0.25	+14.1
0	4k	0.87 ± 0.00	0.89	+2.3
0	16k	4.10 ± 0.01	4.10	+0.0
0	32k	9.92 ± 0.01	9.89	-0.3
0	64k	26.72 ± 0.05	26.73	+0.1
0	96k	50.56 ± 0.17	50.56	-0.0
16k	512	0.18 ± 0.00	0.20	+12.5
16k	2k	0.65 ± 0.00	0.67	+3.3
16k	8k	2.71 ± 0.01	2.73	+0.6
48k	512	0.32 ± 0.01	0.31	-2.6
48k	2k	1.11 ± 0.00	1.11	-0.3
48k	8k	4.45 ± 0.00	4.47	+0.4
88k	512	0.50 ± 0.01	0.44	-10.5
88k	2k	1.71 ± 0.04	1.66	-3.3
88k	8k	6.65 ± 0.02	6.66	+0.1

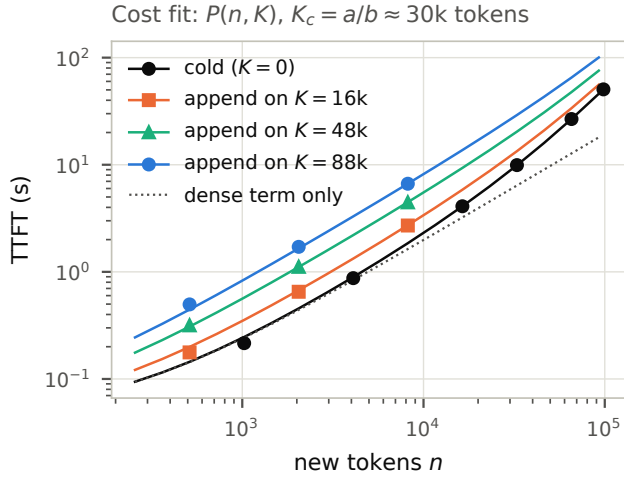


Figure 5. Cost fit: measured TTFT of cold prompts and of appends onto a cached prefix of K tokens (markers, mean over repeats) against the fitted $P(n, K)$ (lines); the dotted line is the dense term alone.

Table 18. Long-context replay: open-loop replay of the coding-agent trace on the testbed (four ranks, sessions pinned), one session every s seconds, at most $N_{\max}$ live. $\bar{N}$: mean live sessions; door: mean wait for admission at the client; sojourn: mean session time including it; λ : turns completed per second; hit: follow-up turns whose cached prefix covers at least 90 % of the previous prompt plus completion; KV: mean pool occupancy; ρ : per-rank prefill utilisation from the fitted cost model; TTFT hit/partial/miss: mean over follow-up turns whose prefix was resident, partly evicted, or gone; W_q : the server’s mean queueing time before the first prefill chunk; PK and finite: the open PK wait and the Markovian finite-source wait (Proposition 2, Z = measured decode plus think time) per rank, request-weighted. The finite-source wait exceeds the PK wait at the loose caps because at that Z the closed model’s utilisation on the busiest rank is 0.90 (0.75 at the cap of 16) against the observed 0.38: neither Z nor the cost model contains the block wait.

s (s)	$N_{\max}$	$\bar{N}$	door (s)	sojourn (s)	λ (1/s)	hit (%)	KV (%)	ρ per rank	TTFT hit (s)	partial (s)	miss (s)	p99 (s)	W_q (s)	PK (s)	finite (s)
20	∞	18.2	0	2583	0.070	46	61	0.03–0.38	2.6	80.1	278.6	503	122.6	3.2	14.5
20	16	12.3	499	2139	0.075	59	52	0.03–0.39	2.6	17.3	184.1	369	57.5	2.8	6.8
20	8	6.5	638	1344	0.092	83	37	0.03–0.20	2.6	24.5	117.3	176	7.3	0.7	0.5
10	8	6.6	772	1481	0.092	84	38	0.03–0.20	2.0	25.5	142.4	175	6.8	0.7	0.5

Table 19. Short-context replay: open-loop replay of the trace derived from the long-context one (nine copies of the 37 sessions, first prompt 2k tokens, the source’s appends scaled by 0.44 so that the mean append is ≈ 940 tokens, contexts 2k to 9k at the median and up to 27k, outputs of at most four tokens, the source’s think times capped at 30 s, a nonce per session so that no two share a prefix; sessions pinned round-robin over four ranks), one session every s seconds, with a share δ of follow-up turns forced to miss by a nonce at the head of that and every later prompt of the session (seed: the draw of nonces and forced turns). Sessions: replayed; $\bar{N}$: mean live sessions per rank; λ : turns per second per rank; ρ : per-rank prefill utilisation from the fitted cost model; CV_A^2 : interarrival CV^2 at a rank; hit: follow-up turns whose cached length is at least the previous prompt plus completion less one 512-token sub-block (the server rounds the cached length down to the sub-block); TTFT of hits and misses (mean) and p99 over all turns; W_q , S_{srv} , front: the server’s mean queueing time before the first prefill chunk, its prefill time, and the remainder of the mean TTFT (client and frontend); PK and finite: the open PK wait and the Markovian finite-source wait (Proposition 2) per rank; cap: share of the one-second samples with eight sequences running on the busiest rank. Window: from 90 s after the first arrival to the last arrival.

s (s)	δ (%)	seed	sessions	$\bar{N}$	λ (1/s)	ρ per rank	CV_A^2	hit (%)	TTFT hit (s)	miss (s)	p99 (s)	W_q (s)	S_{srv} (s)	front (s)	PK (s)	finite (s)	cap (%)
5	0	0	222	3.3	0.49	0.17–0.17	1.5	99	0.45	nan	1.7	0.04	0.37	0.04	0.05	0.03	0
3.5	0	0	222	4.8	0.71	0.24–0.25	1.3	99	0.48	nan	1.5	0.05	0.38	0.05	0.09	0.07	0
3.5	10	0	222	5.1	0.71	0.30–0.35	1.2	90	0.61	2.12	4.3	0.17	0.52	0.07	0.26	0.13	0
2.5	0	0	333	7.0	0.99	0.35–0.36	1.0	98	0.57	1.61	2.2	0.11	0.42	0.07	0.16	0.13	0
2.5	0	1	333	7.0	0.99	0.35–0.36	1.0	98	0.57	1.88	2.3	0.10	0.42	0.07	0.16	0.13	0
2.5	10	0	333	8.0	0.98	0.46–0.49	1.1	87	0.89	2.59	6.4	0.45	0.59	0.08	0.54	0.28	1
2.5	10	1	333	8.0	0.99	0.46–0.53	1.0	87	0.91	2.55	6.3	0.45	0.60	0.08	0.59	0.30	1
1.5	0	0	333	20.7	1.24	0.67–0.74	1.3	75	5.22	11.84	31.9	5.76	0.74	0.41	1.66	2.42	25

Table 20. The price test: the forced-miss arm against the baseline at the same spacing and seed (Table 19), per rank and summed. Forced: turns forced to miss in the window; unforced: follow-up turns with nothing resident, baseline $\rightarrow$ forced arm (the pool evicts more once the forced arm holds more KV); ρ , ρ' : the baseline’s prefill load and the load the added work of every miss the change caused implies; L_P : baseline number in the prefill stage (λ times mean TTFT); ΔL_P : its observed rise; [lo, hi]: the bracket of Proposition 1 from the baseline’s λ , ρ and PK wait and each miss’s own hit and miss work, over the forced misses plus the unforced ones net of the baseline’s; HOL: share of lo from the head-of-line term of (4); byst.: share of ΔL_P borne by turns that did not miss; Fin.: the finite-source price of Proposition 2.

s	seed	rank	forced	unforced	ρ	ρ'	L_P	ΔL_P	lo	hi	HOL (%)	byst. (%)	Fin.
3.5	0	0	35	0 $\rightarrow$ 0	0.24	0.31	0.33	0.22	0.18	0.20	58	57	0.09
3.5	0	1	50	0 $\rightarrow$ 0	0.24	0.31	0.35	0.17	0.15	0.17	47	48	0.09
3.5	0	2	43	0 $\rightarrow$ 0	0.25	0.35	0.36	0.24	0.28	0.33	60	45	0.13
3.5	0	3	35	0 $\rightarrow$ 0	0.25	0.31	0.35	0.12	0.13	0.14	49	42	0.08
3.5	0	all	163	0 $\rightarrow$ 0			1.39	0.75	0.74	0.83	55	49	0.39
2.5	0	0	57	7 $\rightarrow$ 16	0.36	0.48	0.59	0.51	0.44	0.55	67	61	0.19
2.5	0	1	69	4 $\rightarrow$ 11	0.35	0.48	0.58	0.41	0.39	0.48	61	60	0.19
2.5	0	2	62	8 $\rightarrow$ 21	0.35	0.50	0.59	0.60	0.50	0.64	65	57	0.22
2.5	0	3	55	7 $\rightarrow$ 30	0.36	0.50	0.60	0.53	0.47	0.60	63	52	0.22
2.5	0	all	243	26 $\rightarrow$ 78			2.36	2.05	1.80	2.27	64	58	0.82
2.5	1	0	57	5 $\rightarrow$ 22	0.35	0.48	0.56	0.58	0.43	0.54	62	62	0.21
2.5	1	1	62	8 $\rightarrow$ 11	0.36	0.47	0.60	0.35	0.37	0.45	61	55	0.17
2.5	1	2	62	8 $\rightarrow$ 19	0.36	0.48	0.60	0.47	0.40	0.50	61	58	0.20
2.5	1	3	70	8 $\rightarrow$ 32	0.36	0.54	0.59	0.75	0.69	0.95	68	58	0.28
2.5	1	all	251	29 $\rightarrow$ 84			2.34	2.15	1.89	2.44	64	59	0.86

Table 21. Experiments, the result each tests, where it runs (T = testbed, S = simulator, O = offline on traces) and its primary metric.

Experiment	Tests	Where	Primary metric
Cost fit	fits of (1), (2), φ	T	held-out MAPE
Replays	§2.3, Props. 1, 3	T, S	PK wait, ΔL_P
Offloading test	keep/offload/drop (§3.3)	T, S	turns/s
Eviction replay	§3.1, Prop. 4, cap	O, S, T	cost/OPT, TTFT
Placement test	Eq. (8)	T, S	inversion load
Faithfulness	decision faithfulness	T, S	Kendall τ