RENT OR RECOMPUTE? LEARNING-AUGMENTED PLACEMENT OF AGENT KV CACHES ACROSS MEMORY AND STORAGE TIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents keep contexts of 100K tokens or more alive across dozens of turns. Most of that time, they sit idle, waiting on tool calls, sub-agents, and people. Each idle gap forces a serving system to decide whether a session's KV cache is worth keeping and where it should live. Production traces show that prefix-cache hit rates collapse once a session idles for minutes, and a miss costs several times the pre-fill for the next turn. Today, this decision is made based on recency or fixed time-to-live rules. We show that placing an idle session's KV cache across GPU memory, host memory, and storage is an instance of multi-state dynamic power management. Each tier is a state with a rent rate. Returning from a tier, by reload or by recomputation, is its wake-up cost. This lets us import learning-augmented ski-rental guarantees directly: a policy that consumes a predicted idle gap is $(1 + \lambda)$-consistent and $(1 + 1/\lambda)$-robust per gap, for a trust parameter λ. We train an idle-gap predictor on features observable when a request finishes, and evaluate it on 346K idle gaps from 4,265 real coding-agent sessions, split by user. Whether a gap ends in a tool result or a human reply is observable, and it matters (seconds versus minutes). The length of the gap within each kind is only weakly predictable: 42% of tool gaps and 24% of human gaps fall within $2\times$ of the prediction, against 26% and 23% for a per-tool lookup table. On held-out users, the learning-augmented policy meets its per-gap bounds exactly. At a reference parameter set, a hedged policy with a validation-tuned trust parameter uses 11% fewer GPU-seconds than the best prediction-free policy; exact predictions would save 20%. Using the same predictions without hedging costs $7.6\times$ the clairvoyant optimum. A closed-form crossover context length, $K^\star = (\text{bytes/token} \,/\, B_{\text{eff}} - a)/b$, says when a storage tier beats recomputation. We measured its inputs with vLLM for an 8B model on H100, A100, and L40 GPUs, and for a 72B model on four H100S GPUs, across five kinds of storage paths from seven clients on three clusters. The binding quantity is usually the client's network link, not the storage system: tiers cap at the link (1.2 GB/s on 10 GbE nodes, 3.0 GB/s on 25 GbE). At measured prefill costs, an L40 breaks even against every mount we measured; an A100 against a 10 GbE NFS mount at 34K tokens; and an H100 against a 25 GbE mount at 32K tokens, or a node-local NVMe at every length. The 72B model prefills $6.8\times$ slower than the 8B model because tensor parallelism over PCIe returns only a third of the four GPUs. It therefore needs only 1.1 GB/s, and a 1.3 GB/s NFS mount beats recomputing it at every length. Re-running the policy comparison with the measured parameters, where prefill is 4–28$\times$ costlier per token than the assumed reference set, removes the learned policy's advantage (0–3%). With a 1 s host-memory round trip against a 9–32 s re-prefill, the decision points move into the tails, where the predictor has no skill. Our analysis identifies this as the regime where prediction has the least to decide, and the advantage returns (10%) only when host memory becomes expensive. Because the policy consumes only the predicted *tier*, the measurements turn into targets: a predictor must flag the long tool calls that a regression on gap length misses, and a link must deliver 2.5 GB/s to an H100 for storage to pay. We release the harness, simulator, and predictor as a testbed for both.

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.

1 INTRODUCTION

An LLM agent is not a request. It is a session. A coding agent holds a prefix of a hundred thousand tokens or more, issues a tool call, waits, appends the result, and repeats dozens of times before a human reads the output and replies. Recent characterizations of production agent traffic agree on the shape of this workload. In 4,265 Claude Code and Codex sessions, human pauses have a median of 1.4 minutes and a 99th percentile of 13.9 hours, and they account for 92% of session wall-clock (Zhu et al., 2026a). Across 13 million GitHub Copilot sessions, within-turn KV-cache hit rates near 90–98% fall to about 55% across turn boundaries, and they collapse once a session idles for 2–10 minutes (Liu et al., 2026). NVIDIA's serving guidance puts the problem in one sentence: a 2–30 second tool-call pause can age out an agent's entire prefix (Dhanani & Kosec, 2026).

Every idle gap is therefore a decision. The session's KV cache, tens of gigabytes for a long context, can stay in GPU memory, move to host memory, be written to a storage tier, or be discarded and recomputed on return. Serving systems make this decision with recency (LRU) or with a fixed time-to-live. Recent work that makes the decision smarter operates over at most two tiers (Pan et al., 2025; Li et al., 2025a; Xia et al., 2026; Zhang et al., 2026; Tiwari et al., 2026). Storage-tier systems build the tiers, but they drive them with the same recency or TTL rules (Qin et al., 2025; Liu et al., 2025; Xie et al., 2026; Qiu et al., 2026; Zhu et al., 2026b).

This paper. We observe that the decision has a classical form. Holding a session's KV cache in a tier while waiting for an unknown return time is a rent-or-buy problem. With several tiers it is exactly *dynamic power management with multiple states* (Irani et al., 2003; Lotker et al., 2008). Each tier is a state with a power rate, which is its rent. Returning to the active state costs a wake-up penalty, which is a reload or a recomputation. Antoniadis et al. (2021) gave learning-augmented algorithms for this problem with tight consistency-robustness trade-offs. We instantiate their model for KV caches and ask the two questions the theory leaves open. *Can the idle gap be predicted from what a serving system sees at decision time?* And *how much does the prediction buy on real agent traces?*

Contributions.

- **Formulation (§3).** We model KV placement across HBM, host DRAM, storage, and recomputation as multi-state ski rental. Rents and wake-up costs are in GPU-seconds and come from measurable quantities: bytes per token, tier bandwidth, and prefill cost $R(K) = aK + bK^2$. A closed-form crossover context length follows (Prop. 1). A fan-out corollary bounds when a shared tier dominates node-local placement (Prop. 2).
- **Guarantees (§4).** A deterministic policy scales the optimal policy's breakpoints toward a predicted tier. It is $(1 + \lambda)$-consistent and $(1 + 1/\lambda)$-robust per gap (Thm. 1), via the reduction of Antoniadis et al. (2021). The randomized variant inherits their tight frontier.
- **Predictability (§4.2).** We train an idle-gap predictor on features that are observable when a request finishes, on public traces with a user-level split. The tool/human distinction is observable and carries most of the value. Within a kind, gaps are only weakly predictable: 42% of tool gaps and 24% of human gaps fall within $2\times$, versus 26% and 23% for a per-tool lookup. A per-type-median heuristic is *worse* than no prediction. The round's tool latency would make tool gaps look predictable, but it is measured during the gap itself, so we exclude it as leakage.
- **Evaluation (§5).** On 46,700 held-out idle gaps at reference parameters, the hedged policies use 7–11% fewer GPU-seconds than the best prediction-free policy (exact predictions: 20%). The per-gap maxima equal $1 + 1/\lambda$ exactly. Using the same predictions without hedging costs 6–8$\times$ the optimum. Cost parameters measured with vLLM on four GPU configurations and five kinds of storage path (§5.5) show that the client link, not the storage system, usually sets the crossover. It is 34K tokens on an A100 over 10 GbE, 32K on an H100 over 25 GbE, and every length for the 72B model over 1.3 GB/s. At measured prefill costs the problem becomes two-tier, and the learned policy's advantage disappears (0–3%) until host memory becomes expensive.
- **Targets (§6).** The policy uses only the predicted tier. The open problem is therefore a classifier of breakpoint crossings. It is worth 16–20% of GPU-seconds and currently fails

on the long tool calls. On the storage side, the target is the required link bandwidth per GPU class. The released harness and simulator make both measurable.

2 BACKGROUND AND RELATED WORK

Agent workloads. Zhu et al. (2026a) released 357K LLM rounds from 43 developers with per-round timing. Liu et al. (2026) characterized 13M Copilot sessions and found that a lightweight idle-time predictor captures 86–90% of idle time. The AgentX traces show median inputs of 142K tokens across 43 turns, with sub-agents in 44% of sessions (vLLM Team & Inferact, 2026). Practitioners treat KV-cache hit rate as the primary production metric (Manus, 2025).

KV cache tiering. Mooncake pools DRAM and SSD across a cluster (Qin et al., 2025). LMCache is a reusable layer with CPU, disk, and remote backends (Liu et al., 2025). Strata, Tutti, CXL-attached memory, and object storage extend the hierarchy (Xie et al., 2026; Qiu et al., 2026; Jang et al., 2026; Zhu et al., 2026b). KVTC transform-codes KV for compact storage (Staniszewski & Łańcucki, 2026). Two recent measurements bound the benefit. On H100-class GPUs, the average request in some real workloads sits below the break-even point for NVMe offload (Kanichai et al., 2026). Offloading is within noise below about 90% KV utilization (Mao & Zhou, 2026). All of these systems decide what to keep with recency or with a fixed TTL.

Agent-aware and learned policies. KVFlow evicts by steps-to-execution in a workflow graph (Pan et al., 2025). Continuum pins KV during a tool call, with a TTL derived from the predicted tool duration (Li et al., 2025a). MORI offloads idle programs to host memory (Xia et al., 2026). CacheWise predicts time-to-reuse from tool arguments (Tiwari et al., 2026). CacheScout learns a Markov model over agent transitions (Zhang et al., 2026). For chat prefix caches, Learned Prefix Caching predicts continuation (Yang et al., 2025), and SAECache learns token-type-aware eviction (Fang et al., 2026). These operate GPU-only or between GPU and CPU, and none carries a competitive guarantee.

Learning-augmented online algorithms. Ski rental with predictions has a $(1+\lambda)$-consistent, $(1+1/\lambda)$-robust deterministic algorithm (Purohit et al., 2018). Multi-state power management is multislope ski rental. It is 2-competitive deterministically (Irani et al., 2003) and $e/(e-1)$-competitive with randomization (Lotker et al., 2008). Antoniadis et al. (2021) give the learning-augmented version, with a tight error dependence and a reduction from many states to two. Variants exist (Wang et al., 2020; Li et al., 2025b). Learned routing and randomized eviction for KV caches were studied at ICLR 2026 (Wang et al., 2026). Our contribution is the KV-placement structure and its empirical validation, not the algorithmic template.

3 KV PLACEMENT AS MULTI-STATE SKI RENTAL

3.1 TIERS, RENTS, AND WAKE-UP COSTS

A session with K tokens resident goes idle at $t = 0$ for an unknown duration T. Its KV cache occupies $B = K \cdot \mathrm{bpt}$ bytes, where $\mathrm{bpt} = 2\, n_{\mathrm{layers}}\, n_{\mathrm{kv}}\, d_{\mathrm{head}}\, s_{\mathrm{dtype}}$. This is 128 KiB/token for Llama-3.1-8B in fp16 and 320 KiB for 70B; fp8 KV halves both. While idle, the cache may descend monotonically through tiers $j \in \{0,1,2,3\}$ = {HBM, host DRAM, storage, dropped}. Following Antoniadis et al. (2021), tier j has a rent rate α_j, paid per second of residence, and a wake-up cost β_j, paid when the session returns from tier j:

$$\alpha_j = c_j\, f(K), \quad f(K) = \frac{B}{\text{HBM KV budget}}, \quad \beta_j = \sum_{i \le j} W_i + L_j, \quad \beta_3 = R(K) = aK + bK^2. \tag{1}$$

Here $c_0 = 1 > c_1 > c_2 > c_3 = 0$ is the relative byte-second cost of each tier, W_i and L_i are the write and read times through the path to tier i (from measured bandwidths), and $R(K)$ is the measured prefill cost. The rent is an opportunity cost. Holding B bytes in HBM for t seconds rents the fraction $f(K)$ of the GPU's KV capacity for t seconds, so all quantities are GPU-seconds. Transfer seconds are charged at a factor $\gamma \le 1$. We use $\gamma = 1$, which is conservative against offloading. $R(K)$ is the

isolated, batch-1 cost, which is again conservative, since deployed recompute competes with other requests. The cost of a policy that ends in tier j after an idle gap T is $\beta_j + \int_0^T \alpha_{j(t)}\, dt$.

3.2 The optimal offline policy and the crossover

The clairvoyant optimum is the lower envelope of four lines, $\mathrm{OPT}(T) = \min_j(\beta_j + \alpha_j T)$. It visits tiers in increasing order and skips any tier that is never on the envelope. A tier whose wake-up cost exceeds recomputation is skipped in this way, which happens whenever a storage round trip is slower than re-prefill. The deterministic *lower-envelope* policy (LE) follows OPT's own breakpoints and is 2-competitive (Irani et al., 2003). On the segment where OPT uses tier j, $\mathrm{LE}(T) = 2\beta_j + \alpha_j T \le 2\,\mathrm{OPT}(T)$ (Appendix A).

Proposition 1 (Crossover context length)**.** *Take a tier with effective restore bandwidth B_{eff} in bytes/s (the slower of the tier and the PCIe leg when pipelined) and a prefill cost $R(K) = aK + bK^2$. Loading beats recomputing for all contexts longer than*

$$K^\star = \frac{\mathrm{bpt}/B_{\text{eff}} - a}{b}. \tag{2}$$

When $\mathrm{bpt}/B_{\text{eff}} < a$, $K^\star = 0$ and the tier always wins. When $b \le 0$ there is no crossover and recompute always wins.

Two consequences drive the empirical study. First, $K^\star$ scales with $1/b$, so a faster GPU needs a *longer* context before any tier pays. This is the negative result of Kanichai et al. (2026) in closed form. Second, going from 8B to 70B multiplies bpt by $\sim$2.5 but prefill compute by $\sim$10. The bandwidth a tier needs should therefore fall $\sim 4\times$, and the tier's value should concentrate in large models. Section 5.5 measures $2.3\times$.

Proposition 2 (Fan-out)**.** *Suppose a prefix will be re-read by N sub-agents spread over M workers. Node-local placement serves about N/M of them and forces the rest to recompute. Shared placement serves all N at the contended restore rate $B(N)$. Shared placement dominates when $\frac{B}{B(N)} < \left(1 - \frac{1}{M}\right) R(K)$, that is, up to the fan-out $N^\star$ at which contention degrades $B(N)$ below $B/((1-1/M)R(K))$.*

$N^\star$ is a property of the storage system under contention. Measuring it requires controlled shared storage, and we leave it to follow-up work (§6).

4 Learning-augmented placement

4.1 The policy

Let $0 = t_0 < t_1 < \dots$ be OPT's breakpoints on the envelope. Let $\hat{T}$ be a predicted idle gap, with predicted optimal tier $j^\star = \arg\min_j(\beta_j + \alpha_j \hat{T})$. The policy $\mathrm{LA}(\lambda)$, $\lambda \in (0, 1]$, follows the envelope with scaled breakpoints. Transitions that the prediction says will happen (into tiers up to $j^\star$) are taken at λt_i. Later transitions are taken at t_i/λ. With $\lambda = 1$ the policy is LE. As $\lambda \to 0$ it jumps to the predicted tier immediately and never leaves. A Bayes variant replaces the point prediction with three quantiles (q_{10}, q_{50}, q_{90}) and targets the tier of least expected cost.

Theorem 1. *For any idle gap, $\mathrm{LA}(\lambda)$ incurs at most $(1+\lambda)\,\mathrm{OPT}$ when the prediction is exact and at most $(1 + 1/\lambda)\,\mathrm{OPT}$ for any prediction.*

Proof sketch. Consistency is a telescoping argument. LE's rent up to t_j equals $\mathrm{OPT}(t_j) = \beta_j + \alpha_j t_j$, so scaling the breakpoints before $j^\star$ by λ gives $\mathrm{ALG} = (1+\lambda)\beta_{j^\star} + \alpha_{j^\star} T \le (1+\lambda)\mathrm{OPT}$ (Appendix A). Robustness is a case analysis on whether the algorithm is below, at, or beyond the predicted tier when the gap ends. In each case the same telescoping identity bounds its rent by OPT at a scaled breakpoint, and the concavity of OPT turns the scaling into the factor $1/\lambda$. The full proof is in Appendix A. It is the deterministic special case of the reduction of Antoniadis et al. (2021, Lemma 3) and of the two-state bound of Purohit et al. (2018). □

Antoniadis et al. (2021) also give a randomized policy with cost $\le (1+\epsilon)\min_{\rho \in [1, e/(e-1)]}\{\rho\,\mathrm{OPT} + \mu(\rho)\,\eta\} + O(\beta_k/\epsilon \log(1/\epsilon))$ for prediction error $\eta = \alpha_0|\hat{T} - T|$, with tight μ. We deploy the

deterministic policy for its simplicity. Because λ is fixed per gap before the gap is observed, the bounds hold unchanged for a λ chosen per gap type (§4.3).

4.2 THE IDLE-GAP PREDICTOR

When a request finishes, the serving system knows several things. It knows whether the model ended with a tool call or with a message to the user, which tools it called and how many, the round index, the session's age, the resident context, and the previous gap. None of these looks into the future. We featurize them, together with the size of the tool calls' arguments, the length of the model's message, what the previous round's tools returned, and the clock (Appendix C). On these features we fit gradient-boosted quantile regressors for q_{10}, q_{50}, q_{90} of $\log(1+T)$. "Ended with a tool call" agrees with the ground-truth cause of the next round on 97.8% of gaps, so the human/tool distinction is effectively observable. One trace field is deliberately *not* a feature: each round records its tools' wall latency, which is measured during the gap itself and equals it within 2% on most tool rounds. A predictor given it reaches 85% of tool gaps within $2\times$, a number no serving system can obtain at decision time.

4.3 CHOOSING λ

λ trades consistency for robustness and should track prediction quality. We select λ_{tool} and λ_{human} from $\{0.1, 0.25, 0.5, 0.75, 1\}$ on half the held-out users and report on the other half. Since $\lambda = 1$ (LE) is in the grid, the tuned policy is never worse than the prediction-free one on validation. Setting λ from the predicted interval width $\log(q_{90}/q_{10})$ did not help (Appendix D): interval width is not aligned with the error that matters, which is whether T crosses a breakpoint.

5 EVALUATION

5.1 SETUP

Traces. We use TraceLab's public release (Zhu et al., 2026a): 357,161 LLM rounds from 4,265 Claude Code and Codex sessions by 43 pseudonymous developers (CC BY 4.0). We extract one row per idle gap, the time from a round's last event to the next round's first event. This gives 346,030 gaps with the features above and the ground-truth cause of the next round. Our extraction reproduces the paper's published statistics (human p_{99} = 13.9 h, 93% of idle time; tool p_{50} = 0.2 s). We split *by user*: 31 users train (299,330 gaps) and 12 users test (46,700 gaps). Gaps under 10 ms are not decisions and are excluded. So are sessions whose KV exceeds the GPU's budget for a given configuration, since they cannot be resident at all.

Cost parameters. Sections 5.3–5.4 use a *reference* parameter set for an 8B model on an H100-class server (Appendix E). We chose it so that recomputation and the tiers are within a small factor of each other and every decision is live. Section 5.5 then reports parameters measured with the released harness and re-runs the comparison at them. The harness measures prefill on vLLM with prefix caching disabled, PCIe transfers, and tier bandwidth under warm and cold conditions with integrity verification.

Metrics. We report total GPU-seconds relative to the clairvoyant optimum (aggregated over gaps), the distribution of per-gap ratios (the guarantees are per instance), and the added time-to-first-token on return.

5.2 ARE IDLE GAPS PREDICTABLE?

Figure 1 and Table 1 answer the question by cause. On held-out users the predictor's q_{50} pinball loss in log-space is 0.371. A lookup table indexed by tool signature gets 0.432, a per-type median 0.508, and a global median 0.702. The q_{10}–q_{90} interval covers 78% of gaps (target 80%). Within a factor of two of the truth are 42% of tool gaps and 24% of human gaps, against 26% and 23% for the signature lookup. So the model adds to the lookup table, but the tool signature still carries most of what is learnable: `Read` returns in 0.2 s at p_{90}, and `Bash` has a 32 s tail. Human gaps are a heavy tail that no feature we have explains. The length of a `Bash` call is not written in its arguments. This is why the robustness half of the guarantee matters.

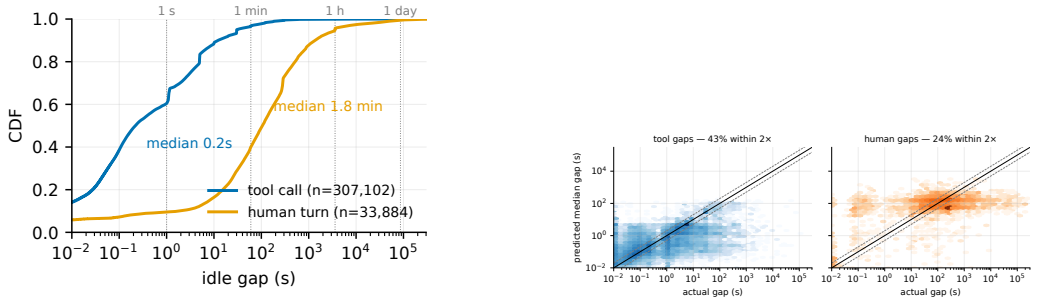

Figure 1: **Left:** idle gaps by cause. Human turns are heavy-tailed (median 1.75 min, p_{99} 13.9 h) and make up 93% of idle time. Tool calls are short (median 0.2 s), but the 3% that exceed a minute account for 82% of tool time. **Right:** predicted median vs. actual gap on held-out users, with the 2× band.

Table 1: Predictor on 46,700 held-out gaps from 12 unseen users. The per-signature median is a lookup table over (gap type, tool signature) with a per-type fallback. Its quantiles give the coverage number.

	predictor	per-signature median	per-type median	global median
pinball loss, q_{50}, log-space	**0.371**	0.432	0.508	0.702
within 2× of truth (all / tool / human)	**40% / 42% / 24%**	25% / 26% / 23%	16% / 14% / 22%	–
q_{10}–q_{90} coverage	78%	83%	85%	–

5.3 Policy comparison

Table 2 and Figure 2 compare policies on the held-out gaps under the reference parameters. Two prediction-free baselines bracket the problem. Keeping everything in HBM costs 115× the optimum, because of human pauses. Evicting and recomputing costs 2.6×, because of tool calls. Fixed TTLs sit between the two and are sensitive to the threshold (1.8× at 10 s, 4.4× at 60 s). The lower-envelope policy, the best a prediction-free deterministic algorithm can guarantee, costs 1.69×, with a per-gap p_{90} of exactly 2.000. Using the predictor *without* hedging is far worse than not using it. Jumping straight to the predicted tier (LA with $\lambda \to 0$) costs 7.6×, because a wrong "short" prediction strands a session in HBM through a human pause (10.5× on tool gaps, 3.8× on human gaps, worst gap 2.5×10^4). A TTL equal to the predicted gap, the natural rule for a system that pins KV for a predicted duration (Li et al., 2025a), costs 5.9×, because every under-prediction ends in a re-prefill. Hedged, the same predictions help. LA with the point prediction and $\lambda = 0.5$ costs 1.68×, no better than LE. LA-Bayes, which targets the tier of least expected cost under the predicted $q_{10}/q_{50}/q_{90}$, costs 1.57×. With λ tuned on validation users, the deterministic policy costs **1.54**× the optimum on the disjoint test half, versus 1.73× for LE. That is a **10.8%** reduction. The tuned values are $\lambda_{\text{tool}} = 1$ and $\lambda_{\text{human}} = 0.25$: trust the ordinal information in human predictions, but not the noisy magnitude of tool predictions. Exact predictions would give 1.35× (20% below LE), so the predictor realizes half of what prediction could buy. A per-type-median "prediction" gives 1.72×, worse than LE, which is the case for learning rather than guessing. LA consumes $\hat{T}$ only through the predicted tier $j^\star$. The predictor names OPT's tier for 83% of gaps, against 82% for the per-type median, since three quarters of gaps are short tool calls where both say "stay". Its errors are almost all too shallow: the 14% of tool gaps that turn out long, which are exactly the costly ones. Per-gap maxima equal $1 + 1/\lambda$ for every λ, and the p_{90} equals

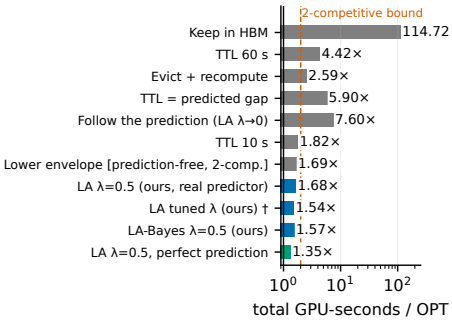

Figure 2: Total cost relative to the optimum (log scale) under the reference parameters. †: tuned on validation users, reported on disjoint test users.

Table 2: Held-out gaps, reference parameters. Cost: total GPU-seconds relative to the clairvoyant optimum. Per-gap columns: the distribution of per-instance ratios. TTFT: wall time added on return at p_{95}.

policy	cost/OPT	per-gap p_{90}	per-gap max	tool	human	TTFT p_{95} (s)
Keep in HBM	114.7	12.5	–	14.2	257	0.00
Follow the prediction (LA $\lambda \to 0$)	7.60	1.93	–	10.5	3.77	2.08
TTL = predicted gap (q_{50})	5.90	42.9	866	2.98	10.1	5.92
Evict + recompute	2.59	182	–	3.66	1.14	8.33
TTL 10 s	1.82	2.50	3.52	1.65	2.07	3.47
Lower envelope (prediction-free, 2-competitive)	1.69	**2.000**	**2.00**	1.51	1.95	2.65
LA λ=0.5, learned predictor	1.68	1.98	**3.00**	1.66	1.72	2.55
LA-Bayes λ=0.5 (ours)	1.57	1.51	**3.00**	1.54	1.61	2.69
LA, tuned λ (ours; test half, LE there 1.73)	**1.54**	–	–	–	–	–
LA λ=0.5, exact prediction	1.35	**1.500**	**1.50**	1.25	1.48	2.65

$1 + \lambda$ under exact predictions (Appendix B): the bounds are tight and never exceeded. Added TTFT at p_{95} is unchanged (2.55–2.69 s), because the learned policy changes *when* sessions leave HBM, not how many are recomputed.

Robustness to prediction error. Under random log-normal noise (Appendix B), predictions that are off by a factor of 55 on average degrade $\lambda = 0.25$ only to LE's level (1.75, not its worst case of 5). The real predictor's errors are worse than random noise, because they concentrate on the decisive gaps: 1.68 at $\lambda = 0.5$, against about 1.45 for random noise of the same scale.

5.4 When does the storage tier matter?

With the reference parameters the optimal solution never uses the storage tier. A 5 GB/s round trip for a 12 GB cache takes longer than the 1.3 s re-prefill, so tier 2 is dominated, and OPT goes HBM→DRAM→drop or HBM→drop. This is not a modeling artifact: it is Proposition 1 at these parameters, and Kanichai et al. (2026) reached the same conclusion experimentally. Figure 4 sweeps model class and tier bandwidth. Storage enters OPT for up to 13% of gaps on a 40 GB A100 at 50 GB/s, where re-prefill is slow and HBM is scarce. For 70B models the required bandwidth falls as the proposition predicts. The hedged policy's saving over LE is 6–12% across the surface, and it moves little with tier bandwidth. The saving comes from timing the HBM→DRAM and DRAM→drop decisions, not from the storage tier. Under host-memory pressure (Figure 4, right, with c_1 rising toward 1) the intermediate tier leaves OPT, and the problem degenerates to two-state HBM-vs-drop. LA-Bayes keeps its margin over LE there (1.52 vs 1.69), because the decision that remains, whether a gap outlasts the 7 s breakpoint, is the one the human/tool distinction answers.

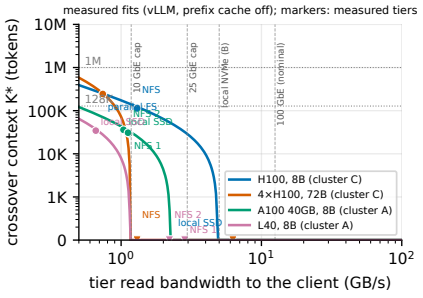

Figure 3: $K^\star$ against tier read bandwidth at the client, from the fits of Table 3. Markers are the tiers measured from each GPU's node (▼: $K^\star = 0$). Dashed guides: observed link caps and a nominal 100 GbE.

5.5 Measured cost parameters

Table 3 and Figure 3 report what the harness measured. Appendix E lists every client and condition. Four facts stand out. **Prefill is far costlier than the reference set assumes.** vLLM prefills Qwen3-8B at 23.3K tokens/s on an H100 PCIe, 12.4K on an A100, and 6.8K on an L40 at 2K tokens. These fall to 16.9K, 7.4K, and 4.6K at 32K tokens (every fit has $R^2 \geq 0.9999$). Even the H100 is 3.7× slower per token than the reference a. **Tier bandwidth is a property of the client, not of the storage system.** From the A100 node and from a data-mover node, both NFS systems and the node-local SSD read at 1.0–1.1 GB/s for one session, and the NFS systems saturate at 1.18 GB/s, which is a 10 GbE link. Cold equals warm there, because the link, not the array, is the bottleneck. The L40 node mounts the same all-flash system over 25 GbE and reads it at 2.84 GB/s. On the third cluster,

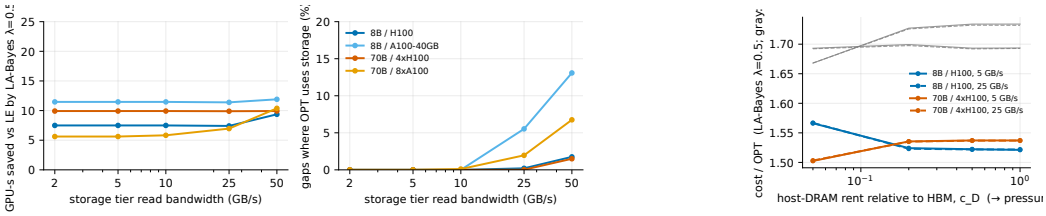

Figure 4: **Left pair:** across model class and storage bandwidth, the saving of LA-Bayes ($\lambda = 0.5$) over the lower envelope (left) and the share of gaps where OPT places the cache on storage (right). **Right:** as host-DRAM pressure rises ($c_1 \to 1$), the three-tier problem degenerates to two tiers. LA-Bayes (colored) keeps its margin over LE (gray).

Table 3: Measured cost parameters. $R(K) = aK + bK^2$ is fitted on vLLM 0.30 with prefix caching off, batch 1, and 2K–32K tokens, for Qwen3-8B (144 KiB/token, fp16 KV) on one GPU and for Qwen2.5-72B (320 KiB/token) on four H100s with TP=4, timed wall-clock on all four. Tier read bandwidth is measured from the GPU's own node with one session, 36 MiB O_DIRECT chunks, integrity verification, and the median of three runs. Cold reads were measured from CPU nodes (Appendix E): within 4% of warm on link-bound tiers, and 70% lower on the parallel file system. $K^\star$ is from Prop. 1, with reads not pipelined with PCIe.

GPU (cluster)	a (s/tok)	b (s/tok^2)	R^2	tier (client): read GB/s warm/cold	$K^\star$
H100 80GB PCIe, 8B (cluster C)	3.69×10^{-5}	6.76×10^{-10}	1.000	node-local SSD: 6.26 / –	0 (always)
				research NFS: 1.30 / –	122K
				parallel FS: 0.74 / –	249K
4×H100 80GB PCIe, 72B (cluster C)	2.84×10^{-4}	6.42×10^{-10}	1.000	node-local SSD: 6.26 / –	0 (always)
				research NFS: 1.30 / –	0 (always)
				parallel FS: 0.74 / –	248K
A100 40GB PCIe, 8B (cluster A)	7.33×10^{-5}	1.89×10^{-9}	1.000	all-flash NFS 1: 1.14 / –	34K
				all-flash NFS 2 (older): 1.12 / –	35K
				node-local SSD: 1.04 / –	40K
L40 48GB, 8B (cluster A)	1.33×10^{-4}	2.59×10^{-9}	1.000	all-flash NFS 1: 2.84 / –	0 (always)
				all-flash NFS 2 (older): 2.21 / –	0 (always)
				node-local SSD: 0.66 / –	37K

the H100 node has a 6.3 GB/s NVMe. It reads the parallel file system at 0.74 GB/s for one session but at 3.0 GB/s for eight, so its single-stream number is as much a property of the loader as of the system. **The crossover is low, and the link and the GPU generation set it.** With measured a and b, Prop. 1 needs only $\mathrm{bpt}/(a + bK)$ of effective bandwidth: 2.5 GB/s on the H100, 1.1 on the A100, and 0.7 on the L40 at 32K tokens. So on the A100 the 10 GbE NFS mount ties recomputation at $K^\star = 34$K tokens. On the L40, loading beats recomputing at every length ($\mathrm{bpt}/B_{\mathrm{eff}} = 58\,\mu$s/token $< a$). On the H100, the same 10 GbE mount would need 147K tokens, a 25 GbE mount 32K, and the parallel file system's single stream 249K, while the node-local NVMe wins at every length. The first-order lever for a storage tier is therefore the link in the GPU server, and the second is the GPU generation. The storage system itself was binding only where its per-stream service rate fell below the link: cluster B's older NFS system at 2.1 GB/s, and the parallel file system's single stream. **A 72B model lowers the bar less than its FLOPs suggest.** Qwen2.5-72B on four PCIe H100s (tensor parallel, no NVLink) prefills at 3.4K tokens/s. That is $6.8\times$ slower than the 8B on one H100; the parameter ratio over four GPUs predicts $2.2\times$. The term a rises $7.7\times$, while b is unchanged, because attention shards across heads for free. Prop. 1 then asks for 1.1 GB/s at 32K tokens. That is $2.3\times$ less than the 8B, not the $\sim 4\times$ that the FLOP count in Section 3 gave, and only $1.3\times$ less at 128K, where the shared b dominates. Any tier above $\mathrm{bpt}/a = 1.15$ GB/s now wins at every length. The 1.3 GB/s NFS mount does. Cluster A's 10 GbE mounts tie at 5–13K tokens, and the parallel file system's single stream needs 248K.

Table 4 re-runs the policy comparison at the measured parameters. The per-gap maxima still equal $1 + 1/\lambda$, but the learned policy's advantage disappears. Validation selects $\lambda = 1$ on the H100 for both models, so the tuned policy *is* LE. It saves 0.1% on the A100 and 0.9% on the L40. LA-Bayes is within $\pm 2.5\%$ of LE on the 8B rows and 5% worse on the 72B row. Exact predictions would still save 16–18%. The case for hedging grows: following the prediction without a robustness wrapper costs

Table 4: Policy comparison at measured parameters, with λ tuned on the disjoint test half. "Gaps kept" is the share of held-out gaps that fit the KV budget. The next three columns are for the median kept session.

parameters	KV budget	gaps kept	$R(K_{\text{med}})$	DRAM r/t	OPT leaves HBM	follow ($\lambda \to 0$)	LE	LA (tuned)	saving
reference (8B/H100-class, App. E)	60 GiB	82%	1.3 s	1.2 s	6.7 s	7.60	1.73	1.54	10.8%
H100 80GB PCIe, 8B (cluster C), parallel FS	58 GiB	80%	8.8 s	1.1 s	5.2 s	8.18	1.42	1.42	0.0%
4×H100 80GB PCIe, 72B (cluster C), parallel FS	156 GiB	83%	31.8 s	0.5 s	3.1 s	5.83	1.41	1.41	0.0%
A100 40GB PCIe, 8B (cluster A), all-flash NFS 1	21 GiB	61%	15.0 s	1.2 s	2.7 s	11.88	1.59	1.59	0.1%
L40 48GB, 8B (cluster A), all-flash NFS 1	26 GiB	68%	26.2 s	0.9 s	2.3 s	13.16	1.53	1.51	0.9%

6–13× the optimum at measured parameters, against 1.4–1.6× for LE. A wrong "short" prediction on a human gap strands a 10–28 GiB session in HBM for minutes, and a wrong "long" one pays a 9–32 s re-prefill. The H100 has the reference set's budget, but re-prefilling the median session costs 8.8 s rather than 1.3 s. So OPT parks sessions in host memory (24% of gaps) instead of dropping them (2%), and its second breakpoint moves from 32 s to 12 minutes. On the 40 GB A100, the median kept session (72K tokens, 10 GiB) rents half the KV budget, so OPT leaves HBM after 2.7 s rather than 6.7 s. The 72B model has a 156 GiB budget against a 32 s re-prefill, so OPT parks 36% of gaps in host memory and drops under 1%. In every measured case, the first breakpoint sits in the tool-gap tail and the second deep in the human tail. The predictor has no skill there, so validation refuses to trust either. Prediction pays when breakpoints fall where the predictor can tell which side a gap lands on. That holds at the reference set, whose 7–32 s HBM-versus-drop decision the human/tool distinction answers. Raising the host-memory rent brings the decisions back. At the measured H100 parameters, the tuned policy saves 0% for $c_1 \leq 0.1$, 1% at 0.2, and 10% at 0.5, before the problem collapses to HBM-versus-drop at $c_1 = 1$, where the validation choice does not transfer (Table 7). Storage enters OPT for 2% of gaps on the L40 and none elsewhere: even where loading beats recomputing, the *round trip* through a 1–3 GB/s link rarely beats a 1 s round trip to host memory.

6 Conclusion, limitations, and open problems

Placing an idle agent's KV cache is multi-state ski rental, and the learning-augmented theory transfers with its guarantees intact. Whether a gap ends in a tool result or a person is observable. Its length within each kind mostly is not. That is enough for a hedged policy to save 7–11% where the cost parameters let the tool/human distinction decide, and nothing where measured parameters move the decisions into the tails. It also makes unhedged prediction 6–13× worse than no prediction. The crossover formula answers whether a storage tier pays from three measured numbers. Measuring them showed that the GPU server's link, not the storage system behind it, decides.

Limitations. The model is per-session. Rents are opportunity costs and host memory is uncapacitated, so storage is rarely optimal here. The fleet setting is modeled only through c_1. Prop. 2 is derived, not measured. $R(K)$ is the isolated cost, and transfers are charged at $\gamma = 1$; both work against the tiers. The traces are coding agents from 43 developers. Measured parameters cover three PCIe GPU types, an 8B model on one GPU, and a 72B model on four without NVLink. c_1, c_2 and γ remain modeling choices. The reference set of Sections 5.3–5.4 is not a measurement.

What would change the answer. Each item is testable with the released code. (i) *Decision-aware prediction.* The policy consumes a tier, not a duration. The target is therefore a classifier of breakpoint crossings. It is worth at most the exact-prediction rows: 20% of GPU-seconds at the reference set and 16–18% at measured parameters. A regression on $\log T$ is the wrong tool for it. It matches a per-type median on tier accuracy, and it misses the 14–17% of tool gaps that turn out long. The promising signals are the agent's plan and the command's semantics, which agent-aware systems already expose (Pan et al., 2025; Zhang et al., 2026; Tiwari et al., 2026); the theory needs only a better $j^\star$. (ii) *The link.* The client's NIC set $K^\star$ on every server we measured, and Prop. 1 says what a path must deliver: 2.5 GB/s to an H100 at 32K tokens, and half that at 128K. The measurements that would move storage into OPT are 25 GbE, 100 GbE and RDMA paths, and parallel loaders for the parallel file system. (iii) *Capacity coupling.* Prediction returned (10%) once host memory became expensive (Table 7; c_1 is DRAM's shadow price). A fleet model with finite DRAM and the contended restore rate of Prop. 2 is where predictor and tier should pay most. (iv) *Closing the loop.* Non-coding agents, fan-out traces, and the policy inside a serving stack.

AI USE STATEMENT

In this work, we used generative AI tools, a large language model used as a coding and writing assistant, for the following: proof assistance (the proof sketches and Appendix A), implementation (the harness, simulator, and predictor), and dataset cleaning (extracting idle gaps from the public traces). We have not used generative AI tools for synthetic data generation or for translation assistance. Qualitative analysis is not applicable to this work. Additionally, we used generative AI tools for figure creation, paper drafting and reference formatting. We have reviewed all AI-assisted work. Every experimental number in the paper was produced by the authors running the released code on the described traces and clusters. The proofs were checked by the authors. The AI-assisted code was reviewed and tested by the authors, including the end-to-end self-test shipped with the harness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The only data used are the publicly released, pseudonymized coding-agent traces of Zhu et al. (2026a) (CC BY 4.0). We performed no data collection involving human subjects. Our extraction keeps only per-round timing and token counts, not prompt or response content. The cluster measurements ran under the acceptable-use policies of the clusters concerned and wrote only to the authors' own scratch space. The work informs how serving systems allocate memory and storage. We see no direct potential for harm beyond the general energy and cost considerations of LLM serving, which the work aims to reduce.

REPRODUCIBILITY STATEMENT

The traces are public (CC BY 4.0). The released artifact contains the gap extractor, the predictor with its user split and seed, the simulator with every parameter set used, and the cluster harness that produced the cost parameters, including the job scripts. All tables and figures regenerate from three commands (Appendix F).

REFERENCES

Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Learning-augmented dynamic power management with multiple states via new ski rental bounds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Ishan Dhanani and Matej Kosec. Full-stack optimizations for agentic inference with Dynamo. NVIDIA Dynamo documentation, v1.3.0 digest, `https://docs.nvidia.com/dynamo/v1.3.0/digest/agentic-inference`, 2026.

Shaoke Fang, Ziang Li, Wenfei Wu, et al. Not all tokens are worth caching: Learning semantic-aware eviction for LLM prefix caches. *arXiv preprint arXiv:2605.18825*, 2026.

Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems*, 2(3):325–346, 2003.

Hakbeom Jang, Inho Song, Hoshik Kim, et al. A CXL memory rack for multi-turn LLM serving. *arXiv preprint arXiv:2607.18141*, 2026.

Joseph Kanichai, Tiziano De Matteis, and Animesh Trivedi. Building py-kvcache: A performance characterization of external KV caching for vLLM with NVMe SSDs. *arXiv preprint arXiv:2609.11744*, 2026.

Hanchen Li, Runyuan He, Qiuyang Mang, et al. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025a.

Ziwei Li, Bo Sun, Zhiqiu Zhang, et al. Combinatorial ski rental problem: Robust and learning-augmented algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b.

Banruo Liu, Haoran Qiu, Íñigo Goiri, et al. Agentic coding in the wild: Characterizing GitHub Copilot traces at production scale. *arXiv preprint arXiv:2608.00101*, 2026.

Yuhan Liu, Yihua Cheng, Jiayi Yao, et al. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025.

Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease or buy: Randomized algorithms for multislope ski rental. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.

Manus. Context engineering for AI agents: Lessons from building Manus. `https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus`, 2025.

Gilbert L. Mao and Ruiyang Zhou. Energy characterization of KV cache offloading under agentic workloads. In *Workshop on Sustainable Computer Systems (HotCarbon)*, 2026.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, et al. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Ruoyu Qin, Zheming Li, Weiran He, et al. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *USENIX Conference on File and Storage Technologies (FAST)*, 2025.

Shi Qiu, Yifan Hu, Xintao Wang, et al. Tutti: Making SSD-backed KV cache practical for long-context LLM serving. *arXiv preprint arXiv:2605.03375*, 2026.

Konrad Staniszewski and Adrian Łańcucki. KV cache transform coding for compact storage in LLM inference. In *International Conference on Learning Representations (ICLR)*, 2026.

Shubham Tiwari, Tapan Chugh, Nash Rickert, et al. CacheWise: Understanding workloads and optimizing KVCache management for efficiently serving LLM coding agents. *arXiv preprint arXiv:2606.16824*, 2026.

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

Fangzhou Wang et al. Randomization boosts KV caching, learning balances query load. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2601.18999.

Shufan Wang, Jian Li, and Shiqiang Wang. Online algorithms for multi-shop ski rental with machine learned advice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Tian Xia, Hanchen Li, Zhifei Li, et al. Idleness is relative: Exploiting tool-call idle windows for offloading in agentic systems with MORI. *arXiv preprint arXiv:2606.00866*, 2026.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, et al. Strata: Hierarchical context caching for long context language model serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2026.

Dongsheng Yang, Austin Li, Kai Li, et al. Learned prefix caching for efficient LLM inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Rui Zhang, Chaeeun Kim, Shaoting Feng, et al. Learning agent execution for KV-cache management in agentic serving. *arXiv preprint arXiv:2608.14624*, 2026.

Kan Zhu, Mathew Jacob, Chenxi Ma, et al. Tracelab: Characterizing coding agent workloads for LLM serving. *arXiv preprint arXiv:2606.30560*, 2026a.

Yu Zhu, Aditya Dhakal, Yunming Xiao, et al. ObjectCache: Layerwise object-storage retrieval for KV cache reuse. *arXiv preprint arXiv:2605.22850*, 2026b.

A PROOFS

Setup. Tiers are $j = 0..k$ with $\alpha_0 > \cdots > \alpha_k \geq 0$ and $0 = \beta_0 < \cdots < \beta_k$, after removing tiers that are not on the envelope. The breakpoints t_j satisfy $\beta_{j-1} + \alpha_{j-1}t_j = \beta_j + \alpha_j t_j$, that is, $(\alpha_{j-1} - \alpha_j)t_j = \beta_j - \beta_{j-1}$.

Lemma 1 (LE's rent equals OPT at breakpoints)**.** $\sum_{i<j} \alpha_i(t_{i+1} - t_i) = \beta_j + \alpha_j t_j = \mathrm{OPT}(t_j)$.

Proof. Reindex the sum: $\sum_{i<j} \alpha_i t_{i+1} - \sum_{i<j} \alpha_i t_i = \alpha_{j-1}t_j + \sum_{i=1}^{j-1}(\alpha_{i-1} - \alpha_i)t_i = \alpha_{j-1}t_j + \beta_{j-1}$. Then use $\alpha_{j-1}t_j = \beta_j - \beta_{j-1} + \alpha_j t_j$. □

LE is 2-competitive. For $T \in [t_j, t_{j+1})$: $\mathrm{LE}(T) = \mathrm{OPT}(t_j) + \alpha_j(T - t_j) + \beta_j = 2\beta_j + \alpha_j T \leq 2\,\mathrm{OPT}(T)$.

Consistency of LA(λ). Let $T = \hat{T} \in [t_{j^\star}, t_{j^\star+1})$. The algorithm reaches tier $j^\star$ at $\lambda t_{j^\star} \leq T$. It has not left that tier, since $T < t_{j^\star+1} < t_{j^\star+1}/\lambda$. Its cost is $\lambda \sum_{i<j^\star} \alpha_i(t_{i+1} - t_i) + \alpha_{j^\star}(T - \lambda t_{j^\star}) + \beta_{j^\star} = (1+\lambda)\beta_{j^\star} + \alpha_{j^\star}T \leq (1+\lambda)\mathrm{OPT}(T)$ by Lemma 1.

Robustness. Write $t'_i = \lambda t_i$ for $i \leq j^\star$ and $t'_i = t_i/\lambda$ for $i > j^\star$ for the algorithm's breakpoints. The algorithm is on tier a exactly when $t'_a \leq T < t'_{a+1}$, with cost $\mathrm{ALG}(T) = \sum_{i<a} \alpha_i(t'_{i+1} - t'_i) + \alpha_a(T - t'_a) + \beta_a$. Throughout we use three facts: OPT is nondecreasing and concave with $\mathrm{OPT}(0) = 0$; $\mathrm{OPT}(t_j) = \beta_j + \alpha_j t_j$; and $\mathrm{OPT}(T) = \beta_j + \alpha_j T$ on $[t_j, t_{j+1}]$. Let o be OPT's tier at time T. There are three cases.

Case $a < j^\star$. Every breakpoint passed was scaled by λ. Lemma 1 therefore gives $\sum_{i<a} \alpha_i(t'_{i+1} - t'_i) = \lambda\,\mathrm{OPT}(t_a)$, and $\mathrm{ALG} = (1+\lambda)\beta_a + \alpha_a T$. Since $T < \lambda t_{a+1} \leq t_{a+1}$, we have $o \leq a$. If $o = a$ then $\mathrm{ALG} \leq (1+\lambda)\mathrm{OPT}(T)$. If $o < a$ then $\lambda t_a \leq T < t_a$. By monotonicity and concavity, $\mathrm{OPT}(T) \geq \mathrm{OPT}(\lambda t_a) \geq \lambda\,\mathrm{OPT}(t_a)$, so $\mathrm{ALG} \leq (1+\lambda)(\beta_a + \alpha_a t_a) = (1+\lambda)\mathrm{OPT}(t_a) \leq \frac{1+\lambda}{\lambda}\mathrm{OPT}(T) = (1+1/\lambda)\mathrm{OPT}(T)$.

Case $a = j^\star$. The algorithm entered $j^\star$ at $\lambda t_{j^\star}$ and leaves it only at $t_{j^\star+1}/\lambda$. So again $\mathrm{ALG} = (1+\lambda)\beta_{j^\star} + \alpha_{j^\star}T$, but now T may exceed $t_{j^\star+1}$. If $T < t_{j^\star+1}$, the previous case applies verbatim. Otherwise $t_{j^\star+1} \leq T < t_{j^\star+1}/\lambda$, so $\mathrm{OPT}(T) \geq \mathrm{OPT}(t_{j^\star+1}) = \beta_{j^\star} + \alpha_{j^\star}t_{j^\star+1} \geq \beta_{j^\star}$, and

$$\begin{aligned}\mathrm{ALG} &\leq (1+\lambda)\beta_{j^\star} + \alpha_{j^\star}t_{j^\star+1}/\lambda = \big(1 + \lambda - \tfrac{1}{\lambda}\big)\beta_{j^\star} + \tfrac{1}{\lambda}\mathrm{OPT}(t_{j^\star+1}) \\ &\leq \beta_{j^\star} + \tfrac{1}{\lambda}\mathrm{OPT}(t_{j^\star+1}) \leq (1+1/\lambda)\mathrm{OPT}(T),\end{aligned}$$

using $\lambda \leq 1$.

Case $a > j^\star$. The algorithm reached $j^\star$ through λ-scaled breakpoints. It sat on $j^\star$ from $\lambda t_{j^\star}$ to $t_{j^\star+1}/\lambda$. It then followed $1/\lambda$-scaled breakpoints and entered a at t_a/λ. Applying Lemma 1 to the two runs of breakpoints gives $\sum_{i<j^\star} \alpha_i(t_{i+1} - t_i) = \mathrm{OPT}(t_{j^\star})$ and $\sum_{j^\star<i<a} \alpha_i(t_{i+1} - t_i) = \mathrm{OPT}(t_a) - \mathrm{OPT}(t_{j^\star+1})$, so

$$\begin{aligned}\mathrm{ALG} &= \lambda\,\mathrm{OPT}(t_{j^\star}) + \alpha_{j^\star}\big(\tfrac{t_{j^\star+1}}{\lambda} - \lambda t_{j^\star}\big) + \tfrac{1}{\lambda}\big(\mathrm{OPT}(t_a) - \mathrm{OPT}(t_{j^\star+1})\big) + \alpha_a\big(T - \tfrac{t_a}{\lambda}\big) + \beta_a \\ &= \lambda\beta_{j^\star} - \tfrac{1}{\lambda}\beta_{j^\star} + \tfrac{1}{\lambda}\beta_a + \beta_a + \alpha_a T \\ &= (1+1/\lambda)\beta_a + \alpha_a T - \big(\tfrac{1}{\lambda} - \lambda\big)\beta_{j^\star} \leq (1+1/\lambda)\beta_a + \alpha_a T,\end{aligned}$$

where the second line substitutes $\mathrm{OPT}(t_{j^\star}) = \beta_{j^\star} + \alpha_{j^\star}t_{j^\star}$, $\mathrm{OPT}(t_{j^\star+1}) = \beta_{j^\star} + \alpha_{j^\star}t_{j^\star+1}$, and $\mathrm{OPT}(t_a) = \beta_a + \alpha_a t_a$. Since $T \geq t_a/\lambda \geq t_a$, we have $o \geq a$. If $o = a$, then $\mathrm{OPT}(T) = \beta_a + \alpha_a T$ and $\mathrm{ALG} \leq (1+1/\lambda)\mathrm{OPT}(T)$. If $o > a$, then $t_{a+1} \leq T < t_{a+1}/\lambda$, so $\alpha_a T < \alpha_a t_{a+1}/\lambda$ and

$$\mathrm{ALG} \leq (1+1/\lambda)\beta_a + \tfrac{1}{\lambda}\alpha_a t_{a+1} = \beta_a + \tfrac{1}{\lambda}\mathrm{OPT}(t_{a+1}) \leq (1+1/\lambda)\mathrm{OPT}(T),$$

using $\mathrm{OPT}(T) \geq \mathrm{OPT}(t_{a+1}) \geq \beta_a$. In every case $\mathrm{ALG} \leq (1+1/\lambda)\mathrm{OPT}(T)$. This proves the robustness half of Theorem 1. With $T = \hat{T}$, the first case (or the first sub-case of the second) applies with $o = a = j^\star$ and gives $(1+\lambda)$, which is the consistency half. The bound is attained: the empirical per-gap maxima in Table 5 equal $1 + 1/\lambda$ for every λ, at the reference and at the measured parameters alike.

B TRUST-PARAMETER SWEEP AND ROBUSTNESS CURVE

Figure 5 corrupts exact predictions with multiplicative log-normal noise. Every λ sits at or below its consistency bound at $\sigma = 0$. At $\sigma = 4$, where predictions are off by a factor of $e^4 \approx 55$ on average, $\lambda = 0.25$ has degraded from 1.17 to 1.75. That is LE's level, not its worst case of 5. The worst case in Theorem 1 is adversarial, and random error is far kinder. The real predictor's errors are not random. They concentrate on the gaps whose length decides the placement. This is why it sits at 1.68 with $\lambda = 0.5$, while random noise of the same log-scale would sit near 1.45. Table 5 sweeps λ with the learned predictor.

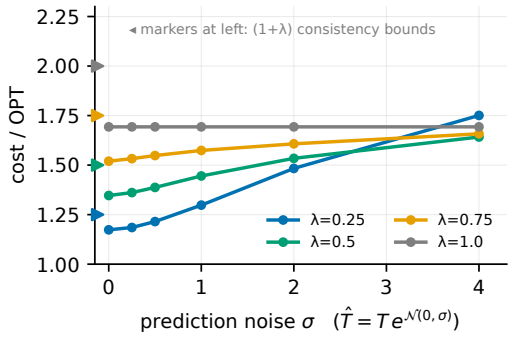

Figure 5: Graceful degradation. Predictions are corrupted with log-normal noise of scale σ. Each curve starts at or below its $(1 + \lambda)$ consistency bound, and at $\sigma = 4$ (a factor of e^4) it has not yet reached the prediction-free policy's 1.69.

Table 5: Consistency-robustness trade-off on held-out gaps with the learned predictor. The robustness bound, per-gap max $\leq 1 + 1/\lambda$, is met with equality at every λ. The consistency bound $1 + \lambda$ applies to exact predictions; with them the p_{90} is 1.500 at $\lambda = 0.5$. The p_{90} with the real predictor shows how far the point predictions are from that.

λ	0.25	0.5	0.75	1.0
cost/OPT, LA (point prediction)	1.81	1.68	1.67	1.69
cost/OPT, LA-Bayes (quantiles)	1.59	**1.57**	1.62	1.69
per-gap p_{90}, LA-Bayes	1.77	1.51	1.75	2.00
per-gap max (bound $1 + 1/\lambda$)	5.00 (5)	3.00 (3)	2.33 (2.33)	2.00 (2)

C PREDICTOR FEATURES

Waiting-on-tool indicator; $\log(1 + \#\text{tools})$; $\log(1 + \text{round})$; $\log(1 + \text{session age})$; $\log(1 + K)$; $\log(1 + \text{appended tokens})$; $\log(1 + \text{output tokens})$; $\log(1 + \text{previous gap})$; previous gap was human; provider; $\log(1 + \text{tool-call argument characters})$ (max and sum over the round's calls); $\log(1 + \text{assistant message characters})$; $\log(1 + \text{characters returned by the previous round's tools})$; $\log(1 + \text{input events})$; previous round's tools errored; hour of day (sine, cosine) and weekend, from the request timestamp; one-hot tool signature over the 24 most frequent signatures plus "other". The trace's per-tool wall latency is excluded (§4.2). The argument-size, message-length, previous-result and clock features add less than one point of within-2× accuracy over the first ten features. A serving system that only knows the tool signature and the session's history therefore loses little. We fit three `HistGradientBoostingRegressors` (quantile loss $\tau \in \{0.1, 0.5, 0.9\}$, 300 iterations, learning rate 0.08, 31 leaves) on $\log(1 + T)$. Users are split 31/12 with seed 0.

D ABLATION: CONFIDENCE-ADAPTIVE λ

Setting $\lambda_i = \lambda_{\min} + (1-\lambda_{\min}) \min(1, \log(q_{90}/q_{10})/\log w)$ with $w \in \{10, 100\}$ gave 1.69 and 1.68, no better than LE. Wider intervals occur on human gaps. The point prediction's ordinal information ("long") is still useful there, so discounting those gaps discards signal. Two alternatives work: a per-type λ chosen on validation (§4.3), which ends up trusting human predictions and not tool predictions, and the Bayes variant, which uses the interval rather than its width.

E COST PARAMETERS: REFERENCE SET, MEASURED SETS, AND EVERY TIER MEASUREMENT

Table 6 lists the parameter sets behind the simulations: the reference 8B/H100-class set of Sections 5.3–5.4 and the measured sets of Section 5.5. In all sets $\gamma = 1$, reads are not pipelined with the PCIe leg, and c_1 and c_2 are the rent ratios of the reference set. The configurations of Figure 4 are analytic variations of the reference set (`sim/surface.py`). They are: 8B/H100 as the reference; 8B/A100-40GB with a 20 GiB budget and $a = 2.3 \times 10^{-5}$, $b = 9 \times 10^{-11}$; 70B/4×H100 with 320 KiB/token, 150 GiB and $a = 2.5 \times 10^{-5}$, $b = 10^{-10}$; and 70B/8×A100 with 160 GiB and $a = 5.8 \times 10^{-5}$, $b = 2.3 \times 10^{-10}$. Storage read bandwidth is swept over 2–50 GB/s, with writes at 60% of reads. The measured A100 prefill (Table 3) is about 3× costlier than that sweep assumed. The measured 72B prefill on four PCIe H100s is 11× costlier per token than the 70B/4×H100 configuration assumed ($a = 2.8 \times 10^{-4}$ against 2.5×10^{-5}; b is 6.4× larger). This is why Section 5.5 is reported separately rather than folded into the sweep. For tensor-parallel rows, the KV budget is the sum over the four GPUs, and the PCIe leg is charged at four times the per-GPU rate, because each rank restores its own shard. The storage tier is the single-stream rate measured from the node.

Pipelined reads. The tables charge a restore serially, $1/B_{\text{eff}} = 1/B_{\text{tier}} + 1/B_{\text{PCIe}}$. This is what a connector does when it issues the read and the host-to-device copy in sequence. Overlapping chunked reads with the copy gives $B_{\text{eff}} = \min(B_{\text{tier}}, B_{\text{PCIe}})$ plus one chunk of PCIe latency (36 MiB at 22 GB/s: 1.7 ms). PCIe was 3–30× faster than every tier we measured, so the PCIe leg drops out and $K^\star = (\text{bpt}/B_{\text{tier}} - a)/b$. The shift is small where the tier is slow and grows with the tier's speed. The A100's 10 GbE NFS mount moves from 34K to 30K tokens. The H100's 10 GbE mount moves from 147K to 137K, and the parallel file system's single stream from 250K to 240K. A 25 GbE-class mount on the H100 moves from 29–32K to 19–22K, and the parallel file system's eight-stream rate from 28K to 18K. The tier-side requirement for an H100 at 32K tokens is 2.8 GB/s serial and 2.5 GB/s pipelined (1.24 and 1.18 at 128K), and 1.09 against 1.07 for the 72B model. No tier changes sides. Every tier that wins at all lengths still does, and the 72B crossovers move by under 3%. The harness reports both (`effective_bandwidth(..., pipelined=True)`). Table 8 lists every tier measurement: each client, each condition, and the saturation behaviour over concurrency. Table 7 varies the host-memory rent c_1 at the measured H100 parameters. The measured prefill points, the PCIe transfers at four chunk sizes, and the per-run spreads are in the released result files.

Table 6: Parameter sets used by the simulator.

set	bpt B/tok	budget GiB	a s/tok	b s/tok^2	PCIe h2d/d2h GB/s	tier w/r GB/s	c_1/c_2
reference (8B/H100-class, App. E)	131,072	60	1.00×10^{-5}	4.00×10^{-11}	20.0/20.0	3.00/5.00	0.05/0.002
H100 80GB PCIe, 8B (cluster C), parallel FS	147,456	58	3.69×10^{-5}	6.76×10^{-10}	22.3/28.3	0.36/0.74	0.05/0.002
4×H100 80GB PCIe, 72B (cluster C), parallel FS	327,680	156	2.84×10^{-4}	6.42×10^{-10}	27.7/28.5 (×4)	0.36/0.74	0.05/0.002
A100 40GB PCIe, 8B (cluster A), all-flash NFS 1	147,456	21	7.33×10^{-5}	1.89×10^{-9}	19.3/16.3	1.06/1.14	0.05/0.002
L40 48GB, 8B (cluster A), all-flash NFS 1	147,456	26	1.33×10^{-4}	2.59×10^{-9}	24.1/26.1	1.86/2.84	0.05/0.002

F REPRODUCING THE RESULTS

The trace pipeline is three commands: `python analysis/extract_gaps.py <trace.jsonl.gz>`; `python sim/predictor.py`; and `python sim/simulate.py && python sim/surface.py && python sim/surface.py --pressure && python sim/figures.py`. The cluster measurements run `kvtier submit characterize --engine vllm --model <dir>` on a GPU node (prefill, PCIe,

Table 7: Sensitivity to the host-memory rent c_1 at the measured H100 parameters (parallel file system tier). Costs are relative to OPT on all held-out gaps. The tuned policy and its LE reference are on the disjoint test half. As c_1 rises, the DRAM tier leaves OPT, and the HBM-versus-drop decision moves to where the human/tool distinction settles it. At $c_1 = 1$ the validation choice of λ_{human} no longer transfers to the test users.

c_1	OPT in DRAM	OPT drops	LE	LA λ=0.5	LA-Bayes	exact pred.	LA tuned ($\lambda_{\text{tool}}, \lambda_{\text{human}}$)	saving vs LE
0.02	26%	1%	1.53	1.59	1.53	1.27	1.51 (1.0, 1.0)	0.0%
0.05	24%	2%	1.47	1.53	1.48	1.24	1.42 (1.0, 1.0)	0.0%
0.1	23%	3%	1.44	1.51	1.46	1.22	1.39 (1.0, 1.0)	0.0%
0.2	20%	5%	1.43	1.49	1.43	1.22	1.37 (1.0, 0.75)	0.9%
0.5	11%	11%	1.70	1.62	1.53	1.35	1.59 (1.0, 0.5)	9.9%
1.0	0%	13%	1.72	1.93	1.71	1.36	2.01 (1.0, 0.25)	-12.9%

Table 8: Every tier measurement. Storage-only jobs use the Llama-3.1-8B geometry with 32 MiB chunks; the sweeps run inside the GPU jobs use the Qwen3-8B geometry with 36 MiB chunks. "Read max" is the largest aggregate over 1–8 concurrent sessions, with the concurrency at which it occurred. Restore is the median single-session time for a 4–4.5 GiB session. p_{99} is the chunk-latency tail. Clients are labeled per cluster. Cluster A's `cpu1` is a data-mover node, and `gpu1`/`gpu2` are the A100 and L40 nodes. Cluster B's `cpu1` and `cpu2` are two CPU nodes with 25 GbE storage links, measured concurrently by two copies of the storage job. Cluster C's `gpu1` is the H100 node and `cpu1` is a compute node. The cold run at concurrency 1 on cluster B's second NFS system failed on both nodes with a stale NFS handle and is omitted. On cluster C the node-local drives had too little free space for more than one 4.5 GiB session, so only concurrency 1 is reported there.

cluster	client	tier	cond.	read c1 GB/s	write c1 GB/s	read max (conc.) GB/s	restore 4.5 GiB s	p_{99} chunk ms
cluster C	cpu1	research NFS	warm	1.17	1.02	1.23 (2)	3.66	42
cluster C	cpu1	research NFS	cold	1.03	1.00	1.20 (4)	4.16	63
cluster C	cpu1	parallel FS	warm	0.74	0.30	1.24 (8)	5.76	119
cluster C	cpu1	parallel FS	cold	0.22	0.33	0.78 (4)	19.22	193
cluster C	cpu1	node-local SSD	warm	0.45	0.20	0.45 (1)	9.52	75
cluster C	gpu1	research NFS	warm	1.30	1.45	1.92 (8)	3.69	215
cluster C	gpu1	parallel FS	warm	0.74	0.36	3.02 (8)	6.49	55
cluster C	gpu1	node-local SSD	warm	6.26	3.71	6.26 (1)	0.77	6
cluster A	cpu1	all-flash NFS 1	warm	1.13	1.00	1.18 (8)	3.80	32
cluster A	cpu1	all-flash NFS 1	cold	1.10	1.02	1.18 (4)	3.90	69
cluster A	cpu1	node-local SSD	warm	1.05	0.33	1.12 (4)	4.10	33
cluster A	cpu1	node-local SSD	cold	1.09	0.52	1.12 (4)	3.93	31
cluster A	cpu1	all-flash NFS 2 (older)	warm	1.12	0.87	1.18 (4)	3.83	32
cluster A	cpu1	all-flash NFS 2 (older)	cold	1.11	0.83	1.18 (4)	3.88	35
cluster A	gpu1	all-flash NFS 1	warm	1.14	1.06	1.18 (8)	4.23	33
cluster A	gpu1	node-local SSD	warm	1.04	0.90	1.07 (8)	4.62	38
cluster A	gpu1	all-flash NFS 2 (older)	warm	1.12	0.79	1.17 (8)	4.30	42
cluster A	gpu2	all-flash NFS 1	warm	2.84	1.86	2.96 (8)	1.69	14
cluster A	gpu2	node-local SSD	warm	0.66	0.39	0.77 (8)	7.28	58
cluster A	gpu2	all-flash NFS 2 (older)	warm	2.21	0.89	2.92 (8)	2.18	31
cluster B	cpu1	all-flash NFS 1	warm	2.45	1.69	2.94 (8)	1.75	54
cluster B	cpu1	all-flash NFS 1	cold	1.87	1.70	2.81 (4)	2.30	53
cluster B	cpu1	node-local SSD	warm	5.00	1.44	5.16 (4)	0.86	7
cluster B	cpu1	node-local SSD	cold	4.99	1.43	5.16 (4)	0.86	7
cluster B	cpu1	all-flash NFS 2 (older)	warm	1.48	1.00	2.15 (8)	2.90	31
cluster B	cpu1	all-flash NFS 2 (older)	cold	–	–	1.88 (4)	–	–
cluster B	cpu2	all-flash NFS 1	warm	1.98	1.76	2.93 (8)	2.17	53
cluster B	cpu2	all-flash NFS 1	cold	1.94	1.62	2.79 (4)	2.21	53
cluster B	cpu2	node-local SSD	warm	4.17	1.47	5.16 (8)	1.03	12
cluster B	cpu2	node-local SSD	cold	4.99	1.44	5.16 (4)	0.86	7
cluster B	cpu2	all-flash NFS 2 (older)	warm	1.45	0.92	2.04 (8)	2.96	40
cluster B	cpu2	all-flash NFS 2 (older)	cold	–	–	1.90 (4)	–	–

tiers) and `kvtier submit storage --cold-gib 64` on a CPU node (warm and cold tiers) on each cluster. Then `python sim/fill_params.py results/ --simulate && python sim/fill_paper_tables.py && python sim/fig_breakeven.py` regenerates Tables 3, 4, 6, 8 and Figure 3 from the result files.