REMEMBER OR RECITE? PRICING STATE RESTORATION FOR ON-DEVICE MULTI-AGENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent large language model (LLM) workflows are moving onto personal devices, where several roles share one base model through low-rank adaptation (LoRA) adapters and take turns, each with its own key–value (KV) cache. Device memory holds only a few such sessions, so evicted state must be written to flash or recomputed later, a cost that dominates the workflow's latency and energy. Existing serving systems price this choice as data centers do, assuming symmetric memory and independent requests. Our measurements show that both assumptions fail on a device: flash writes over 130× slower than it reads, and the workflow graph already says which turns are ready, mergeable, and finished, so much state need never be written. We propose RECITE, an on-device runtime that turns these two facts into three closed-form rules: a per-tier save-or-recompute threshold in which session length cancels, liveness read off the workflow graph, and batched decoding of ready same-role turns under a memory-traffic ceiling. From traces and device constants alone, the rules predict opposite decisions on the two flash tiers of one board and where batching hurts. On Jetson boards with two model families, RECITE cuts debate and voting makespan 2.1–2.5× and energy 2.3–2.7× at equivalent accuracy against the strongest of five reference configurations, and runs up to 6.8× faster than the stateless ones across all workloads. Where both arms of a decision are measured, the rule picks the faster arm every time, and the cost model predicts the restore share of held-out configurations with $R^2 = 0.99$.

1 INTRODUCTION

Large language model (LLM) agents are moving from single-turn queries to workflows in which several roles plan, act, and check one another: debate and voting ensembles (Du et al., 2024; Wang et al., 2023), planner–actor–critic loops (Wu et al., 2024; Hong et al., 2024), and graphical user interface (GUI) agents that operate a phone on the user's behalf (Li et al., 2024; Zhang et al., 2025a). Privacy pushes these workflows onto the devices people carry, since roles that reason over a user's messages, files, and accounts cannot ship that context to a cloud (Shen et al., 2025b; Chung et al., 2026). On the device the roles share one base model, specialize through low-rank adaptation (LoRA) adapters (Hu et al., 2022), and execute in turn along a workflow directed acyclic graph (DAG), so a turn needs the base weights, the role's adapter, and the session's key–value (KV) cache resident at once. Only a few such sessions fit in memory; the rest are evicted, and their KV is written to flash or recomputed when they return. Restoring that state has become the dominant cost of on-device multi-agent inference, and how the restoration is priced decides how much of that cost is paid.

Two lines of work handle KV that does not fit. Serving engines and KV stores move state across memory tiers or recompute it (Kwon et al., 2023; Abhyankar et al., 2024; Gao et al., 2024; Liu et al., 2024a; 2025; Qin et al., 2025; Nian et al., 2026; Qiu et al., 2026; Jeong & Ahn, 2025; Jin et al., 2025; Chen et al., 2025; Meng et al., 2026), and agent-aware servers prefetch, pin, and release KV along the execution graph (Pan et al., 2025; Li et al., 2025; Bian et al., 2025; Guo et al., 2026), both in data-center memory, where a write costs about as much as a read. On-device servers handle stateless requests or a single agent (Shen et al., 2025b; Chung et al., 2026; Zhang et al., 2026a; Shkolnikov, 2026). The closest systems show the gap from both sides. Continuum (Li et al., 2025) pins KV for a time-to-live (TTL) and releases it only when the whole program finishes, but writes every evicted session first; KVSwap (Zhang et al., 2026a) serves a single context larger than memory and never meets a second session; EdgeLoRA (Shen et al., 2025b) serves many adapters on a Jetson but

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.

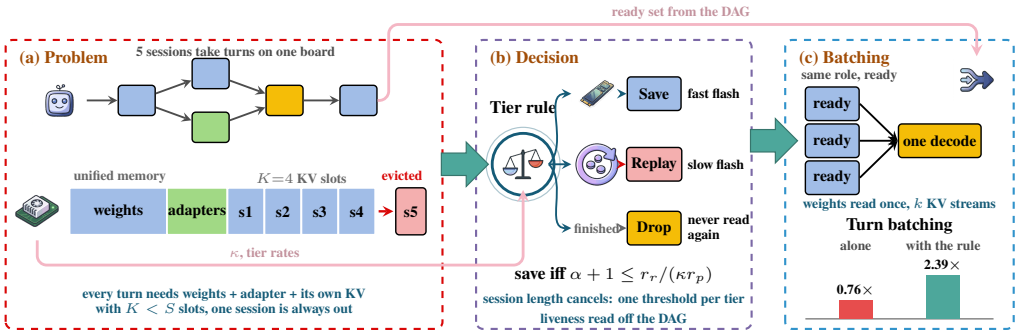


Figure 1: (a) A turn needs weights, adapter, and KV resident, so with K slots for $S > K$ sessions something is always evicted. (b) Evicted state is saved, replayed, or dropped for good, decided by the tier rule and by liveness read off the DAG. (c) Ready same-role turns decode as one batch under a memory-traffic ceiling on the merge width.

discards a session's KV on every eviction. None prices the write that an eviction triggers against the tier's write–read asymmetry, and none consults the workflow graph, which knows that a role session finishes long before the workflow does. Each of them therefore remembers a session's state or recites its history by default, not by price. Pricing that choice on the device raises two challenges.

The first challenge is deciding, at each eviction, whether a session's state should be written at all and, if so, whether saving or recomputing it is cheaper. The answer depends on the tier's write and read rates, the model's prefill rate, and whether the session will ever return. A request stream knows none of these (Kwon et al., 2023; Abhyankar et al., 2024; Gao et al., 2024; Li et al., 2025), so existing systems settle the question with one policy for every eviction, and on a device either fixed answer is wrong on one of its two flash tiers. Saving everything pays the slow write of embedded MultiMediaCard (eMMC) flash on state that no turn will read, and recomputing everything replays whole histories on NVM Express (NVMe) storage, where saving is cheap. *The second challenge is that batching, the standard throughput move, interacts with eviction.* Decoding ready same-role turns as one batch reads the weights once and lifts decode throughput 2.5×, but merged groups hold more sessions resident, and on slow flash the extra evictions turn write-back that the page cache used to hide into synchronous writes. Batching alone therefore *loses* 24% on debate over eMMC and wins 2.39× on the same cell once eviction follows a rule. The two mechanisms have to be designed together, under a cost model that predicts when each of them pays.

In this paper we present RECITE, an on-device runtime in which state restoration is a predicted decision, not a default. Its core idea is that the cost of a wrong placement depends on four coordinates a deployment already has: how long the turn is, how fast the tier writes and reads, how many sessions fit, and how many others run before the session returns. The right action can therefore be predicted before the turn runs, and three rules follow (Figure 1). At each eviction one number per storage tier decides between saving and recomputing, because the session's length divides out of the comparison. A session whose DAG has no further turn is dropped without being written. Ready turns of the same role are merged only as far as a memory-traffic bound allows. The rules take only traces and device constants, so each of their predictions can be tested against the cell it was made for.

The contributions are as follows.

1. **Pricing state restoration on a device.** Two facts that request-stream cost models omit, the flash tier's write–read asymmetry and the liveness the workflow graph exposes, turned into a four-coordinate model of restore share, an identity that bounds any restore mechanism by $1/(1-R)$, and a lemma that predicts restore counts from the trace alone (Section 3).
2. **A save-or-recompute rule in which the session cancels.** One threshold per storage tier, computed from device constants, that decides between saving and recomputing at each eviction, and liveness-first eviction that drops finished sessions without writing them (Section 3.3).
3. **DAG-aware turn batching under a memory-traffic ceiling,** with the finding that it turns hidden write-back into synchronous writes, so merging pays only with the restore rule (Section 3.4).
4. **The RECITE runtime and its evaluation.** Built on llama.cpp and evaluated on three Jetson boards, a phone, two model families, a live Android loop, and a published edge server, every

prediction registered before its cell ran: 2.1–2.5× makespan and 2.3–2.7× energy on parallel patterns, parity where the rules predict none, and $R^2 = 0.99$ on held-out restore share (Section 4).

2 Related Work

KV State Management for LLM Serving. Work that manages KV state beyond memory falls into two lines. Serving engines and KV stores swap, persist, compress, or disaggregate KV across tiers and recompute when retrieval is slower (Kwon et al., 2023; Abhyankar et al., 2024; Gao et al., 2024; Liu et al., 2024a; 2025; Qin et al., 2025; Nian et al., 2026; Qiu et al., 2026; Jeong & Ahn, 2025; Jin et al., 2025; Chen et al., 2025; Meng et al., 2026). They charge both directions of a symmetric GPU–CPU swap (Abhyankar et al., 2024) or hide write-back behind compute (Gao et al., 2024). Agent-aware servers exploit the execution graph instead. They prefetch, pin, share, or repair KV across the steps and roles of a workflow (Pan et al., 2025; Li et al., 2025; Ye et al., 2025; Liu et al., 2026; Bian et al., 2025; Guo et al., 2026; Ni et al., 2026; Zheng et al., 2026; Zhang et al., 2026b; Rodionov et al., 2025; Hsu et al., 2025; Woo et al., 2026a; Jeon et al., 2026), and Continuum frees KV when a program's last request finishes (Li et al., 2025). All of them keep state in GPU or CPU memory, where the two directions of a swap cost about the same. On a mobile flash tier the write side is two orders of magnitude slower and cannot always be hidden, and none of these systems decides, before the program ends, whether a session leaving memory needs to be written.

On-Device and Multi-Adapter Serving. On-device serving optimizes a single model or a single agent. EdgeLoRA serves multi-tenant adapters for stateless requests (Shen et al., 2025b), CoServe and D2MoE swap expert weights on a router's fixed order (Suo et al., 2025; Wang et al., 2025), S-LoRA and Punica batch adapter requests through segmented kernels (Sheng et al., 2024; Chen et al., 2024), and RadixAttention reuses prefixes without regard to storage tiers or turn structure (Zheng et al., 2024). KVSwap serves one context that does not fit memory (Zhang et al., 2026a), Agent-X accelerates a single agent pipeline (Chung et al., 2026), Agent Memory persists per-agent KV on a laptop (Shkolnikov, 2026), and Halo consolidates agent DAGs on a server GPU (Shen et al., 2025a). Agent frameworks fix the DAG and treat turn order as application logic (Wu et al., 2024; Hong et al., 2024; Yao et al., 2023; Shinn et al., 2023; Li et al., 2023; Qian et al., 2024; Zhang et al., 2025a; Wang et al., 2024). None of them combines the three decisions an evicted session forces on a device, whether to write it, whether to save or recompute it, and whether to merge its next turn with others, and none prices the first two against the tier's write–read asymmetry.

3 Pricing State Restoration

This section prices one turn of a non-resident session from quantities the device and the workflow already provide: the storage tier writes far slower than it reads, and the workflow graph tells which sessions will return. The three rules of RECITE follow from that price; proofs are in Appendix A.

3.1 Preliminaries

Setting. A device serves S concurrent *sessions*, each a workflow instance whose turns are nodes of a DAG. Every turn belongs to a *role*, and roles share one base model and differ by a LoRA adapter. A turn of session s needs the base weights, its role's adapter, and the session's KV cache of n_s tokens, $b_s = \kappa n_s$ bytes at a measured κ bytes per token, resident in unified memory. Beside weights, adapters, and buffers, that memory holds $K = \lfloor (M - M_{\text{fixed}})/(\kappa L_{\max}) \rfloor$ slots provisioned for the longest context $L_{\max}$. That is four 32K-token slots on the 32 GB board and barely one on an 8 GB device. An evicted session's state is either written to storage and read back, or dropped and recomputed by replaying its history through prefill.

Restore Share. For one turn of a non-resident session s, write R for the share of wall clock spent restoring state,

$$R_\pi \approx \frac{\phi(K)\, C_\pi(s)}{\phi(K)\, C_\pi(s) + T_{\text{turn}}(s)}, \qquad C_{\text{save}}(s) = \frac{b_s}{r_w} + \frac{b_s}{r_r}, \qquad C_{\text{adapt}}(s) = \min\Big(C_{\text{save}}(s),\ \frac{n_s}{r_p}\Big), \tag{1}$$

where r_p is the prefill rate, T_{turn} the turn's own compute, and $\phi \in [0, 1]$ the fraction of turns that find their session evicted under policy π. Four coordinates govern the overhead. *Turn length* sets

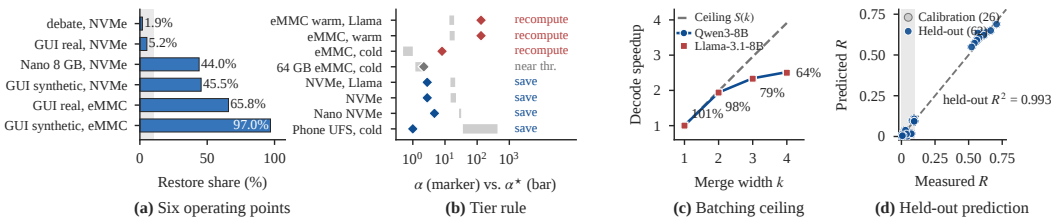


Figure 2: Closed forms against measurement. (a) Restore share at six operating points under save-everything; shaded, the neutral zone where no restore mechanism pays more than 1.11×. (b) The tier rule on every board, tier, and family: asymmetry α against the $\alpha^\star$ interval. (c) Decode speedup of both families against the ceiling $S(k)$. (d) Predicted against measured restore share for the calibration and the held-out cells.

the denominator, so a long reasoning turn amortizes any restore and a short action turn amortizes nothing. *Storage rate* sets the numerator as two numbers, not one, because averaging r_w and r_r would misjudge every eviction on eMMC. *Residency capacity* K and *reuse distance* D_t, the number of distinct other sessions run between a session's consecutive turns, enter ϕ together.

Lemma 1 (Reuse distance)**.** *Under least-recently-used (LRU) eviction over K equal-size slots, a turn of session s finds s resident iff fewer than K distinct other sessions ran since s's previous turn, so $\phi(K) = \Pr[D_t \geq K]$ and $\phi(K) - \phi(K{+}1) = \Pr[D_t = K]$.*

One more slot removes exactly the revisits at distance K. The five-workflow GUI trace has $D_t = 4$ on two of its revisits, so four slots should miss exactly twice, and the three-workflow trace never, since its $D_t \leq 2$; Section 4.4 confirms both.

3.2 Analysis

Write–Read Asymmetry. Writing r_w and r_r for a tier's measured write and read rates, the KV file path a runtime takes on the 32 GB board reads at 2116 MB/s and writes at 16 MB/s on eMMC, a 132× asymmetry, against 2277/827 MB/s on NVMe. A serving engine that averages the two rates, or prices only the read it will later perform, misjudges every eviction on eMMC by an order of magnitude. Which side is paid depends on the policy. Saving pays the write now and the read later, and recomputing pays neither, only a prefill.

Liveness in the DAG. A request stream never knows whether a session will return, but a DAG does. Every voting session is single-turn and every debate session ends after its second turn, so under save-everything nearly every eviction writes state that no turn will read, 13.3 and 16.8 GB on the two traces. A runtime that reads liveness off the DAG can drop that state instead of writing it to flash.

When Management Pays. Figure 2a places six operating points on one board and model with two resident slots under save-everything. Restore share reads 1.9% for a long-turn debate workload on NVMe against 97.0% for a short-turn GUI workload on eMMC, where a synthetic every-turn-switch trace sets that ceiling and the real GUI traces reach 65.8%. The spread is two orders of magnitude, and whether any restore mechanism is worth its overhead follows from one identity.

Proposition 1 (Restore-accounting identity)**.** *Let a reference run spend a share R of its makespan restoring state. A restore-path mechanism that removes a fraction η of that time and adds overhead ϵ, both as shares of the reference makespan, while leaving all other execution unchanged, changes makespan by $S = 1/(1 - \eta R + \epsilon)$; it pays iff $\eta R > \epsilon$, and no restore-path mechanism can exceed the bound $1/(1 - R)$.*

From the trace and the tier rates alone the identity puts the five-workflow eMMC cell at $S = 2.15$–$2.21\times$, and Section 4.3 measures 2.10–2.21×. At the 1.9% and 5.2% cells even $\eta = 1$ yields 1.019× and 1.055×, a neutral zone where RECITE disables restore-specific work. Equation (1) assumes every evicted session returns; Section 3.3 prices the return probability, read off the DAG.

3.3 SAVE OR RECOMPUTE

Eviction is where the write side is paid or avoided. RECITE chooses per evicted session between saving KV for later read-back and discarding it for recompute, comparing

$$C_{\text{save}} = \frac{b}{r_w} + \frac{b}{r_r} \qquad \text{against} \qquad C_{\text{recomp}} = \frac{n}{r_p}, \tag{2}$$

with r_p trailed online as an exponentially weighted average of recent prefills. Write $\alpha = r_r/r_w$ for the tier's asymmetry between reading and writing.

Proposition 2 (Tier rule)**.** *Fix a tier and a prefill rate r_p. Under Equation (2), saving beats recomputing if and only if*

$$\alpha + 1 \;\le\; \frac{r_r}{\kappa\, r_p}, \tag{3}$$

in which the session's length n has cancelled. With return probability q, the rule becomes $\alpha + q \le q\, r_r/(\kappa r_p)$: at $q = 1$ it is Equation (3), at $q = 0$ it never holds.

Context length survives only through r_p, which RECITE trails. What remains is one number per tier and model, $\alpha^\star = r_r/(\kappa r_p) - 1$, above which recomputing always wins. On the 32 GB board the measured asymmetry sits well below $\alpha^\star$ on NVMe and well above it on eMMC, and no prefill rate or I/O coding crosses either threshold (Corollary 1). Figure 2b instantiates the rule on every board, tier, and family we measured. One case is near the threshold: the 64 GB board's eMMC has $\alpha = 2.2$ against an $\alpha^\star$ of 1.2–2.1, so its decision rides on the trailed r_p. Solved for the prefill rate, the rule reads recompute until r_p falls below $r_p^\star$, which Appendix E.3 tests prospectively over a 2K–32K context sweep. The cancellation assumes whole-session save and replay; prefix reuse keeps it and incremental save would not.

Liveness First. Equation (2) presumes the session comes back; RECITE instead reads q off the DAG. A session with no future turn is evicted before any live one and dropped unsaved, where $q_s = 0$ only on an explicit terminal event and everything unresolved defaults to $q_s = 1$.

3.4 DAG-AWARE TURN BATCHING

The first two rules price sessions that leave memory; the third schedules the turns that stay. Decode at batch size one is bandwidth-bound on these boards. Every generated token re-reads the weights, and merging ready same-role turns into one batched decode reads them once. Merging itself is standard continuous batching. What the DAG adds is certainty about which turns are ready, share an adapter, and are finished, so same-role width is scheduled, not accidental. Let M be the weight bytes a decode step reads, L the context length, and κ_{mem} the KV bytes read per token.

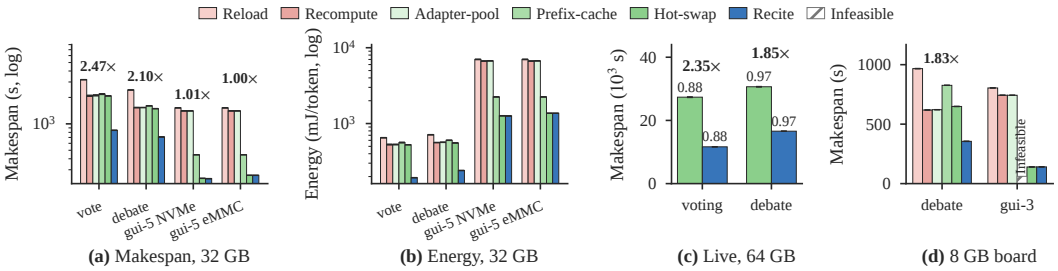
Proposition 3 (Memory-traffic ceiling and width)**.** *Under a bandwidth-bound model of decode, merging k ready turns cannot exceed*

$$S(k) \;=\; \frac{k\,(M + L\kappa_{\text{mem}})}{M + kL\kappa_{\text{mem}}}, \tag{4}$$

which approaches k as $M/(L\kappa_{\text{mem}})$ grows. Under a per-step latency budget D the throughput-maximizing width is the largest feasible one, $w^\star = \max\{k \le w_{\text{ready}} : T_b(k) \le D\}$, where $T_b(k) = T_0 + (M + kL\kappa_{\text{mem}})/B$ is the merged step time at bandwidth B with fixed overhead T_0.

The model counts bytes only, so $S(k)$ is a ceiling on the memory-traffic component of the speedup. Weights outweigh the per-sequence KV read by an order of magnitude at our workloads' context lengths, so Equation (4) is close to k; the runtime fixes w and measures the attainment and the tail-latency price in Section 4.4.

Ordering. Turn order within the DAG's freedom is a precedence-constrained scheduling problem with sequence-dependent setups, NP-hard even in simpler forms (Pinedo, 2022; Allahverdi et al., 2008). RECITE seeds each group with the earliest ready turn and expands it with same-role ready turns in turn order up to a width $w \le K$, so a group is always co-resident. With one resident slot a session-affine greedy, which keeps the current session while it has a ready turn, is optimal: it activates every session once at zero restore cost (Lemma 2). RECITE's in-order seeding coincides with it only when turn identifiers are session-contiguous, and the guarantee is specific to that case and to $K = 1$. Under memory pressure a rolling twelve-turn solver improves on it by at most 10.5% (Appendix C).



(a) Makespan, 32 GB **(b)** Energy, 32 GB **(c)** Live, 64 GB **(d)** 8 GB board

Figure 3: Main results. (a, b) Makespan and energy per token on the 32 GB board on a log scale; factors are RECITE over the strongest reference configuration. (c) Live voting and debate on the 64 GB board, accuracy above each bar. (d) The 8 GB board; the hatched stub is the one infeasible cell, where the unbudgeted prefix pool overflows.

3.5 THE RECITE RUNTIME

The per-turn loop (Algorithm 1) composes the rules. While ready turns remain, it seeds a group, evicts for its members with finished sessions first and the tier rule on live ones, clones the longest resident same-role prefix into each admitted session, binds the role's adapter, and runs the group. Adapter switching is a pointer operation of under a millisecond, so every role stays registered. Within a role the clone is exact, so a session pays only for its divergent suffix.

4 EXPERIMENTS

4.1 SETUP

Hardware and Reference Configurations. A Jetson AGX Orin 32 GB with NVMe and eMMC is the main platform. An Orin Nano Super 8 GB is the capacity-constrained board, and an AGX Orin 64 GB hosts the quality and crossover experiments; its eMMC writes ten times faster. A Snapdragon 8 Elite Gen 5 phone provides the UFS cross-point. Five configurations of the same engine differ only in mechanism and serve as reference configurations; an external system is compared in Section 4.5. RELOAD restarts the process per role switch. HOT-SWAP keeps every adapter registered and serves sessions in first-come-first-served (FCFS) order with KV save and load. RECOMPUTE replays each session's full history per turn. PREFIX-CACHE reuses prefixes in four slots and persists nothing, and ADAPTER-POOL serves stateless requests through an adapter-LRU pool. RECITE-T (w=4) is the configuration of every headline number and RECITE-I (w=2) the latency knob. Every restore-capable system runs the same adaptive restore policy and the same four resident slots, so the main comparison isolates batching and liveness, and Section 4.3 isolates the restore policy.

Models and Workloads. The main matrix serves Qwen3-8B (Qwen Team, 2025) in 4-bit weights with five rank-16 LoRA adapters, the 8 GB board serves Qwen3-1.7B with 8-bit KV, and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) in 4-bit weights is the second family (Section 4.5). Adapters are shape-only: load, bind, and switch costs depend on shape, not content, so answer quality is the base model's (Section 4.2). Workloads are `debate`, three agents over grade-school math (Du et al., 2024); `vote`, five-way self-consistency with a judge (Wang et al., 2023); `gui-3/5/7`, three to seven concurrent GUI workflows from real Android episodes (Li et al., 2024); and `linear`, an 11-turn pipeline with no mergeable structure that serves as the negative control.

Metrics and Protocol. We report makespan of the workflow set, energy per token from the board's power rails, and restore traffic by direction; the batching trade-off in Section 4.4 also reports the 95th-percentile (P95) turn latency, since merging widens the tail. We report averages over repeated runs; run-to-run spread stays under 1.5%, except on the two heaviest save-everything cells, where eMMC write drift raises it to 7.5%. Every prediction is computed independently of the cell it predicts and collected in Table 1.

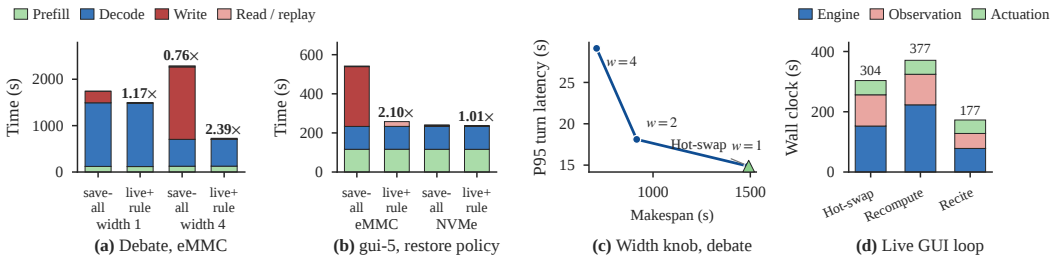


Figure 4: Mechanisms on the 32 GB board. (a) Time decomposition of restore policy × merge width on debate over eMMC. (b) The same for gui-5 on both tiers. (c) The width knob trades tail latency for makespan. (d) The live three-agent GUI loop, wall clock above each bar.

4.2 Main Results

Quantitative Comparison. Figure 3a,b run all six configurations under the unified restore policy and capacity. Parallel patterns gain the most. On five-way voting RECITE finishes 2.47× faster at 2.72× less energy per token than the strongest reference configuration, and on debate the factors are 2.10× and 2.31×. Two mechanisms produce the gain. Voter turns are independent, so same-role merging reads the weights once for four turns, and every voter session finishes after one turn, so liveness drops its state without a write. Debate serializes its rounds, so fewer turns are ready at once and the factor is smaller. The neutral zone holds: on the GUI traces and the linear control, where turns do not merge and the adaptive restore policy removes the storage penalty, HOT-SWAP ties RECITE within 0.6%. The stateless configurations lose 1.7–6.8× because they redo work that RECITE does once, reloading the weights at every role switch or prefilling the session's history again at every turn.

Live Workflows. A live experiment on the 64 GB board runs both workflows end to end on GSM8K test questions (Cobbe et al., 2021). Voting runs five voters and a judge; debate runs three debaters over two rounds. Each role is a separate session, and HOT-SWAP and RECITE see identical prompts, seeds, and a 1024/512-token generation budget (Figure 3c). RECITE finishes voting 2.35× faster at 2.52× less energy and debate 1.85× faster at 1.97× less energy. Energy per correct answer falls by the same factors, so the gain is not bought with shorter answers. Accuracy is equivalent within the pre-registered ±5 percentage-point (pp) margin on both workflows, and on debate the final answers are identical on every question. Voting lands inside its predicted speedup interval; debate lands just below it, at 1.85× against 1.9–2.3×. The doubled budget lengthens every decode turn, so the serialized second round takes a larger share of the makespan and same-role merging covers less of it. The truncated 384/256-token protocol is reported in Appendix E.1.

Accuracy. A ground-truth evaluation confirms that the mechanisms change when and where tokens are computed, not what comes out. On the 64 GB board we score Qwen3-8B on GSM8K and HumanEval+ (Liu et al., 2023) under four conditions that isolate one mechanism each: *solo*, *batched* in a merged group of four, *restored* by a save–drop–load mid-session, and *shared* through a prefix cloned from a same-role session. The three mechanism conditions stay within one point of solo accuracy on both benchmarks (Table 4). This is expected. Batching changes only which sequences share a decode step, a restore returns the saved KV unchanged, and a same-role clone recomputes the same keys and values.

4.3 Mechanism Analysis

Restore Policy in Isolation. Fixing every system to save-everything and re-running the GUI matrix isolates the restore policy (Figure 4b). On the five-workflow eMMC cell the rule routes every eviction to recompute and finishes 2.10–2.21× faster than save-everything, within 2.3% of what Proposition 1 predicted. The mechanism is the write side. Save-everything writes gigabytes of state to eMMC at 16 MB/s, over half of it for sessions that never return, whereas replaying the evicted tokens costs 24.4 s of prefill. On NVMe the rule saves every live session, and the two policies differ by only 1.2–1.3%, because the write is cheap there.

Mechanism Interaction. Figure 4a activates both mechanisms where both bind and decomposes each cell from the per-turn logs into prefill, decode, write, and read time (Table 5). At width one,

Table 1: Prediction ledger. Every prediction was computed from the trace and device constants alone; † marks rows registered before the runs.

Claim	Predicted	Measured	Verdict
Prop. 1	gui-5/eMMC restore-policy gain 2.15–2.21×; 1.9% and 5.2% cells at most 1.02–1.06×	2.10–2.21×; parity 1.000–1.006×	hit
Lemma 1	gui-5 misses twice at $K{=}4$, gui-3 never	2 and 0 observed	hit
Prop. 2	save on NVMe and UFS, recompute on eMMC	uniform per tier; phone 74×	hit
Prop. 3	decode ceiling 1.99, 2.96, 3.91 at $k{=}2, 3, 4$	1.94, 2.35, 2.52 (98/79/64%)	hit
Lemma 2†	cost-seeded vs. in-order seeding within 3% without pressure	vote $+0.1\%$; debate $+52.7\%$	miss on debate
Prop. 2†	64 GB eMMC: recompute at every $n \leq 32$K, since $r_p^\star = 557 < r_p$ tokens/s	21/21 agree; no crossover	hit
Prop. 2†	Llama-3.1-8B: $\kappa{=}128$ KiB, same decisions, gui-5/eMMC gain 2.31–2.61×	128.016 KiB; same; 2.46×	hit
Eq. (1)	held-out restore share from trace statistics and constants	$R^2{=}0.993$ on 62 cells; 14/14 decisions	hit
Quality†	solo GSM8K ≥ 0.80, HumanEval+ ≥ 0.55; other conditions within ± 5 pp	0.947 / 0.758; 6/6 within ± 5 pp	hit
Prop. 3†	live vote / debate, 1024/512 budget: 2.2–2.7×/ 1.9–2.3×, accuracy within ± 5 pp	2.35×/ 1.85×; both within ± 5 pp	hit / miss by 0.05
Prop. 3†	the same at the 384/256 budget	2.46×/ 1.90×; debate within ± 5 pp, vote misses by 1.0 pp	miss by 0.002 / miss
Prop. 2†	TTL-only ablation of Continuum's pinning (Li et al., 2025) within $[0.99, 1.06]$ of RECITE at any multiplier; learned liveness within $[0.99, 1.03]$	TTL 1.02–1.03× at $\geq 4\times$ reload, 1.10–1.22× at 1×, 0.91× on GUI eMMC; learned 4/4	partial / hit
Prop. 2†	rule vs. capability-matched reference: debate/eMMC 1.25–1.40×, gui-5/eMMC 1.35–1.60×	0.98×/ 1.65×	miss / miss by 0.05
External†	EdgeLoRA 0.8–1.5× of RECITE on vote/debate, 1.5–2.0× slower on gui-5	3.3× and 79–84×	miss

HOT-SWAP under save-everything spends 249 s writing 16.8 GB of state that no turn will read again, an effective 67 MB/s. The write is cheap because each eviction is 0.3 GB and the next is 23 s of decode away, so the page cache absorbs it and the kernel flushes it behind compute. Dropping those finished sessions removes the 249 s, a 1.17× gain that is all liveness; the tier rule has nothing to decide at this width, because no live session is evicted. Width four cuts decode time by more than half. Under save-everything, however, it evicts four sessions at once and shortens the gaps, so 19.7 GB of writes no longer fit the kernel's dirty-page budget and cost 1561 s at the device's 12.6 MB/s, on the critical path. Batching alone therefore *loses*, at 0.76×. With liveness and the rule, the finished sessions are dropped and the live evictions that batching creates are replayed in 22 s instead of written, and the cell runs 2.39× faster than the save-everything reference at width one. Batching does not merely add writes. It turns hidden write-back into synchronous writes, and the rule is what removes them; Proposition 4 bounds the synchronous part of the writes from below.

Capability-Matched Reference. A reference that batches, drops finished sessions, and saves every live eviction isolates the save-or-recompute decision itself. On gui-5 over eMMC the rule alone buys 1.65×. Each of the two live evictions there is 1.2 GB, a write of that size is synchronous, and replaying it is cheaper. On debate it buys nothing, at 0.98×. The live evictions that batching creates there are 53 MB each, and the page cache absorbs all of them, so the write costs almost nothing. The rule prices it at the device rate and over-estimates it, so replaying the tokens costs 13 s more than saving would have. The rule is conservative where the page cache absorbs the write.

4.4 PREDICTIONS AGAINST MEASUREMENTS

Ledger. Every neutral cell measured neutral and every cell under memory pressure delivered the predicted gain (Table 1); the rows that miss are explained where they occur.

Held-Out Calibration. To test the model instead of fitting it, we calibrate its two rates r_p and r_d on two traces, gui-3 and debate. Every other cell is then predicted from the trace's token counts, the measured κ, the tier rates, and a deterministic replay of residency under Lemma 1. The held-out cells span gui-5/7, voting, the 8 GB board, the phone, the Llama family, and the crossover sweep, and they give $R^2 = 0.993$ against measurement (Figure 2d). That R^2 reflects the spread across regimes; within each device group the mean absolute error in R is at most 0.022. Where both arms of a decision were measured, the rule picks the faster arm every time, with penalties from 1.09× to 146× had it chosen otherwise (Table 8).

Merge Width. Merging four ready turns lifts decode throughput 2.52× and cuts energy per token 2.94×, and the width ladder attains 98% of the ceiling at width two and 64% at width four. End to end on debate the ratios trail the ladder because prefill and lead turns do not batch. Width one ties HOT-SWAP, so the mechanism costs nothing when idle, and width two (RECITE-I) buys 1.63× of the makespan gain for 3.2 s of P95 turn latency, the trade that Figure 4c plots.

4.5 GENERALITY

Capacity-Constrained Tier. On the Orin Nano Super a 40K-token pool leaves room for barely one full-length session. RECITE still finishes a mergeable debate 1.83× faster than HOT-SWAP and ties it on the GUI workload, while the recompute class loses 5.3× (Figure 3d). The factor is below the 32 GB board's 2.10× because a single resident slot leaves no room to merge four turns.

Phone Storage. On a Snapdragon 8 Elite Gen 5 phone we run a storage microbenchmark instead of a workflow. Its UFS reads and writes at the same rate when cold, so $\alpha^\star$ sits far above the measured asymmetry at every prefill rate, and Proposition 2 predicts saving, the opposite of eMMC. A cold save-and-load round trip of an 8K-token session beats CPU replay by 74×.

Live GUI Loop. With three GUI agents driving three Android emulators end to end, engine time falls 1.9× and wall clock 1.7× (Figure 4d). The observation critical path halves because all three agents are served per loop, so part of the wall-clock gain is loop structure, not KV management.

Second Model Family. Llama-3.1-8B-Instruct in 4-bit weights replaces Qwen3-8B on the 32 GB board, with the GUI trace re-tokenized. Its architecture fixes κ at 128 KiB per token. With that constant the rule predicts the same decisions as for Qwen, save on NVMe and recompute on eMMC, and the runtime makes them. The restore-policy gain on gui-5 over eMMC, predicted at 2.31–2.61×, measures 2.46×, and the decode ladder and voting factor match Qwen's.

Learned Alternatives. Learned replacements for both decisions need device data the rules do not. A save-or-recompute classifier fed raw device features transfers poorly, because what transfers is the ratio the closed form already supplies, and a learned liveness model misses the GUI episode ends, which no positional feature announces and the DAG supplies for free (Appendix F).

External System. EdgeLoRA (Shen et al., 2025b) is a multi-tenant LoRA server built on llama.cpp and evaluated by its authors on the same Jetson boards. Its artifact does not support Qwen3, so the comparison uses Llama-3.1-8B-Instruct with the same adapters and traces and a same-day RECITE reference (Table 10). It is 3.3× slower on voting and debate and 79–84× slower on gui-5. On voting neither system moves or replays any state, so that cell measures decode paths alone. EdgeLoRA sustains 15.8 tokens/s aggregate against our 52.7 on identical work, while a single uninterrupted stream of our own engine runs at 23.2. The 3.3× is therefore a decode-path gap. Most of it comes from the 2024 llama.cpp its artifact pins and from a merge policy that batches by arrival and not by role. We claim none of it. What the comparison does establish is independent of the kernel. EdgeLoRA's slots do not preserve a session's KV across an eviction, so on debate it repeats 81,041 tokens of prefill and on the five-workflow GUI trace 324,807. RECITE repeats none on NVMe and 16,455 on eMMC by the rule's own choice. EdgeLoRA's makespan differs by 1.7% between the two tiers, inside its own 2.2% run-to-run spread, because a system that never writes KV has no placement decision to make. RECITE differs by 8.6% against a 0.1% spread, as Proposition 2 prescribes. A fifth EdgeLoRA slot fits once its micro-batch is quartered. It removes every GUI eviction and 6.2× of the gap; the remaining 13.6× is the engine gap that voting already exposed.

5 CONCLUSION

We introduced RECITE, a runtime that prices state restoration for multi-agent LLM workflows on one memory-constrained device from two facts that request-stream cost models omit: flash is far slower to write than to read, and the workflow DAG tells which sessions are finished. Three closed-form rules follow, which evicted sessions to save, which to drop, and how many ready turns to merge. The rules need only traces and device constants, and across three Jetson boards, a phone, two model families, and a live Android loop they predicted where each mechanism pays, with the gains holding at equivalent accuracy. Future work will replace the device-rate write cost with a burst-aware one, extend the comparison to data-center engines, and scale up the live deployment.

THE USE OF LARGE LANGUAGE MODELS

Generative AI tools (LLM-based coding and writing assistants operated by the authors) assisted with the tasks the ICLR policy requires us to disclose: organizing the measurements into the cost model and the decision rules, drafting the propositions and their proofs from the authors' derivations, designing the pre-registration and calibration protocol, implementing the control plane, the analysis and plotting scripts, and the assertion suite, and auditing reported numbers against the raw logs. Among the recommended disclosures, they helped draft and edit prose, revise figures, survey related literature, and format references. They generated no data and cleaned or reformatted none; every number in the paper is computed by scripts from measured logs, and the frozen-number assertion suite in the artifact recomputes each headline value. The authors checked every claim, proof, line of code, and reference, and take full responsibility for the content.

REPRODUCIBILITY STATEMENT

The artifact contains the data-plane engine, the control plane implementing every mechanism and reference configuration, the recorded traces, the raw per-run logs and energy samples behind every figure and table, the regeneration scripts, the frozen-number assertion suite, and the prediction ledger with its commit history. Appendix A gives every proof, Appendix B lists the hardware, software versions, and engine commit, and Appendix H defines every workload and reference configuration.

ETHICS STATEMENT

The work targets on-device execution so that a user's context need not leave the device, and reduces energy per token. No human subjects were involved; GUI traces derive from a public Android interaction dataset used under its license, and the live loop drove emulators, not user devices.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. INFERCEPT: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 81–95, 2024.

Ali Allahverdi, C. T. Ng, T. C. E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008. doi: 10.1016/j.ejor.2006.06.060.

L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi: 10.1147/sj.52.0078.

Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. Tokencake: A KV-cache-centric serving framework for LLM-based multi-agent applications, 2025. arXiv:2510.18586; to appear in EuroSys 2027.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. IMPRESS: An Importance-Informed Multi-Tier prefix KV storage system for large language model inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 187–201, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8.

Jinha Chung, Byeongjun Shin, Jiin Kim, and Minsoo Rhu. Agent-X: Full pipeline acceleration of on-device AI agents. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 144–157, 2026. doi: 10.1145/3745756.3809195. arXiv:2605.10380.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

J. G. Dai, Tianze Deng, Yueying Li, and Tianyi Peng. Throughput-optimal scheduling algorithms for LLM inference and AI agents. arXiv:2504.07347, 2025. URL https://arxiv.org/abs/2504.07347.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 11733–11763, 2024.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (ATC)*, pp. 111–126, 2024. Earlier preprint titled "AttentionStore", arXiv:2403.19708.

Georgi Gerganov and llama.cpp contributors. llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Kristjan Greenewald, Luis Lastras, Thomas Parnell, Vraj Shah, Lucian Popa, Giulio Zizzo, Chulaka Gunasekara, Ambrish Rawat, and David Cox. Activated LoRA: Fine-tuned LLMs for intrinsics. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 53022–53049. Curran Associates, Inc., 2025. doi: 10.52202/085713-1769. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4d0b6303d4a4811445f69f357bf6def5-Paper-Conference.pdf.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. SAGA: Workflow-atomic scheduling for AI agent inference on GPU clusters, 2026. arXiv:2605.00528.

Liwei Guo, Wonkyo Choe, and Felix Xiaozhu Lin. STI: Turbocharge NLP inference at the edge via elastic pipelining. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 791–803, 2023. doi: 10.1145/3575693.3575698.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity. arXiv:2505.11107, 2025. URL https://arxiv.org/abs/2505.11107.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. In *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=ooRukY3zjQ. arXiv:2602.01053.

Jinwoo Jeong and Jeongseob Ahn. Accelerating LLM serving for multi-turn dialogues with efficient resource management. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*, pp. 1–15, 2025. doi: 10.1145/3676641.3716245.

Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Zhuoqing Mao. Compute or load KV cache? Why not both? In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 28031–28043, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. MELTing point: Mobile evaluation of language transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pp. 890–907, 2024. doi: 10.1145/3636534.3690668.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pp. 51991–52008, 2023.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live, 2025. arXiv:2511.02230.

Wei Li, William E. Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. AndroidControl dataset; arXiv:2406.03679.

Fulin Lin, Shaowen Chen, Ruishan Fang, Hongwei Wang, and Tao Lin. Stop wasting your tokens: Towards efficient runtime multi-agent systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations (ICLR)*, pp. 113497–113528, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b86a195e70f27017c514fa0e5f80595f-Paper-Conference.pdf. arXiv:2510.26585.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the 7th Annual Conference on Machine Learning and Systems (MLSys)*, pp. 87–100, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. arXiv:2305.01210.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024a. doi: 10.1145/3651890.3672274.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. arXiv:2510.09665, 2025.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. DroidSpeak: KV cache sharing across fine-tuned model variants. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 319–338, 2026. Preprint: arXiv:2411.02820, under an earlier title.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 32431–32454, 2024b.

William Meng, Benjamin Lee, and Hong Wang. Understanding bottlenecks for efficiently serving LLM inference with KV offloading. arXiv:2601.19910, 2026. URL https://arxiv.org/abs/2601.19910.

Luca Moschella, Laura Manduchi, and Ozan Sener. Learning to evict from key-value cache. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2602.10238.

Hongqiu Ni, Han Tian, Chi Zhang, Guopeng Li, and Haisheng Tan. TOPAS: Workflow-aware prefix-state scheduling for multi-agent LLM serving, 2026. arXiv:2608.25523.

Sean Nian, Jiahao Fang, Qilong Feng, Zhiyu Wu, and Fan Lai. Cacheflow: Efficient LLM serving with 3D-parallel KV cache restoration, 2026. arXiv:2604.25080.

Zaifeng Pan, Ajjkumar Dahyalal Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 6th edition, 2022. doi: 10.1007/978-3-031-05921-6.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810.

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

Shi Qiu, Yifan Hu, Xintao Wang, Wenhao Zhu, Jianqin Yan, Hao Chen, Kaiqiang Xu, Kai Chen, and Yiming Zhang. Tutti: Making SSD-backed KV cache practical for long-context LLM serving, 2026. arXiv:2605.03375.

Qwen Team. Qwen3 technical report, 2025.

Zebin Ren, Krijn Doekemeijer, Tiziano De Matteis, Christian Pinto, Radu Stoica, and Animesh Trivedi. An I/O characterizing study of offloading LLM models and KV caches to NVMe SSD. In *Proceedings of the 5th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*, pp. 23–33. ACM, March 2025. doi: 10.1145/3719330.3721230. URL https://doi.org/10.1145/3719330.3721230.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel LLM generation via concurrent attention. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 46592–46633. Curran Associates, Inc., 2025. doi: 10.52202/085713-1551. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4293727fc48c60b4adfe7a8e44f6a446-Paper-Conference.pdf.

Junyi Shen, Noppanat Wadlom, and Yao Lu. Batch query processing and optimization for agentic workflows. arXiv:2509.02121, 2025a. URL https://arxiv.org/abs/2509.02121.

Zheyu Shen, Yexiao He, Ziyao Wang, Yuning Zhang, Guoheng Sun, Wanghao Ye, and Ang Li. EdgeLoRA: An efficient multi-tenant LLM serving system on edge devices. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 138–153, 2025b. doi: 10.1145/3711875.3729141.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Yakov Pyotr Shkolnikov. Agent memory below the prompt: Persistent Q4 KV cache for multi-agent LLM inference on edge devices, 2026. arXiv:2603.04428.

Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Jiashun Suo, Xiaojian Liao, Limin Xiao, Li Ruan, Jinquan Wang, Xiao Su, and Zhisheng Huo. CoServe: Efficient collaboration-of-experts (CoE) model inference with limited memory. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*, pp. 178–191, 2025. doi: 10.1145/3676641.3715986.

Haodong Wang, Qihua Zhou, Zicong Hong, and Song Guo. D^2MoE: Dual routing and dynamic scheduling for efficient on-device MoE-based LLM serving. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking (MobiCom)*, pp. 574–588, 2025. doi: 10.1145/3680207.3723493.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. In *ICLR Workshop on Large Language Model (LLM) Agents*, 2024.

Shao Wang, Rui Ren, and Lin Gui. ForkKV: Scaling multi-LoRA agent serving via copy-on-write disaggregated KV cache. arXiv:2604.06370, 2026. URL https://arxiv.org/abs/2604.06370.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

Sunghyeon Woo, Jaeeun Kil, Hoseung Kim, Minsub Kim, Joonghoon Kim, Ahreum Seo, Sungjae Lee, Minjung Jo, Jiwon Ryu, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. ICaRus: Identical cache reuse for efficient multi-model inference. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations (ICLR)*, pp. 32937–32957, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/37f6be32f832caf0f7980469fb06165b-Paper-Conference.pdf. arXiv:2603.13281.

Sunghyeon Woo, Hoseung Kim, Sunghwan Shim, Minjung Jo, Hyunjoon Jeong, Jeongtae Lee, Joonghoon Kim, Sungjae Lee, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. PrefillShare: A shared prefill module for KV reuse in multi-LLM disaggregated serving. arXiv:2602.12029, 2026b. URL https://arxiv.org/abs/2602.12029.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling (COLM)*, 2024.

Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. LLMCad: Fast and scalable on-device large language model inference, 2023.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. PowerInfer-2: Fast large language model inference on a smartphone, 2024.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2510.12872.

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. EdgeMoE: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24(8):7059–7073, 2025. doi: 10.1109/TMC.2025.3546466.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. AppAgent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI)*, 2025a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations (ICLR)*, pp. 75389–75428, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/bbc461518c59a2a8d64e70e2c38c4a0e-Paper-Conference.pdf.

Huawei Zhang, Chunwei Xia, and Zheng Wang. KVSwap: Disk-aware KV cache offloading for long-context on-device inference. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 723–736, 2026a. doi: 10.1145/3745756.3809234. arXiv:2511.11907.

Rui Zhang, Chaeeun Kim, Shaoting Feng, Kuntai Du, Yuhan Liu, Yi Zhong, Cheng-Wei Ching, Junchen Jiang, and Liting Hu. Learning agent execution for KV-cache management in agentic serving, 2026b. arXiv:2608.14624.

Haoyu Zheng, Fangcheng Fu, Jia Wu, Binhang Yuan, Yongqiang Zhang, Hao Wang, Yuanyuan Zhu, Xiao Yan, and Jiawei Jiang. Efficient serving for dynamic agent workflows with prediction-based KV-cache management, 2026. arXiv:2605.06472.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

A PROOFS

Proof of Proposition 1. Write T for the reference run's makespan. It decomposes as RT of restoration and $(1-R)T$ of everything else. The mechanism leaves the latter untouched, keeps $(1-\eta)RT$ of the former, and adds ϵT, so its makespan is $(1-\eta R+\epsilon)\,T$ and S is the reciprocal of that factor. $S>1$ iff $\eta R>\epsilon$, and S is maximized at $\eta=1$, $\epsilon=0$, where it equals $1/(1-R)$. □

Proof of Lemma 1. Under LRU the resident set after any turn is the K most recently run distinct sessions (fewer while the cache fills). Session s ran at its previous turn; it is still among the K most recent iff at most $K-1$ distinct other sessions ran since, that is, iff $D_t<K$. So a turn of s misses iff $D_t\geq K$, and $\phi(K)=\Pr[D_t\geq K]$; differencing at K and $K+1$ gives $\Pr[D_t=K]$. Unequal sizes break the argument only through the identity of the K most recent sessions, which byte-budget eviction can alter; evicting finished sessions first removes victims that would never be revisited, so it can only lower ϕ further. □

Proof of Proposition 2. A session of n tokens holds $b=\kappa n$ bytes. Saving writes those bytes and reads them back, so $C_{\text{save}}=\kappa n\,(1/r_w+1/r_r)=(\kappa n/r_r)(\alpha+1)$; recomputing prefills the same tokens, so $C_{\text{recomp}}=n/r_p$. Saving is preferred exactly when $(\kappa n/r_r)(\alpha+1)\leq n/r_p$; both sides are linear in n, which cancels. With return probability q the read and the replay are each incurred with probability q, giving $b/r_w+q\,b/r_r\leq q\,n/r_p$, i.e. $\alpha+q\leq q\,r_r/(\kappa r_p)$. Solving Equation (3) for the prefill rate gives the equivalent form $r_p\leq r_p^{\star}=r_r r_w/(\kappa(r_r+r_w))$. □

The cancellation assumes whole-session save and replay. Prefix reuse shrinks both sides to the divergent suffix and keeps the ratio, whereas incremental save against a stale on-disk prefix would

reintroduce the session length. On our traces no live session is evicted twice at four slots, so incremental save would not change any 32 GB cell.

Proof of Proposition 3. Executing k turns one at a time reads the weights M and one sequence's KV $L\kappa_{\text{mem}}$ per step, $k(M + L\kappa_{\text{mem}})/B$ at achieved bandwidth B; merging performs one step that reads the weights once and all k sequences' KV, $(M + kL\kappa_{\text{mem}})/B$. The ratio is Equation (4), B cancels, and dividing through by M shows $S(k) \to k$ as $L\kappa_{\text{mem}}/M \to 0$; for unequal contexts the same argument gives $(kM + \kappa_{\text{mem}} \sum_i L_i)/(M + \kappa_{\text{mem}} \sum_i L_i)$. For the width rule, T_b is affine and increasing in k, and $k/T_b(k)$ has derivative $(T_0 + M/B)/T_b(k)^2 > 0$; the feasible set $\{k : T_b(k) \le D\}$ is a prefix of $\{1, \dots, w_{\text{ready}}\}$, and throughput is therefore maximized at its largest element, the widest feasible width. □

Lemma 2 (No re-entry at $K = 1$)**.** *With one resident slot, a DAG without cross-session ordering, and no exogenous release times, a session-affine greedy (keep the current session while it has a ready turn) activates every session exactly once; its restore cost is zero, and no schedule does better.*

Proof. Greedy leaves a session only when it has no ready turn. Without cross-session ordering, a session's remaining turns depend only on its own earlier turns, so a session with no ready turn has no turn left and never becomes ready again; greedy therefore enters each session once. A first entry restores nothing, because the session has no prior state, and restore costs are nonnegative, so greedy's restore cost is zero and is a lower bound for every schedule. □

Proposition 4 (Write floor)**.** *In a runtime whose KV writes and reads are not overlapped with compute, the makespan of any schedule satisfies $T \ge B_w/r_w + B_r/r_r$, where B_w and B_r are the bytes it writes and reads back synchronously. Under Proposition 2 on a tier where recomputing wins, $B_w = 0$ and the floor vanishes.*

Proof. Each written byte occupies the write path for $1/r_w$ seconds and each byte read back the read path for $1/r_r$; when neither overlaps compute, their sum bounds wall clock from below. Under the rule no live eviction writes on a recompute tier, and finished sessions never write. Writes that the page cache absorbs are not synchronous and fall outside the bound; on the debate–eMMC cells of Figure 4a width four writes 19.7 GB against 16.8 GB at width one, an empirical increase that batching causes there but that no general monotonicity claim covers. □

The measured write time sits on that floor at width four (1561 s for 19.7 GB) and far below it at width one, where write-back overlaps decode. Whether a write is synchronous depends on burst size against the page cache, not on bytes alone.

Corollary 1 (Robust separation)**.** *On the 32 GB configuration κ is a measured 144 KiB per token and the trailed prefill rate runs at 0.7–0.95 K tokens/s, so κr_p is 0.10–0.14 GB/s and $\alpha^\star$ lies between 14 and 22 on both tiers, against measured path asymmetries of 2.8 (NVMe) and 132 (eMMC). Under direct-I/O device rates the eMMC threshold is 1.0–1.8 against a device asymmetry of 22, and under cold-cache reads it stays below 1 against 5–10; no coding and no prefill rate in the measured range moves either tier across its threshold.*

B Implementation and Platform

`rt_engine` is a C++ data plane over a pinned llama.cpp (Gerganov & llama.cpp contributors, 2023), driven by a Python control plane; both run unmodified on Jetson AGX Orin 32 GB (JetPack 7.2, CUDA 13.2), AGX Orin 64 GB and Orin Nano Super 8 GB (JetPack 6.2, CUDA 12). The control plane holds the trace schema, turn-order policy, restore cost model, and capacity mirror; the data plane runs single and batched turns, saves, loads, drops, and prefix-copies KV, and binds adapters. Timing is instrumented in the data plane so that control-plane jitter cannot pollute measurements; the control plane's per-turn cost is 0.3 ms, and an adapter switch takes 0.3–0.9 ms. Deciding between saving and recomputing at eviction, not at admission, lets the runtime use the size a session actually reached and the prefill rate it actually observed. Within a role a session pays only for its divergent suffix, found by a radix index over 256-token block hashes, and the clone is exact because the same adapter over the same prefix recomputes exactly the same keys and values at every position.

Write traffic. Over one run of a three-workflow trace on the 32 GB board, persisting evicted KV writes 105 GB in 6461.6 s and reads 101 GB back in 47.7 s.

Algorithm 1 Per-turn loop of RECITE ($B_{\mathrm{KV}} = K\kappa L_{\max}$ for equal slots).

while ready turns $\mathcal{Q} \neq \emptyset$ **do**
  seed $t^\star \leftarrow$ earliest turn in $\mathcal{Q}$ ▷ cost-seeded variant: $\arg\min_{t\in\mathcal{Q}} \textsc{Cost}(t)$
  group $G \leftarrow \{t^\star\} \cup$ first $\min(w, K)-1$ ready turns of $\mathrm{role}(t^\star)$
  for all $s \in G$ not resident, **while** $\mathrm{Bytes}(\mathcal{R} \cup \{s\}) > B_{\mathrm{KV}}$ **do**
    $v \leftarrow$ a resident session outside G with no future turn, else the least recently used
    if v has no future turn **then** DROP(v) ▷ q=0
    else if $b_v/r_w + b_v/r_r \leq n_v/r_p$ **then** SAVE(v) **else** DROP(v) ▷ Equation (3)
  end for
  clone the longest resident same-role prefix into each admitted s; read its saved remainder or prefill it
  BINDADAPTER($\mathrm{role}(G)$); RUNTURNS(G)
end while

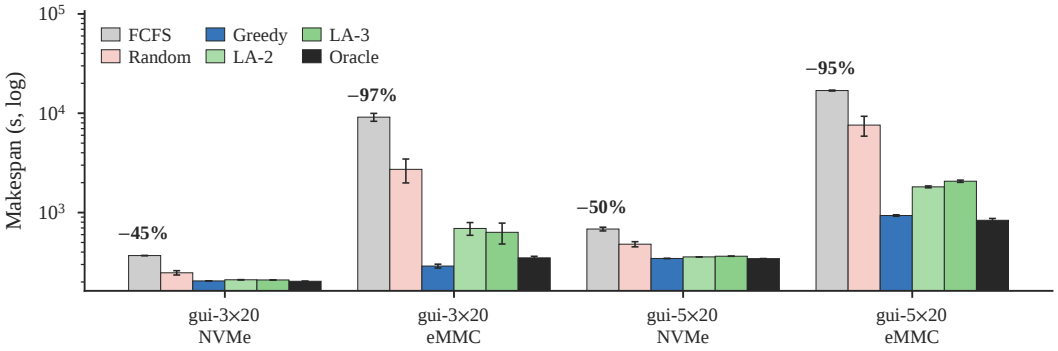


Figure 5: Turn order under memory pressure on synthetic every-turn-switch traces, log scale; LA-k is a k-step lookahead policy and Oracle the rolling twelve-turn optimum.

Capacity. On the 32 GB board four 32K-token slots take 18 GiB at 144 KiB per token beside 4.68 GiB of weights, 0.74 GiB of adapters, and 0.34 GiB of compute buffers; measured at four resident slots the board reports 25.4 of 30.0 GiB physical used, so a fifth 4.5 GiB slot plus engine headroom does not fit. K is a provisioning choice: slots are reserved at $L_{\max}$, and the 8 GB board instead uses a token pool that trades the reservation for overflow risk. On the 8 GB board the 1.7 B model takes 1.1 GB, compute buffers 0.8 GB, and the unified 40K-token pool a measured 59.5 KiB per token.

Platform notes. Unified memory double-counts memory-mapped weights, so long runs disable mapping; a 17 GB model otherwise peaks at 33 GB resident. Prefix copy requires a unified KV pool sized to the longest session times the sequence count. Capacity constants are measured at load instead of carried between runs, because a stale bytes-per-token constant silently disables byte-budget eviction; every run also logs its replayed tokens and tags its harness version. Power checks, window filtering, and text-mirror replay guard the measurements against a wrong power mode, a runaway sampling window, and engine aborts. All headline results use one harness version; the liveness-ablation rows in Table 5 come from the otherwise identical earlier version.

C TURN ORDERING AND VICTIM SELECTION

$\textsc{Cost}(t)$, used by the cost-seeded variant of Algorithm 1, is the time to make one turn runnable: nothing if its session is resident, b_s/r_r if its state was saved, n_s/r_p if dropped, plus b_v/r_w for each live victim that would be saved to admit it. To stress ordering at its ceiling we run synthetic traces that change session every turn, with two resident slots, save-everything, six policies, and two tiers (Figure 5). The best policy beats FCFS by 44.9–96.8%. A rolling twelve-turn optimum ties both NVMe cells and trails on one eMMC cell, because it bounds write avoidance, not makespan. Under Poisson arrivals without memory pressure FCFS and greedy have identical goodput. On victim ranking, a furthest-next-use victim turns the GUI cell's replay from 16.6K tokens to 553 (24.4 to 0.7 s) and its eMMC makespan from 259.0 to 235.5 s, but degrades debate from 709.9 to 1224.4 s on NVMe while leaving groups, prefill, generated tokens, restore traffic, and arena occupancy identical step for step. The tax is the engine's batch fast path: LRU's slot recycling keeps each merged batch's sequence identifiers ascending and llama.cpp's CUDA-graph cache hot (70 ms per decode step),

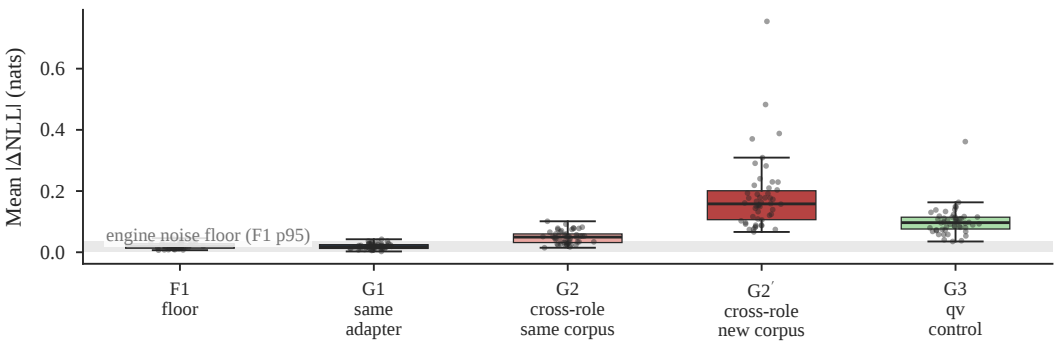


Figure 6: Per-prompt divergence between shared-prefix and recompute paths against the engine's noise floor: same-adapter sharing sits inside it, and cross-role drift grows with corpus divergence.

whereas next-use recycling emits permuted batches that miss that cache on every step (193 ms); an isolated replay reproduces it ($[0, 1, 2, 3]$ at 69.9 ms per step, $[3, 2, 1, 0]$ at 205.8 ms). Cost-aware group admission trips the same tax from the other side (52% on NVMe) while narrowing width from 3.9 to 1.4 where it should on eMMC. RECITE therefore keeps liveness-first LRU and in-order expansion, and counts canonical batch order as part of the effective scheduling cost on this backend. A paired comparison on the real traces confirms the direction and the size of the effect. The cost-seeded variant of line 2 of Algorithm 1, with everything else identical, leaves voting unchanged over NVMe (843.0 against 841.9 s) and slows debate from 709.3 to 1082.8 s (P95 29.1 to 43.4 s, 244 to 383 mJ per token) while moving fewer restore bytes, 2.1 against 2.7 GB, which is the signature of the batch-order tax and not of I/O. The prediction of a difference within 3% therefore holds on vote and fails on debate, and the headline configuration seeds in order.

D CROSS-ROLE PREFIX REUSE (EXPLORATORY)

Within a role the prefix clone is exact; across roles it is an approximation, because the hidden states a role adapter induces differ from the second layer on even when the key and value projections are frozen. The condition for exactness is simple:

Proposition 5 (Prefix-neutral adapters)**.** *Let two roles share a base model and a shared prefix of identical token identifiers, and let each role's adapter contribute zero to every projection at every shared-prefix position in every layer. Then the two roles' hidden states, and hence their keys and values, coincide at every shared-prefix position in every layer, and a cross-role prefix clone is exact.*

Proof. By induction on the layer: the embeddings of identical token identifiers are identical; if the inputs to layer ℓ agree at all shared-prefix positions, then, because attention at a prefix position reads only prefix positions and the adapters contribute nothing there, the attention and feed-forward maps of layer ℓ are the base model's on both roles and their outputs agree, as do the keys and values. □

ICaRus reaches exactness by construction: it freezes the logical encoder that produces keys and values and fine-tunes only the decoder side, so all task-specialized models share one cache, at the cost of a second base-model stream during decoding (Woo et al., 2026a); LRAgent shares the key cache across LoRA agents because measured similarity is high and decomposes the value cache into a shared base and a low-rank adapter part (Jeon et al., 2026); Activated LoRA, PrefillShare, and ForkKV restrict adaptation in time or split prefill from decode to reuse base KV (Greenewald et al., 2025; Woo et al., 2026b; Wang et al., 2026). This appendix reports what the approximate form costs without a duplicate base stream. The constrained placement adapts only the query, output, and feed-forward projections, which satisfies the hypothesis in the key and value projections alone, so the clone is approximate. Three rank-16 adapters were trained on Qwen3-4B, a coder and a critic on CodeAlpaca and a planner on AgentInstruct; held-out loss falls from 1.48 to 0.91 for the coder and from 2.03 to 0.39 for the planner. The coder exists in two placements, constrained and one that also adapts value.

Over 50 prompts, same-adapter sharing is indistinguishable from the floor (0.0193 vs. 0.0208 nats). Between the coder and the critic, trained on one corpus, the constrained placement reads 0.0487

Table 2: Every configuration on the 32 GB board under one restore policy and capacity: means of three runs, lower is better, bold marks the best per column within 0.6%, and the last row is RECITE-T's factor over the strongest reference configuration.

	Makespan (s) ↓					Energy (mJ/token) ↓				
System	vote NVMe	debate NVMe	gui-5 NVMe	gui-5 eMMC	linear NVMe	vote NVMe	debate NVMe	gui-5 NVMe	gui-5 eMMC	linear NVMe
RELOAD	3190	2434	1522	1522	177	651	712	7016	7016	918
RECOMPUTE	2090	1533	1406	1406	80	529	562	6673	6673	588
ADAPTER-POOL	2113	1535	1406	1406	81	529	566	6716	6716	593
PREFIX-CACHE	2184	1606	445	445	**74**	562	601	2247	2247	**559**
HOT-SWAP	2084	1492	239	**259**	**74**	526	555	**1266**	**1366**	**559**
RECITE-T	**844**	**710**	**238**	**259**	**73**	**193**	**240**	**1261**	**1366**	**557**
RECITE-T gain over strongest reference	2.47×	2.10×	1.01×	1.00×	1.00×	2.72×	2.31×	1.00×	1.00×	1.00×

Table 3: Live voting and debate on the 64 GB board at the 1024/512-token budget: the first 100 GSM8K test questions, three seeds pooled. The accuracy difference is RECITE minus HOT-SWAP with its 90% confidence interval from a question-level bootstrap; TOST is the two-one-sided equivalence test at the ±5 pp margin.

	Voting (5 voters + judge)		Debate (3 debaters × 2 rounds)	
	HOT-SWAP	RECITE	HOT-SWAP	RECITE
Accuracy	0.88	0.88	0.97	0.97
Difference (pp), 90% confidence interval, TOST ±5	−0.3, [−4.0, +3.3], pass		0.0, identical answers, pass	
Makespan (s)	27359	11646 (2.35×)	30657	16591 (1.85×)
Energy (kJ)	297.3	117.9 (2.52×)	335.0	170.1 (1.97×)
Energy per correct answer (J)	3378	1345 (2.51×)	3454	1754 (1.97×)

nats against 0.1006 for the unconstrained placement; between the coder and the planner, trained on different corpora, it reads 0.1816, above the unconstrained same-corpus control (Figure 6). Corpus divergence and placement are confounded in this design, so the measurement shows that placement halves the projection term's drift and leaves the roles' own divergence, not that constrained placement fails; a production version would need a per-role-pair gate with a calibration set, which we have not built. Replicating across two JetPack/CUDA generations moves group means at most 8% and leaves the ordering of the four groups identical.

E ADDITIONAL RESULTS

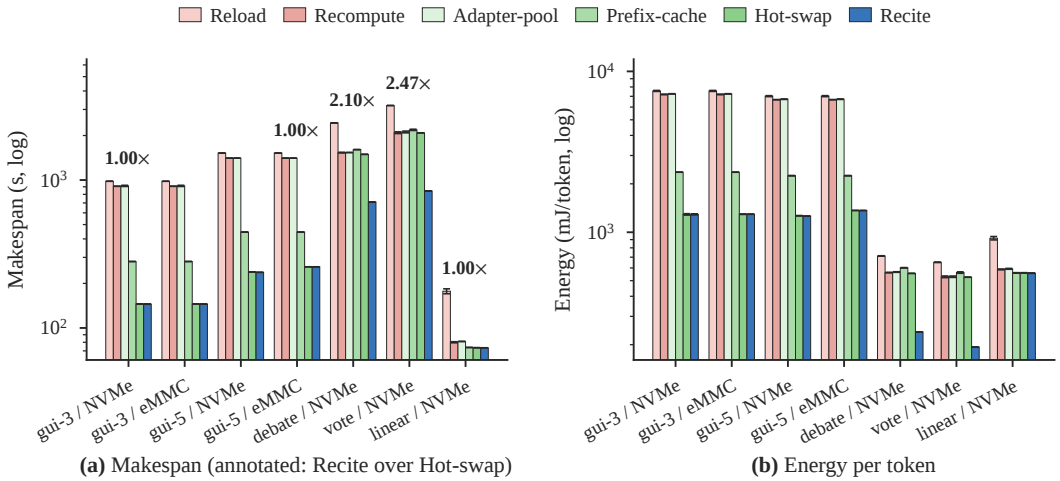
E.1 MAIN RESULTS

Main matrix and live workflows. Table 2 lists every cell behind Figure 3a,b, and Figure 7 plots them with gui-3, which is at parity on both tiers; every 32 GB and 8 GB cell runs three times. Table 3 gives the live workflows behind Figure 3c. In the live runs accuracy is 0.88 against 0.88 on voting and 0.97 against 0.97 on debate. At the trace protocol's 384/256-token budget RECITE finishes voting 2.46× faster at 2.67× less energy and debate 1.90× faster at 2.01× less energy, at 0.35–0.37 absolute accuracy; debate passes the equivalence test, and voting misses it by one point of interval width, its interval reaching +6.0 pp.

Quality detail. Table 4 lists every condition of the ground-truth evaluation of Section 4.2. Solo accuracy is 0.947 on GSM8K and 0.758 on HumanEval+; batched, restored, and shared read 0.943, 0.948, 0.953 and 0.758, 0.750, 0.754. Equivalence is tested with two one-sided tests (TOST) at the ±5 pp margin on 90% confidence intervals from a question-level bootstrap, and all six condition–family pairs pass; every interval lies within −1.6 to +1.7 pp, and no paired permutation test rejects equality ($p \geq 0.22$). The 95% confidence interval of solo accuracy is 0.930–0.960 on GSM8K and 0.718–0.794 on HumanEval+. Greedy decoding scores 0.953 and 0.762.

E.2 MECHANISM ANALYSIS

Restore isolation. Figure 8 plots the comparison of save-everything with the adaptive restore policy behind Section 4.3. On the five-workflow eMMC cell the adaptive path replays 16.6K tokens in 24.4 s and finishes in 259.1 s (HOT-SWAP) and 259.0 s (RECITE), against 573.0 and 543.1 s for



(a) Makespan (annotated: Recite over Hot-swap) **(b)** Energy per token

Figure 7: (a) Makespan and (b) energy per token under one restore policy and capacity: log scale, three runs, error bars are sample standard deviations.

Table 4: Ground-truth accuracy under the four decoding conditions: 64 GB board, Qwen3-8B, temperature 0.7, three seeds pooled, 300 GSM8K and 164 HumanEval+ questions. Diff. is the paired difference to solo in pp with its 90% confidence interval from a question-level bootstrap, TOST is the two-one-sided equivalence test at the ± 5 pp margin, and p comes from a paired permutation test.

Family	Condition	n	Acc.	95% int.	Diff. (pp)	90% int. (pp)	TOST / p
GSM8K	solo	900	0.947	[0.930, 0.960]	–	–	–
	batched	900	0.943	[0.926, 0.957]	-0.33	$[-1.22, +0.56]$	pass / 0.68
	restored	900	0.948	[0.931, 0.961]	$+0.11$	$[-0.56, +0.78]$	pass / 1.00
	shared	900	0.953	[0.938, 0.965]	$+0.67$	$[-0.33, +1.67]$	pass / 0.35
HumanEval+	solo	492	0.758	[0.718, 0.794]	–	–	–
	batched	492	0.758	[0.718, 0.794]	0.00	$[-0.61, +0.61]$	pass / 1.00
	restored	492	0.750	[0.710, 0.786]	-0.81	$[-1.63, 0.00]$	pass / 0.22
	shared	492	0.754	[0.714, 0.790]	-0.41	$[-1.22, +0.41]$	pass / 0.69

save-everything, a 2.21× and 2.10× gain. Of the 284 s removed, dropping 2.7 GB of finished-session writes accounts for 161 s, and replaying the two live evictions instead of writing and reading back 2.4 GB accounts for the rest; dropping the caches before every read-back moves the save-everything cell from 573 to 587 s and the NVMe cells by 1.1%.

KVSwap anchor. KVSwap's public code on the 64 GB board bears the write-side bind out: its 16K-token prefill spends 47.5 s writing 2.5 GB at 53.7 MB/s to eMMC over five runs with a standard deviation of 3.1 s, against 29.7 s with 0.2 on NVMe.

Interaction detail. Table 5 gives the restore-policy × merge-width cells of Section 4.3: dropping finished sessions at width one gives 1489.9 s, batching alone under save-everything gives 2289.1 s, and batching with liveness and the rule gives 727.8 s, with the 56 finished sessions dropped and the 52 live evictions replaying 18.5K tokens. Decode time is 1371 s at width one and 579 s at width four. An all-live debate of 16 sessions over eMMC composes identically (817.0 against 4006.3 s, 4.90×, nothing written), width one reproduces HOT-SWAP (1488.4 against 1491.6 s), and removing liveness costs 1.8–2.9% on workflows with terminal sessions.

Capability-matched reference. The reference of Section 4.3 runs the same engine, width-four batching, and liveness-first eviction, and saves every live eviction instead of applying the tier rule; both arms ran interleaved on the 32 GB board. On gui-5 over eMMC the rule alone buys 1.65× (426.5 against 258.8 s), because the two live evictions write 2.4 GB in 146 s; on debate it buys nothing (0.98×, 714.9 against 727.7 s), because the page cache absorbs all 2.7 GB in 7 s and replaying 18.5K tokens costs 13 s more than saving would have. Debate: RECITE 727.7 s (± 1.4) replaying 18,546 tokens in 22 s against 714.9 s (± 2.6) writing 2.74 GB in 4–8 s and reading it back in 4–7 s

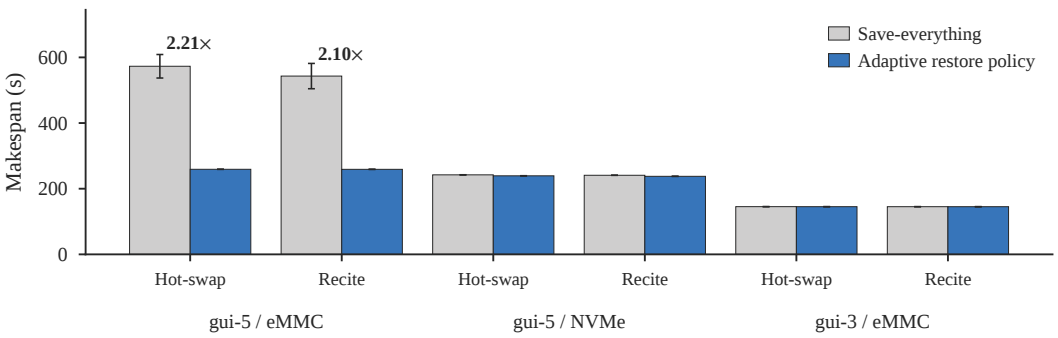


Figure 8: Save-everything against the adaptive restore policy, on the cells where eviction meets slow flash and on the others.

Table 5: Restore policy × merge width on debate over eMMC, means of three; the right block reads P95 turn latency / energy per token / GB written.

	Makespan (s), factor over HOT-SWAP		P95 (s) / mJ per token / GB written	
Restore policy	width 1	width 4	width 1	width 4
save-everything	1742.5 (1.00×)	2289.1 (0.76×)	23.6 / 611 / 16.8	59.4 / 468 / 19.7
tier rule (Proposition 2)	1489.9 (1.17×)	727.8 (2.39×)	14.9 / 551 / 0	29.5 / 245 / 0

(52 evictions of 53 MB); paired ratios 0.977–0.987. gui-5: RECITE 258.8 s (±0.0) replaying 16,569 tokens in 24 s against 426.5 s (±14.1) writing 2.44 GB in 127–162 s and reading it back in 41–53 s (two evictions of 1.2 GB); paired ratios 1.60–1.71. Voting and the NVMe cells coincide with RECITE and were therefore not rerun, since every voting session finishes and the rule chooses save on NVMe.

E.3 PREDICTIONS AGAINST MEASUREMENTS

Reuse distance by workflow class. Lemma 1 turns three common patterns into closed forms (Table 6). *Voting* (single-turn sessions) has no revisits: every eviction is of finished state, so under liveness the workload is restore-free and under save-everything every write is dead. *Debate* with S debaters per question satisfies $D_t \le S - 1$ for every revisit, with equality when a round keeps its order, so $\phi(K) = 0$ for all $K \ge S$: three debaters against four slots never restore in trace order, and the restores the main matrix logs on debate come from batching, which merges same-role turns of different questions and widens the effective reuse distance. *Episodic GUI* traces have an empirical tail, and $\phi(K)$ read off the histogram reproduces the simulated LRU misses at every K. The four-slot entry for the five-workflow trace is the two restores the main matrix logged; finished-first eviction can only lower the others (Table 7). Capacity is sharp in K: three workflows against four slots move zero restore bytes, the five-workflow trace logged the two predicted restores, and seven workflows against four slots still sit at parity (1.005×).

Distance from clairvoyant. LRU with K slots is $K/(K - h + 1)$-competitive against an offline optimum with h slots (Sleator & Tarjan, 1985), a factor K at $h = K$. On the recorded traces the gap is far smaller: replaying each trace under liveness-first LRU and under Belady's furthest-next-use rule (Belady, 1966), with finished sessions treated as never used again, liveness-first LRU restores at most 2.25× as often as the clairvoyant policy at $K \ge 3$, exactly twice at $K = 4$, and matches it on gui-3, debate, and vote (Table 7). The gap is larger in bytes because the clairvoyant rule evicts the largest session that is about to finish; Appendix C implements exactly that choice as the next-use victim and finds that on this engine its batch-order tax outweighs the bytes it saves on debate while it gains 9% on the GUI eMMC cell.

Prospective test. The near-threshold direction was tested before the cells ran. On the 64 GB board's cold eMMC path the day's rates of $r_r = 289$ and $r_w = 115$ MB/s put $r_p^\star$ at 557 tokens/s, below the 678–1235 tokens/s the board prefills at 32K down to 2K tokens, so the frozen prediction was recompute at every context, and 21 of 21 paired measurements agreed. The gap narrows from 2.4× at 2K to 1.09× at 32K, so the boundary lies beyond 32K and was not crossed.

Table 6: Reuse-distance histograms of the recorded traces and the LRU restore counts Lemma 1 predicts from them at equal slots; "observed" is the count the main matrix recorded, and gui-5 at four slots is its one cell with LRU restores.

Trace	D_t histogram	K=2	K=3	K=4	Class
gui-3 (57 revisits)	$\{0{:}51, 1{:}3, 2{:}3\}$	3	0	0	episodic
gui-5 (95 revisits)	$\{0{:}86, 1{:}3, 2{:}1, 3{:}3, 4{:}2\}$	6	5	2 (2 observed)	episodic
gui-7 (133 revisits)	$\{0{:}120, 1{:}1, 2{:}2, 3{:}3, 4{:}3, 5{:}3, 6{:}1\}$	12	10	7	episodic
debate (60 revisits)	$\{0{:}6, 1{:}14, 2{:}40\}$	40	0	0	$D_t \le S-1 = 2$
vote (0 revisits)	–	0	0	0	single-turn

Table 7: Restores under plain LRU, liveness-first LRU (RECITE), and the clairvoyant rule, as count / tokens reloaded on the recorded traces; rows with no restores under any policy are omitted.

Trace	K	LRU	liveness-first LRU	clairvoyant
gui-3	2	3 / 28.2K	2 / 16.3K	2 / 16.3K
gui-5	2	6 / 58.9K	5 / 47.9K	4 / 34.2K
gui-5	3	5 / 47.9K	3 / 22.3K	2 / 6.3K
gui-5	4	2 / 16.6K	2 / 16.6K	1 / 0.6K
gui-7	2	12 / 95.8K	11 / 89.0K	8 / 64.6K
gui-7	3	10 / 70.3K	9 / 62.4K	4 / 22.0K
gui-7	4	7 / 48.0K	4 / 21.1K	2 / 3.5K
debate	2	40 / 14.3K	27 / 9.6K	20 / 7.1K

Calibration protocol behind Figure 2d. The 62 held-out cells and the 26 calibration cells make up the 88 cells of Figure 2d. Rates are calibrated once, on gui-3 and debate; these calibration cells all sit below 1% restore share, so their own R^2 is uninformative. There r_p and r_d are the token-weighted prefill and decode rates in those cells' per-turn logs (735 and 25.7 tokens/s on the 32 GB board); the 8 GB board, the Llama family, the phone, and the crossover sweep use their own independently probed constants. For every other cell the prediction uses only the trace's token counts, κ, the tier's rates, and a deterministic replay of residency (finished sessions evicted first, then LRU over K slots) that yields the bytes written and read back and the tokens replayed; T_{turn} is $N_{\text{prefill}}/r_p + N_{\text{decode}}/r_d$, and R follows from Equation (1). The model is serial accounting, so its known biases show in the residuals: batched decode makes debate and vote turns 1.8–2.1× faster than the single-stream rates predict, GUI turns run 1.26–1.30× slower, the 8 GB board's token-pool eviction is under-replayed, 0.67 GB of predicted restore traffic on gui-3 against 3.26 logged, and the phone's sub-second save and load carry a fixed latency the bandwidth term omits. None of these moves a cell across a decision boundary, and the large-R cells that decide whether management pays are predicted within 0.07 in R. Table 8 lists every decision for which both arms were measured; the rule picks the faster arm 14 times out of 14. Per device group the held-out mean absolute error in R is 0.006 (32 GB main matrix, 26 cells), 0.014 (8 GB board, 11), 0.022 (64 GB crossover, 7), 0.003 (Llama, 10), and 0.009 (phone, 3); the within-group R^2 is 0.99 on the 32 GB matrix and on Llama but near zero on the 8 GB board and the phone, whose cells all sit below 5% restore share, so the pooled R^2 of 0.993 measures separation between regimes, not precision inside the neutral zone.

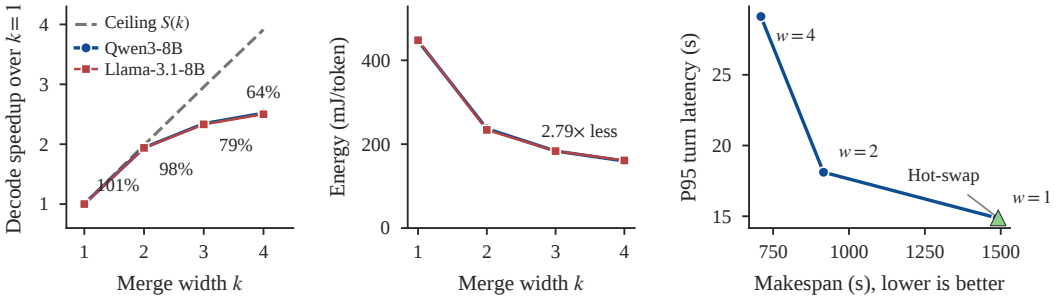
Merge width and energy roofline. Figure 9 plots the width sweep behind Section 4.4.

Energy roofline for batching. Energy per generated token decomposes as $E(k) = P_{\text{idle}}/\text{tps}(k) + e(k)$, with P_{idle} the board's idle draw and $e(k)$ the dynamic energy per token; a bandwidth-bound step spends its dynamic energy moving weights and KV, so $e(k) = e_w/k + e_{kv}$ with the weight term amortized over the k merged streams. On the width microbenchmark P_{idle} reads 3.57 W at every width (sd 0.001), and fitting $e_w = 0.2794$ and $e_{kv} = 0.0223$ J per token on widths one and two predicts the energy per token at widths three and four within 3.4% and 6.3% for Qwen3-8B and 5.4% and 9.7% for Llama-3.1-8B (Table 9); relative to width one it predicts a 2.978× reduction at width four against the measured 2.79×, and 3.452× if decode reached the memory-traffic ceiling, so the energy gain is the throughput gain times the amortized weight-read energy, and it saturates exactly where Proposition 3 does.

Merge-width detail. Figure 9 plots the width sweep; merging four ready turns lifts decode from 24.5 to 61.8 tokens/s, and width one ties HOT-SWAP at 1488.4 s, while the ladder 1.01, 1.94, 2.35, 2.52× at widths one to four attains 101, 98, 79, and 64% of the ceiling 1.00, 1.99, 2.96, 3.91×. A

Table 8: Save-or-recompute decisions with both arms measured (board cells: means of three runs; contexts: medians of three). Penalty is the slowdown factor the other arm would have cost. The Llama eMMC save arm is the parity HOT-SWAP cell.

Board / tier	Case	Rule	Save arm (s)	Recompute arm (s)	Penalty
32 GB NVMe	gui-5, Qwen3-8B	save	237.7	259.1	1.09×
32 GB eMMC	gui-5, Qwen3-8B	recompute	543.1	259.0	2.10×
32 GB NVMe	gui-5, Llama-3.1-8B	save	222.7	242.2	1.09×
32 GB eMMC	gui-5, Llama-3.1-8B	recompute	596.5	242.4	2.46×
64 GB eMMC, cold	2K-token session	recompute	3.92	1.62	2.42×
64 GB eMMC, cold	4K	recompute	7.64	3.87	1.97×
64 GB eMMC, cold	8K	recompute	13.88	8.75	1.59×
64 GB eMMC, cold	12K	recompute	19.23	14.09	1.36×
64 GB eMMC, cold	16K	recompute	25.45	19.94	1.28×
64 GB eMMC, cold	24K	recompute	38.25	32.18	1.19×
64 GB eMMC, cold	32K	recompute	51.49	47.20	1.09×
Phone UFS, cold	2K-token session	save	0.58	14.8	25.7×
Phone UFS, cold	8K	save	1.27	108.5	85.2×
Phone UFS, cold	16K	save	2.74	401.9	146.5×



(a) Speedup vs. ceiling **(b)** Energy per token **(c)** Latency vs. makespan (debate)

Figure 9: (a) Measured decode speedup of both model families against the ceiling $S(k)$ of Proposition 3, attainment annotated for Qwen3-8B; (b) energy per token against merge width; (c) the latency–makespan trade the width knob sets on debate.

separate throughput microbenchmark reads 1.94–2.52×, the single-sequence control reproducing itself exactly; the width-four median turn is 0.27 s, so the latency cost concentrates on lead turns and P95 is the right provisioning metric. A verified low-power mode moves four repeated cells at most 2.6%, which is inside the run-to-run variance of the matrix.

E.4 GENERALITY

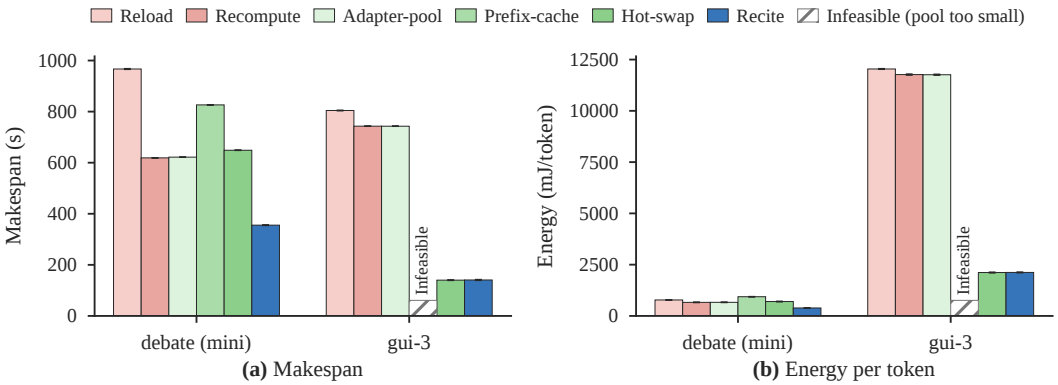
8 GB board. Figure 10 plots the Orin Nano Super cells of Section 4.5.

Phone detail. The phone's UFS reads at 2120 and writes at 2103 MB/s cold, an asymmetry of 1.01, so $\alpha^\star$ spans 36–429 across the GPU's 505 tokens/s and the CPU's 43; a cold save-and-load round trip of an 8K-token session takes 1.5 s against 106 s of CPU replay, 74× over five repetitions. The 1.7 B model prefills at 121 tokens/s on the phone's CPU at 2K context, 80 at 8K, and 43 at 16K (505 on the GPU at 2K); over five cold-start repetitions the thermal spread keeps writes within 11% and cold reads within 2.4×, against a two-orders margin.

Live-loop detail. Engine time falls from 152.6 to 78.3 s, and wall clock (304.5/376.7/176.8 s) decomposes as Figure 11 shows to within 7 s, with liveness taken from the orchestrator's episode-completion events; per step the gap holds (engine 3.70 against 2.04 s; energy 119 against 82 J), and the observation critical path halves from 104 to 50 s because all three agents are served per loop and the next three observations run in parallel. Task completion is the base model's zero-shot competence and identical across systems.

Table 9: Energy roofline: dynamic (e) and total (E) energy per token, J, measured against the two-parameter model fitted on widths one and two.

Family	k	tokens/s	e meas.	e pred.	E meas.	E pred.	error
Qwen3-8B	1	24.7	0.302	0.302	0.446	0.446	+0.0%
Qwen3-8B	2	47.6	0.162	0.162	0.237	0.237	+0.0%
Qwen3-8B	3	57.5	0.122	0.115	0.184	0.177	-3.4%
Qwen3-8B	4	61.8	0.102	0.092	0.160	0.150	-6.3%
Llama-3.1-8B	1	25.5	0.300	0.300	0.448	0.448	-0.0%
Llama-3.1-8B	2	49.5	0.158	0.158	0.234	0.234	-0.0%
Llama-3.1-8B	3	59.5	0.120	0.110	0.183	0.174	-5.4%
Llama-3.1-8B	4	63.8	0.102	0.087	0.161	0.146	-9.7%



(a) Makespan **(b)** Energy per token

Figure 10: Orin Nano Super 8 GB: (a) makespan and (b) energy. The hatched stub marks the one infeasible cell, where the unbudgeted PREFIX-CACHE pool overflows on three concurrent GUI sessions of 27K tokens.

Second family detail. Llama's architecture fixes κ at 128 KiB per token, which the load-time probe measures at 128.016. On the second family the gui-5/eMMC restore-policy gain measures 2.46× (596.5 against 242.5 s). With $\kappa = 128.016$ KiB per token the runtime makes the predicted decisions on Llama-3.1-8B (2.16 GB written and read back on NVMe; zero bytes and two replays on eMMC); on NVMe, forcing recompute instead of the rule's save costs 8.8% (242.2 against 222.7 s) and save-everything 1.7%. The decode ladder reads 1.94, 2.33, and 2.50× at widths two to four against 1.94, 2.35, and 2.52 for Qwen, and voting finishes 2.33× faster than HOT-SWAP (822 against 1915 s) at 2.55× less energy per token.

Model axis. A 4 B model raises restore share to 8.1%, an 8-bit KV tier lowers it to 2.7%, and a 35 B mixture-of-experts checkpoint at 80 KiB of KV per token sits in the neutral zone at 1.005×, each as Equation (1) predicts from its κ and prefill rate.

External system. Table 10 gives the EdgeLoRA cells of Section 4.5. We build EdgeLoRA from its artifact and run it with the same five rank-16 adapters and the same re-tokenized traces as RECITE; it serves four 32K slots with its prompt cache enabled and its adapters selected explicitly per turn, so its router is not on the critical path, and run-to-run spread is below 0.6% except for EdgeLoRA on gui-5/NVMe, where it is 2.2%. A fifth EdgeLoRA slot, which fits once its micro-batch is cut from 512 to 128, removes every GUI eviction. The five-workflow cell falls from 18,733 to 3030 s, a 6.2× gain from removing capacity pressure alone (three runs, spread 0.7%). The remaining 13.6× against RECITE is the engine gap that voting already exposed. Four slots is thus a choice, and its price is a quarter of the micro-batch.

F LEARNED ALTERNATIVES TO THE RULES

We tested learned replacements for both decisions offline on the measurements already in hand and then on the board (Table 1).

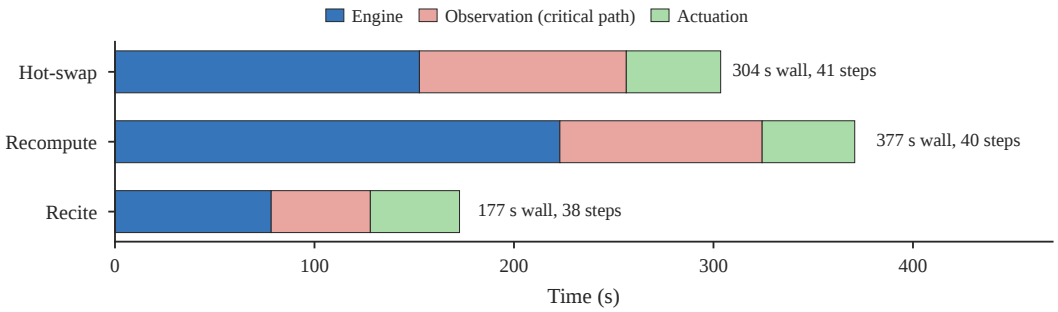


Figure 11: Live three-agent GUI loop, means of three runs: wall clock decomposes into engine time, the observation critical path, and actuation.

Table 10: EdgeLoRA against a same-day RECITE reference on the 32 GB board: Llama-3.1-8B, four 32K slots, interleaved runs. Replayed tokens (EdgeLoRA / RECITE) are the prefill work each system repeats because state it needs is gone.

Workload	Tier	EdgeLoRA (s)	RECITE (s)	Ratio	Replayed tokens
vote	NVMe	2734	821	3.33×	0 / 0
debate	NVMe	2223	680	3.27×	81,041 / 0
gui-5	NVMe	18,733	223	83.9×	324,807 / 0
gui-5	eMMC	19,045	243	78.5×	324,807 / 16,455

Learned save-or-recompute. Every measured pair of save-and-read-back against prefill we have (21 from the 64 GB eMMC sweep, 15 from the phone, four from the 32 GB board cells with both arms; six device–tier groups) trains three logistic classifiers evaluated leave-one-device-tier-out (Table 11). A length-only learner is worse than chance across devices, because session length is exactly the coordinate that cancels; a learner given all five device features fits the training devices perfectly but transfers to 28 of 40, with only six distinct rate vectors to learn from; a learner handed the rule's own margin as its single feature reaches 38 of 40. The rule, with no fitted parameter, decides 40 of 40 measured pairs. The lesson is not that learning cannot work but that the transferable structure is the ratio $r_r/(\kappa r_p)$ against $\alpha + 1$, which a learner must be given or needs many device–tier groups to discover, and which the closed form supplies without data.

Learned liveness. A logistic model over trace features (role, turns completed, round tag, prompt length, a completion keyword) predicts whether a session has a future turn (Table 13). Trained on gui-3 and debate it recovers every terminal event of debate and, once voting is in the training set, of voting, since both are structural; it recovers none of the GUI episode ends on gui-5 and gui-7, which no positional feature announces. A model trained without voting predicts every voter alive: a new workflow type is out of distribution for the learner and free for the DAG signal. Both replacements then ran on the 32 GB board against a same-day RECITE reference (Table 12). The learned liveness model lands inside its predicted interval on all four cells: it never drops a live session, recovers every terminal event of debate and voting, and misses all five GUI episode ends, which costs 2.7 GB of writes that are never read back and 2.1% on NVMe. A TTL-only ablation of Continuum's pinning (Li et al., 2025), which pins a session's state for a multiple of its reload cost, behaves as predicted once the multiple is four or more, 1.5–2.7% slower on voting and debate, from 12.5 and 19.7 GB of dead writes. At a multiple of one the pins expire before the next turn and victim order degrades, 1.10–1.22× with a 108 s run-to-run spread on voting, and on the GUI eMMC cell it is 9% *faster* than RECITE: the pin happens to protect the session that liveness-first LRU would have evicted, so the trace's two long replays shrink to 553 tokens, exactly the clairvoyant choice of Table 7. A heuristic can land on the right victim by accident. Continuum itself also frees KV when a program's last request finishes; at our role-session granularity that release would arrive only when the whole workflow ends, after every debater session's state has already been written once, so the ablation isolates the pin and not Continuum's end-of-program release. Learned eviction at the token level (Moschella et al., 2026) and learned prefix caching (Yang et al., 2025) are complementary: they choose what to keep inside a resident session, not whether an evicted session is written to storage at all.

Table 11: Learned save-or-recompute classifiers against the tier rule on 40 measured pairs from six device–tier groups, evaluated leave-one-device-tier-out.

Decision policy	Held-out correct	In-sample	Fitted parameters
length only (log n)	2/40	23/40	2
all device features	28/40	40/40	6
rule margin as feature	38/40	40/40	2
tier rule (no fitting)	40/40	40/40	0

Table 12: Policy baselines from the literature, reimplemented inside the control plane, on the 32 GB board against a same-day RECITE reference: makespan relative to RECITE, with the bytes each policy writes that are never read back. TTL multiples are expressed in units of the reload cost of the pinned session.

Trace	Tier	RECITE (s)	TTL $\times 1$	TTL $\times 4$	TTL $\times 16$	learned liveness
vote	NVMe	842.8	1.22 (12.5 GB)	1.02 (12.5)	1.02 (12.5)	1.01 (12.5)
debate	NVMe	709.5	1.10 (19.7)	1.03 (19.7)	1.03 (19.7)	1.00 (0)
gui-5	NVMe	237.0	1.02 (2.7)	1.02 (2.7)	1.00 (0.3)	1.02 (2.7)
gui-5	eMMC	258.1	1.00 (0)	0.91 (0)	0.91 (0)	1.00 (0)

G EXTENDED RELATED WORK

KV reuse for multi-agent LLMs. Agent-aware servers prefetch, pin, share, or repair KV across the steps and roles of a workflow, co-schedule it with agent execution around tool idle, session affinity, and prefix retention, or let concurrent agents attend to one cache (Pan et al., 2025; Li et al., 2025; Ye et al., 2025; Liu et al., 2026; Bian et al., 2025; Guo et al., 2026; Ni et al., 2026; Zheng et al., 2026; Zhang et al., 2026b; Rodionov et al., 2025; Hsu et al., 2025). ICaRus and LRAgent share caches across task-specialized adapters by freezing the key and value projections, the exact form of the placement our exploratory cross-role study approximates (Woo et al., 2026a; Jeon et al., 2026). Continuum also frees KV when a program's last request finishes (Li et al., 2025). All keep state in GPU or CPU memory, where writes and reads cost about the same, and before the program ends none decides whether a session leaving memory across an asymmetric flash boundary should be written to storage in the first place.

On-device and multi-adapter serving. EdgeLoRA serves multi-tenant adapters for stateless requests (Shen et al., 2025b), CoServe and D2MoE swap expert weights on a router's fixed order (Suo et al., 2025; Wang et al., 2025), and the wider on-device line optimizes single-model inference (Xu et al., 2023; Guo et al., 2023; Xue et al., 2024; Yi et al., 2025; Liu et al., 2024b; Lin et al., 2024; Laskaridis et al., 2024); S-LoRA and Punica batch adapter requests through segmented kernels (Sheng et al., 2024; Chen et al., 2024); RadixAttention reuses prefixes blind to storage tiers and turn structure (Zheng et al., 2024); Agent-X accelerates a single on-device agent pipeline (Chung et al., 2026); Agent Memory persists per-agent 4-bit KV on an Apple M4 Pro (Shkolnikov, 2026); Halo consolidates the DAGs of batched agentic queries on a server GPU (Shen et al., 2025a); agent frameworks fix the DAG and treat turn order as application logic (Wu et al., 2024; Hong et al., 2024; Yao et al., 2023; Shinn et al., 2023; Li et al., 2023; Qian et al., 2024; Zhang et al., 2025a; Wang et al., 2024), and prompt-level pruning and supervision cut multi-agent cost upstream of serving (Zhang et al., 2025b; Lin et al., 2026). Work-conserving scheduling is throughput-optimal for DAG and fork-join agent workloads in a queueing model (Dai et al., 2025). Lemma 2 concerns restore cost at one slot; throughput optimality is a different question.

KV offload and compute-or-load. Serving engines swap or recompute on preemption (Kwon et al., 2023; Abhyankar et al., 2024); CachedAttention, CacheGen, LMCache, and Mooncake persist, compress, or disaggregate KV across tiers (Gao et al., 2024; Liu et al., 2024a; 2025; Qin et al., 2025); CacheFlow, Tutti, and FlashGen restore persisted KV with parallel I/O and recompute when retrieval is slower (Nian et al., 2026; Qiu et al., 2026; Jeong & Ahn, 2025); Cake overlaps compute and load (Jin et al., 2025); IMPRESS loads only important prefix KV when solid-state drive (SSD) reads dominate (Chen et al., 2025); a critical cached-to-prefill ratio locates the memory-bound crossover for PCIe offload (Meng et al., 2026); and an I/O characterization of offloading measured KV reads at 2.0 GiB/s against writes at 11 MiB/s, a 180$\times$ asymmetry it reports but no policy prices (Ren et al., 2025). These systems charge both directions of a symmetric GPU–CPU swap (Abhyankar et al.,

Table 13: Learned liveness: configuration A trains on gui-3 and debate, configuration B adds voting; "dead recall" is the share of terminal events the model detects, "false dead" the live turns it would have dropped without saving them.

Config	Trace	Split	Terminal events	Accuracy	Dead recall	False dead
A	vote	test	120	0.0	0/120	0
B	gui-3	train	3	0.95	0/3	0
B	debate	train	60	1.0	60/60	0
B	vote	train	120	1.0	120/120	0
B	gui-5	test	5	0.95	0/5	0
B	gui-7	test	7	0.95	0/7	0

2024) or hide write-back behind compute (Gao et al., 2024). On a mobile tier the write side dominates, overlapping cannot recover it, and our rule prices it at eviction with the session's liveness. KVSwap is the nearest on-device system, serving one context that does not fit memory by writing each token's KV once (Zhang et al., 2026a); the two are orthogonal and composable.

H Workloads, Reference Configurations, and Artifact

Traces. Every workload is a JSONL trace of turns with fields `turn_id`, `agent_id` (role instance, keyed with the workflow), `deps`, `mergeable_group`, `prompt_parts` (references, not inlined text, so prefix structure is computable at the schema level), and `n_max`. `debate`: 20 grade-school questions, three debaters, two rounds, 60 sessions of two turns. `vote`: 20 questions, five voters and a judge, 120 single-turn voter sessions. `gui-3/5/7`: three, five, and seven concurrent workflows built from real Android episodes and interleaved by us into concurrent sessions reaching 27K tokens, sessions interleaving in episode segments averaging 4.9 steps with a session's next segment beginning on average 8.1 turns after its last (heavy tail to 29). `linear`: an 11-turn pipeline over five roles with no mergeable structure, the negative control.

Reference configurations. No engine we know of serves multiple adapters over a quantized model with unified-memory KV, so every reference configuration is a configuration of ours (external systems: Section 4.5); a bare engine turn decodes at 22.9–23.1 tokens/s against the platform's published 23.9–25.3. Reload relaunches the process per role switch, 28.5 s of reloads over the 11-turn pipeline. Hot-swap keeps every adapter registered, serves sessions in FCFS order, saves and loads KV per eviction under the same adaptive restore path and the same four slots as Recite. Recompute replays each session's full history including generated outputs per turn, with replayed tokens logged. Prefix-cache keeps four slots over a unified 131K-cell pool, evicts only by LRU drop, persists nothing, and re-prefills what the longest resident prefix misses; at 8 GB our implementation of it has no byte budget, so sessions reaching 31–35K tokens overflow the pool, where a budgeted prefix cache would evict and re-prefill instead. Adapter-pool is an adapter-LRU pool over stateless requests. Storage-free configurations never touch storage and are therefore measured once, on the NVMe tier.

Artifact. The engine, the control plane, the traces, the energy sampler, the batch runners behind the main matrix, the aggregation and figure scripts, the frozen-number assertion suite, and the prediction ledger are released with MD5 manifests. Reproducing the paper needs a board with NVMe and eMMC, power rails, a JetPack-matched CUDA toolchain, Python 3.10+, and fio; about 60 GB of disk, 2 hours to build, and 40 hours for the matrix. Ratios hold within a few percent across reruns; absolutes move with thermals and flash wear.

I Limitations

The main matrix replays recorded turns; the live GUI loop and the live workflows show the gain surviving tool latency and sampling, but larger live deployments are open. The largest end-to-end model is 8 B on one board family, distinct per-role models reduce to Reload or Recompute, and cross-role prefix reuse is exploratory. Recite schedules turns, not tokens, on one device; makespan is a batch metric, the width knob raises P95 turn latency, and batch-order sensitivity is an engine property (Appendix C). The tier rule prices every write at the device rate, so bursts the page cache absorbs are cheaper than it assumes, which made it conservative by 1.8% on debate over eMMC; flash endurance is unmeasured.