KVLEASE: DYNAMIC KV MEMORY SHARING FOR MIXTURE-OF-MODELS INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model applications can combine several models to answer one request. In this Mixture of Models (MoM) setting, workflows connect parallel and sequential model calls through their intermediate outputs. Stage transitions shift key–value (KV) cache demand, but overlapping requests keep earlier-stage engines busy. A downstream model can therefore have its inputs ready yet wait for capacity that those engines repeatedly reuse, delaying the complete answer. Fixed partitions strand capacity; lending idle memory alone cannot recover pages from busy borrowers. The challenge is to serve the next stage without interrupting existing uses. We present KVLease, a KV-memory sharing system designed for MoM workflows. It couples lending and per-model allocation guarantees with targeted return: selected borrowed pages stop accepting new allocations, then return after existing references and GPU accesses finish. Capacity is shared per GPU; model-specific KV contents stay private. Our analysis characterizes how workflow stages correlate memory demand, separates partition imbalance from temporal sharing, and establishes conditional return bounds. Replicated evaluations across heterogeneous GPU platforms, serving engines (e.g., vLLM and SGLang), and diverse MoM workloads show 28–53% lower end-to-end p95 latency than equal-token fixed allocations at the reported memory-constrained operating points, over 50% higher completed-request throughput, and up to 35 percentage points higher service-level objective (SLO) attainment. Against kvcached at matched capacity, p95 falls a further 24–31% at high load. These gains preserve model computation, with identical outputs in paired correctness checks.

1 INTRODUCTION

We define *Mixture of Models (MoM)* as a serving setting in which heterogeneous language models collaborate on user requests behind one endpoint. Each request follows a *workflow*: an arrangement of model calls and dependencies that determines which calls run in parallel and how their outputs feed later calls. Routing, cascades, debate, and aggregation provide different ways to organize these calls (Du et al., 2024; Wang et al., 2025). For example, several models may propose answers in parallel before another combines their outputs. The user receives an answer only after the required calls finish, making their joint execution the unit of serving performance.

These dependencies also shape memory demand. Proposers need capacity first; aggregator demand rises when their outputs arrive. Yet a natural deployment assigns a separate inference engine and key–value (KV) cache allocation to each model. Engines such as vLLM and SGLang efficiently batch their own calls (Kwon et al., 2023; Zheng et al., 2024), but fixed allocations cannot follow this shift. One model can exhaust its slice while another holds unused capacity, delaying the complete answer.

The mismatch is substantial. In our three-seed mechanism measurements, some model is memory-capped for 50–83% of sampled time under fixed slices, while 57–61% of the pool is unused elsewhere (Appendix B.7). In the primary B200 workload, allowing memory to follow demand lowers 95th-percentile (p95) latency from 129.0 to 70.1 seconds across five paired seeds.

Sharing creates an obligation to return. Proposers can borrow spare capacity while the aggregator is idle. When aggregator demand rises, overlapping requests may keep proposers busy and reusing those borrowed pages. The complete answer then waits for memory even though its proposer outputs are ready. Waiting for engine idleness need not resolve this under continuous load. Existing systems

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.

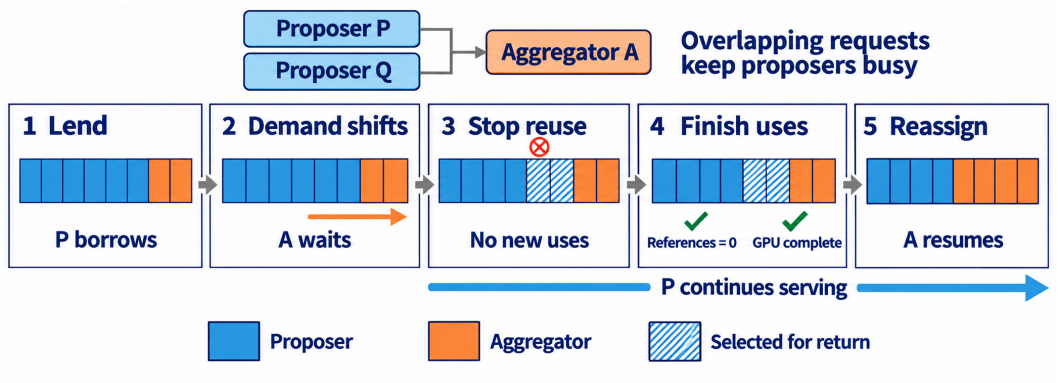

Figure 1: **Return capacity while the borrower remains busy.** Proposers borrow before aggregator demand rises. Hatched pages stop accepting new uses; their capacity returns after references and GPU accesses finish. The schematic tracks P and A; other proposer work continues. Colors denote capacity ownership, not interchangeable KV contents.

provide adaptive quotas and elastic allocation (Duan et al., 2024; Xing et al., 2025; Yu et al., 2026); changing a quota alone does not release pages serving live computations. The allocation contract must connect return decisions to the page lifecycle.

KVLease coordinates KV-memory lending and return across independently scheduled engines, allowing capacity to follow changing MoM demand without interrupting existing uses. Three mechanisms address the challenge. Lending and per-model guarantees prioritize capacity as demand shifts. Targeted return stops new allocations to selected borrowed pages, so busy engines cannot recycle them indefinitely. Shared accounting, reference tracking, and GPU-completion checks determine when capacity can be reassigned. Capacity is shared per GPU; each model keeps its own KV contents.

We call this contract a *capacity lease*. Guarantees give priority to available bytes; return depends on existing uses finishing. In the cumulative ablation, targeted return reduces mean below-guarantee episode duration from 44.7 to 13.8 seconds on B200 and from 160.3 to 27.6 seconds on RTX. Our analysis links workflow overlap to demand covariance, separates partition imbalance from temporal sharing, and establishes conditional return bounds. Five-seed comparisons show 24–31% lower p95 than kvcached at the two higher B200 rates and 28% at the highest RTX rate. The contribution is the enforced lending-and-return contract and its MoM analysis; Section 5.4 distinguishes it from individual policy choices.

2 The Memory Cost of Collaboration

Demand follows dependencies. A request is a directed acyclic graph of model calls. Parallel calls raise demand together; later stages shift it elsewhere. Across overlapping requests, topology and call lifetimes determine aggregate KV footprints. A partition fitted to one workload mixture can therefore become mismatched after a shift.

Joins expose the slowest component. A *join* waits for parallel calls to finish before the workflow can proceed. A parallel stage completes at $T_{\text{stage}} = \max_m T_m$, so $\Pr(T_{\text{stage}} > t) \geq \max_m \Pr(T_m > t)$. A memory-starved branch can govern the user-visible tail even when all other branches are fast. In a single-replication matched-volume diagnostic at two arrival rates, fixed-slice MoM workflows have 16–43% higher p95 than independent-call controls. Pooling helps both traffic types; collaboration compounds the shared-memory problem with dependency-induced delay.

Memory also buys execution efficiency. Insufficient KV capacity forces smaller batches or recomputation of preempted prefixes. Across the three primary evaluation settings, pooling raises generated tokens per engine step from 35.9 to 47.1, 36.6 to 45.7, and 32.6 to 53.1: 20–39% fewer steps per generated token. More requests share each execution step, queues shrink, and workflows reach their

joins sooner. The following theory identifies the recoverable memory opportunity; Section 5 measures the resulting latency and throughput.

3 How workflows shape the memory opportunity

Two workflows with the same model-call counts can have different memory-sharing opportunities: parallel branches overlap, whereas a barrier separates their footprints. We model this distinction explicitly, then identify the pressure an allocator can recover. Proofs appear in Appendix A.

3.1 From workflow templates to correlated demand

A class-c request arriving at t_r contributes a nonnegative vector $h^{(c,r)}(u)$ of KV bytes at lag u. It includes the timing and footprint of every stage. Offered demand is $d_m(t) = \sum_c \sum_{r\in c} h_m^{(c,r)}(t - t_r)$. Conditional on the class-rate paths $\lambda_c(t)$, assume independent Poisson arrivals and independent, identically distributed templates within each class, with finite integrated first and second moments. The templates fix residence times for this analytical model.

Proposition 1 (Workflow covariance)**.** *Let $H_m^{(c)}(u) = \mathbb{E}[h_m^{(c)}(u)]$ and $Q_{mn}^{(c)}(u) = \mathbb{E}[h_m^{(c)}(u)h_n^{(c)}(u)]$. The mean $\mu(t)$ and covariance $\Sigma(t)$ of $d(t)$ are*

$$\mu_m(t) = \sum_c \int_0^\infty \lambda_c(t-u) H_m^{(c)}(u)\, du, \tag{1}$$

$$\Sigma_{mn}(t) = \sum_c \int_0^\infty \lambda_c(t-u) Q_{mn}^{(c)}(u)\, du. \tag{2}$$

Sampling an independent time T uniformly over the observation window gives $\mathrm{Cov}(d(T)) = \mathcal{S} + \mathcal{R}$, where $\mathcal{S} = \mathbb{E}_T[\Sigma(T)]$ is the within-request term and $\mathcal{R} = \mathrm{Cov}_T(\mu(T))$ is the changing-mix term.

This specializes Campbell's formula and total covariance (Last & Penrose, 2017). All entries of $\mathcal{S}$ are nonnegative. For two models confined to disjoint stages, $h_m h_n = 0$ and their within-request term vanishes; parallel branches contribute in proportion to their footprint overlap. A one-model router has diagonal $\mathcal{S}$. Changes in class mix can instead make entries of $\mathcal{R}$ negative, when demand rotates between model sets. The convolution retains stage delays and smooths arrival shifts rather than treating each phase as an instantaneous demand change.

Consequence for pooling. For stationary class proportions π_c and total rate Λ, define $W = \sum_c \pi_c \int Q^{(c)}(u)\, du$. Then $\Sigma = \Lambda W$. With means μ_m, deviations σ_m, and a common safety factor $k > 0$, define the sizing ratio

$$\Gamma_k = \frac{\sum_m \mu_m + k \sum_m \sigma_m}{\sum_m \mu_m + k\sqrt{\mathbf{1}^\top \Sigma \mathbf{1}}}. \tag{3}$$

As $\Lambda \downarrow 0$, its limit is

$$\Gamma_0 = \frac{\sum_m \sqrt{W_{mm}}}{\sqrt{\mathbf{1}^\top W \mathbf{1}}} \le \frac{\sum_m \sqrt{W_{mm}}}{\sqrt{\mathrm{tr}\, W}}, \tag{4}$$

strictly when some off-diagonal $W_{mn} > 0$. Thus simultaneous collaboration reduces statistical multiplexing relative to independent traffic with the same marginals. Stage transitions create a different opportunity: transferring capacity toward downstream models while earlier-stage engines serve other requests. Using light-load templates from a different seed, predicted correlation patterns have Pearson coefficients of 0.93–1.00, with median absolute error 0.093 across 36 comparisons from six target jobs. Appendix A.6 separates this transfer test from same-execution reconstruction.

3.2 Separate partition shape from recoverable pressure

For M engines with prescribed nonzero phase demands $d_m(p) \ge 0$, the minimal static and pooled capacities are

$$C^*_{\text{static}} = \sum_m \max_p d_m(p), \quad C^*_{\text{pool}} = \max_p \sum_m d_m(p), \quad \Gamma = C^*_{\text{static}}/C^*_{\text{pool}} \in [1, M]. \tag{5}$$

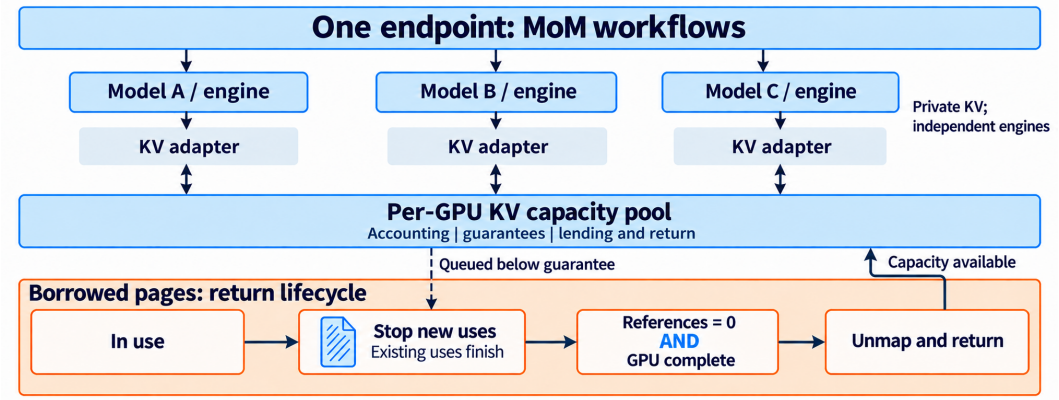

Figure 2: **Independent engines, coordinated capacity.** Engine-local adapters enforce page return. Atomic accounting prevents oversubscription; capacity is refunded only after references reach zero and outstanding GPU accesses complete. Each GPU has its own pool, and KV values remain private to their model.

For $d_m(p) = \Lambda v_m(p)$, a positive split S with $\sum_m S_m = C$ has threshold $\Lambda_S = \min_{m,p:v_m(p)>0} S_m/v_m(p)$. The best static split and ideal pool have thresholds $\Lambda_{\text{best}} = C/\sum_m \max_p v_m(p)$ and $\Lambda_{\text{pool}} = C/\max_p \sum_m v_m(p)$. Consequently,

$$\frac{\Lambda_{\text{pool}}}{\Lambda_S} = \underbrace{\frac{\Lambda_{\text{best}}}{\Lambda_S}}_{\text{partition-shape gain } \geq 1} \quad \underbrace{\frac{\Lambda_{\text{pool}}}{\Lambda_{\text{best}}}}_{\Gamma\text{: temporal multiplexing}} . \tag{6}$$

A fitted partition captures the first factor when its profile remains representative. Dynamic allocation can additionally follow changes in demand. These thresholds concern memory feasibility at fixed residence times; they are not latency speedups.

For a deployed split, define overflow $O(t) = \sum_m [d_m(t) - S_m]_+$ and idle capacity $I(t) = \sum_m [S_m - d_m(t)]_+$. Their recoverable part obeys the accounting identity

$$R(t) = \min\{O(t), I(t)\} = O(t) - \left[\sum_m d_m(t) - C\right]_+ . \tag{7}$$

Its integral is the excess byte-seconds removable by instantaneous sharing on the same trace. This connects sizing to operation: sharing matters when overflow and idle capacity coincide. We estimate that opportunity from fixed-slice telemetry in Appendix B.7 and measure the resulting queueing and latency directly.

The online requirement is return. Idle lending alone cannot recover pages that a busy borrower continuously recycles. A fixed split's optimal worst-case service ratio is $1/M$; reaching a feasible allocation within τ of a phase change preserves at least $1 - \tau/P$ of ideal byte-time service for a phase of length P (Appendix A). These bounds motivate an explicit return mechanism. The implementation must stop new references to selected borrowed pages while allowing existing users to finish.

4 KVLEASE: A RECOVERABLE CAPACITY CONTRACT

KVLease exposes one endpoint over resident model engines. Each GPU has one shared physical-capacity budget. A lease lets an engine temporarily exceed its guarantee; waiting demand triggers return. The return path uses engine demand and references, without requiring a workflow graph; MoM dependencies motivate it and define the end-to-end objective. Reference tracking and GPU-completion checks govern page retirement (Figure 2).

4.1 ATOMIC CAPACITY ACCOUNTING ACROSS ENGINES

Following the virtual/physical separation used by vAttention and kvcached (Prabhu et al., 2025; Xing et al., 2025), each engine reserves a stable virtual KV address range. Physical pages are mapped on demand through CUDA virtual memory management, preserving tensor addresses used by captured CUDA graphs. Logical block release makes pages reusable locally; unreferenced mapped pages form standby capacity. A shared ledger records each engine's committed bytes u_m, online demand report $\widehat{d}_m$, standby bytes, and activity. Claims and releases hold a process-shared lock. An allocation request (a claim) succeeds only if both free physical capacity and the allocation rule permit it; refusal is exposed as unavailable cache capacity to the scheduler.

Let $F = C - \sum_m u_m$, $n_m = \left[\widehat{d}_m - u_m\right]_+$, and guarantee G_m with $\sum_m G_m \le C$. The unmet demand below the guarantee is

$$h_m = \min\{n_m, [G_m - u_m]_+\}. \tag{8}$$

When $H = \sum_m h_m \ge F$, available capacity is apportioned to these deficits. Otherwise, the h_m are covered first and the remainder is distributed over residual demand. The default proportional rule and integer rounding are detailed in Appendix B. Guarantees confer priority on available capacity; they do not revoke live KV. Engines borrow when peers do not need their shares.

4.2 LENDING IDLE CAPACITY WHILE RETAINING VALUABLE PREFIXES

An idle engine must participate in reclamation even when its scheduler has no work. The pool polls pressure on the engine's own thread and releases eligible standby pages. Cached prefixes remain subject to a retention target, or floor, f_m: retaining useful prefixes avoids recomputation, but retaining everything defeats elasticity.

The implementation supports fixed retention shares and optional adaptation using branch waiting. Our primary measurements include the adaptive setting. Full-policy controls isolate its effect: adaptation does not consistently improve on fixed retention across the two platforms (Section 5.4). The allocation contract relies on lending, guarantees, and return; a particular retention signal is not required. Appendix B specifies the optional update rule.

4.3 TARGETED RETURN FROM BUSY BORROWERS

Lending idle pages is not enough when a borrower stays busy and repeatedly recycles them. For queued engines below guarantee, the controller computes the uncovered deficit

$$D = \left[\sum_{m \text{ queued}} \left[\min(\widehat{d}_m, G_m) - u_m\right]_+ - F \right]_+ . \tag{9}$$

It assigns this return target to borrowers in descending excess $[u_m - G_m]_+$, never assigning the same deficit twice. Each borrower selects eligible groups of physical pages that back logical KV blocks, preferring groups with fewer live blocks, and marks their block identifiers unavailable for new allocations. Cached-free blocks are released; existing live users finish normally. Once the last reference leaves and a GPU completion event confirms that prior accesses have finished, the page is unmapped and its budget returned. This stops recirculation without forcibly evicting live requests. Aggregate borrower excess covers D; below-guarantee deficits receive priority on returned bytes (proof and control-loop pseudocode in Appendix B.6).

Proposition 2 (Conditional page-return bound)**.** *Suppose selected page groups remain marked, admit no new references, and all their existing references end within L of marking. If control and marking take at most δ_c, and final retirement takes at most δ_f, the selected bytes return within $\delta_c + L + \delta_f$ of the triggering observation.*

The guarantee applies to selected page groups. Completing a larger deficit also requires enough eligible groups, continued demand, and successful admission of the returned bytes. This gives an implementable return contract based on actual request lifetimes, rather than a prediction that the next stage will need memory.

4.4 MEMORY CORRECTNESS AND NATIVE EXECUTION

The pool preserves model weights, attention values, prompts, and decoding rules. Memory correctness requires three conditions: committed bytes never exceed the per-GPU budget; mappings remain valid until live references and outstanding GPU accesses finish; and cached-prefix entries are removed before their storage is retired. Atomic accounting, reference tracking, and completion events enforce these conditions. Mapping failures roll back pages and accounting. Reassigned capacity is never interpreted as another model's KV content.

The vLLM and SGLang adapters hook native allocation and scheduling interfaces without a maintained source fork. Native prefix caching, chunked prompt processing, asynchronous scheduling, and CUDA graphs remain enabled. vLLM's hybrid KV manager is disabled in both compared configurations, storing sliding-window layers in full. Appendix B specifies deployment scope and correctness checks.

Finite-state checks of the real ledger with a modeled page lifecycle reach 135,552 states without invariant violations. Fixed-batch output checks match the native engine before load, after load, and after idle (Appendix B).

5 EVALUATION

We evaluate complete MoM workflows, isolate the effect of returning borrowed capacity, and test larger catalogues and sustained traffic.

5.1 SETUP AND COMPARISON PROTOCOL

Hardware and models. We use one NVIDIA B200 and two RTX PRO 6000 GPUs. Five resident models span distinct architectures and KV layouts: Gemma-2-9B-it, gpt-oss-20b, Mistral-7B-Instruct-v0.3, Llama-3.1-8B-Instruct, and Granite-3.3-8B-Instruct. The B200 KV budget is 32 GiB; the RTX budget is 16 GiB per card. The RTX placement is Gemma/gpt-oss/Mistral on one GPU and Llama/Granite on the other. We control the KV budget to study the residual-memory resource independently of GPU compute; Appendix B.8 sweeps B200 capacity from 16 to 64 GiB.

Controls. Within each comparison, the baseline and KVLease use identical models, GPU placement, arrival sequences, prompts, call dependencies, output lengths, and total KV capacity. vLLM 0.17.1 runs with prefix caching, CUDA graphs, asynchronous scheduling, prompt processing in chunks of at most 8192 tokens, a 64-sequence concurrency limit, and NVIDIA's Multi-Process Service (MPS) for concurrent engine execution. The baseline uses equal token capacity across engines, which produces unequal byte slices because model KV sizes differ. The elastic KV pool uses those same byte slices as guarantees. SGLang experiments use version 0.5.10.post1.

Workloads. The main collaborative mix combines a single-model service, two-round three-model debate, and three-proposer aggregation on GSM8K, MATH-500, MMLU-Pro, and GPQA (dataset sources in Appendix B.2). A routing-oriented mix combines routed calls, voting, and debate. Recorded call graphs preserve prompts, dependencies, and generation lengths during execution replay. Arrivals use shifted synthetic patterns or Azure conversation/code traces (Stojkovic et al., 2025), with controlled 60-second phases: 80% of arrivals select the dominant class and the remainder select uniformly across classes. Thus the traces supply arrival timing and the experimental design supplies class shifts. Requests arrive for 240 seconds; measurement continues until they finish or time out. Sustained experiments use 30-minute arrival windows.

Metrics. We measure end-to-end user-request latency, including all queueing. Service-level objective (SLO) attainment is the fraction of arrivals completing within 60 seconds unless otherwise specified. One workflow denotes one complete user request, including all its model calls. Throughput includes the time needed to finish outstanding requests after arrivals stop. Unsuccessful arrivals receive the 600-second deadline in aggregate latency accounting and count as SLO misses. The main B200 collaborative comparison uses five seeds at rates 1.5–2.5; the routing and original RTX comparisons use three. Two additional RTX seeds with a different virtual-memory mapping option are reported separately in Appendix B.10. Every comparison pairs identical arrivals across configurations. Percentage changes are computed within each replication and then averaged; they can differ from

Table 1: **End-to-end serving improves across workloads and GPUs.** Means over five replications for B200 collaborative and three for the other workloads; KVLease enables the full policy. Fixed denotes equal-token fixed slices. SLO attainment is the fraction completing within 60 seconds. Brackets give the range of paired p95 changes across replications.

Arrival rate (workflows/s)	Fixed p95 (s)	KVLease p95 (s)	Change [range] (%)	SLO: fixed → ours
B200: collaborative workload				
1.5	73.9	52.3	-28 $[-38, -14]$	$0.82 \rightarrow 0.98$
2.0	129.0	70.1	-45 $[-52, -38]$	$0.50 \rightarrow 0.86$
2.5	173.6	105.6	-39 $[-46, -31]$	$0.41 \rightarrow 0.61$
B200: routing workload				
2.0	92.8	56.2	-39 $[-41, -39]$	$0.81 \rightarrow 0.97$
3.0	139.3	87.2	-37 $[-44, -31]$	$0.79 \rightarrow 0.83$
Two RTX GPUs: collaborative workload				
1.0	193.1	107.1	-45 $[-47, -40]$	$0.49 \rightarrow 0.70$
1.5	337.2	157.6	-53 $[-54, -53]$	$0.41 \rightarrow 0.50$

ratios of displayed mean latencies. Experiments with only one replication are identified explicitly. Appendix B supplies further reproduction details.

5.2 Complete workflows become faster

Table 1 reports complete-workflow gains across GPU platforms and orchestration mixes. At the reported operating points, KVLease lowers p95 by 28–53%. In the collaborative B200 workload at 2 workflows/s, SLO attainment improves by 35.3 percentage points, while throughput rises from 1.33 to 1.66 completed workflows/s. At 1.5 workflows/s on RTX, throughput rises from 0.61 to 0.95. Each reported point improves in every paired seed.

Higher-load rows include backlog; sustained tests follow below. Two further RTX seeds give 40–50% and 53–55% p95 reductions at rates 1 and 1.5, respectively (mapping cohorts and all seeds in Appendix B.10).

5.3 The gain extends beyond pooling

We compare with kvcached (Xing et al., 2025) at matched physical KV capacity, enforced by reserving surplus memory. Prefix caching and compatible attention backends are shared across compared systems. B200 uses different backends from Table 1; RTX retains the primary backends (Appendix B). All arrivals complete and output checks match.

Across five paired seeds, KVLease lowers B200 p95 a further 31% and 24% relative to kvcached at 2 and 2.5 workflows/s. The corresponding 95% Student t intervals are $[-33, -29]$ and $[-30, -18]$ percent. On RTX, the reductions are 17% at 1 workflow/s and 28% at 1.5, with intervals $[-28, -6]$ and $[-32, -24]$. Every higher-load paired seed improves. Figure 3 includes the full tested load range. This comparison measures the complete system; the ablation below isolates policy additions. Idle lending donates cached-free pages only when the engine becomes idle.

5.4 Return connects memory sharing to workflow progress

Figure 4 brings the return mechanism and its deployment consequences together. In the original three-seed B200 mechanism subset at 2 workflows/s, pooled capacity lowers total model-call queueing from 21,462 to 4,968 seconds and p95 by 48%. Execution time rises from 18,390 to 21,465 seconds as batching changes; the queueing reduction dominates.

Why return matters. In Figure 4a, mean below-guarantee episode duration falls from 44.7 to 13.8 seconds on B200 and from 160.3 to 27.6 on RTX. On RTX, p95 falls from 211.8 to 160.4 seconds. On B200, it changes from 64.7 to 68.8 seconds while lender queueing falls from 9.7 to 3.8 seconds per call. Return controls interference rather than minimizing every aggregate metric. The complete cumulative ablation is in Appendix Figure 5.

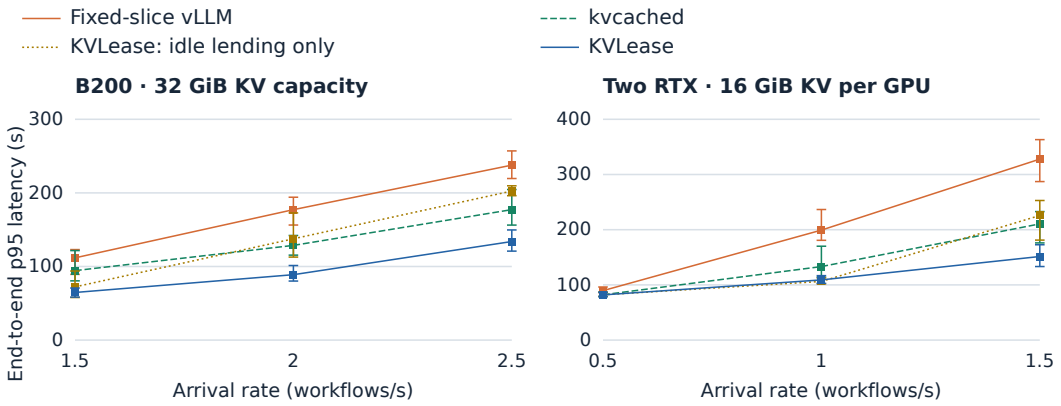

Figure 3: **Pooling is valuable; allocation policy adds a further gain.** Matched-memory comparison with fixed slices, kvcached, a version of KVLease with idle lending alone, and full KVLease. B200 uses kvcached-compatible backends; RTX retains the primary backends. Comparisons are paired within this experiment set. Points show mean p95; whiskers span replications. Fixed slices, kvcached, and full KVLease use five seeds at the two higher rates and three at the lowest; idle lending uses three throughout. Lower is better.

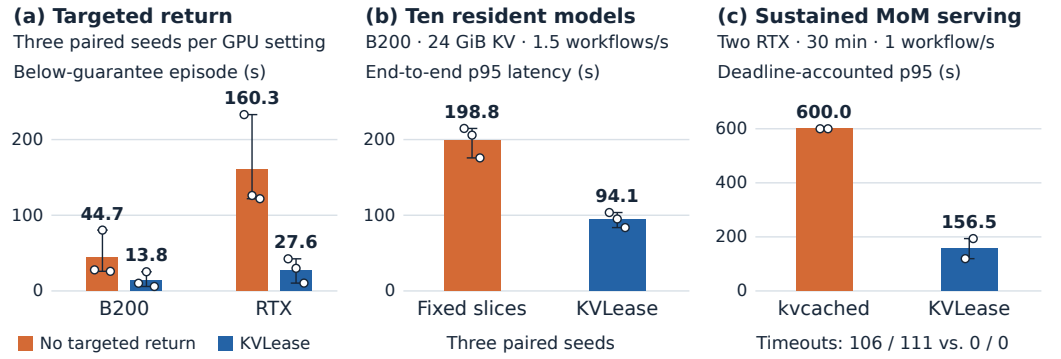

Figure 4: **Recoverable capacity supports MoM progress.** Bars show means, points individual seeds, whiskers their range. (a) Adding targeted return to lending, retention, and guarantees shortens below-guarantee queueing episodes; three seeds at 2 workflows/s on B200 and 1.5 on RTX. (b) Ten resident models, three seeds. (c) Two 30-minute runs, 1,800 arrivals each; timeouts receive the 600-second deadline. Comparisons are paired within each panel.

Beyond idle lending. In the original matched-elastic B200 telemetry subset, KVLease gives gpt-oss 7.2 GiB versus kvcached's 3.0 GiB; preemptions fall from 905 to 332. Lender queueing is 1.3 versus 3.3 seconds per call, compared with 26.1 under idle lending alone. These measurements connect recoverable borrowing to progress by waiting models.

Which policy is necessary? With lending, guarantees, and return fixed, fixed retention gives the lowest p95 in three of four tests and is within 1.3% of the best in the fourth (two seeds on each platform). Periodic quota updates using the same return substrate are within 0–7% of the full policy in the first paired comparisons. Thus the evidence supports the recoverable capacity contract across several policies. Appendix B.12 gives the direct controls.

5.5 MoM DEMAND SHIFTS AND DEPLOYMENT ROBUSTNESS

A profile should guide sharing, not impose a ceiling. A fitted static split captures partition shape and can outperform default guarantees by 12–19% at the highest stable-mix load. Giving KVLease the same profile yields 83.6 versus 82.4 seconds. On a shifted mix, profiled guarantees yield 44.3

versus 102.3 for the static profile (one paired diagnostic per setting; Appendix B.13). Borrowing accommodates demand beyond the profiled share.

Larger resident catalogues. Figure 4b shows 52–54% lower p95 across three ten-model seeds, with every arrival completed. Weights and engine overhead leave 33 GiB; the pool uses 24 GiB. Eight-model and high-capacity controls appear in Appendix B.14.

Sustained operation. Figure 4c compares two 30-minute RTX runs. KVLease completes all 1,800 arrivals in each; kvcached times out on 106 and 111. SLO attainment rises from 0.31 to 0.64 and from 0.31 to 0.44. All arrivals remain in the accounting (Appendix B.16).

Engines and allocation choices. Installing the pool inside SGLang lowers p95 by 28–39% across two seeds (Appendix B.17). Separately, vLLM-based KVLease gives 33–38% mean reductions versus vanilla SGLang across three seeds. Two-seed comparisons give 51–58% reductions versus equal GPU-memory fractions and 42–48% versus native CPU KV offload (Appendix B.15).

Workload diversity. Aggregation-only workflows give 52–56% reductions across three seeds; MMLU-Pro/GPQA gives 41–56% across two; ARC-Challenge/WinoGrande gives 56–61% at the highest rate across two. Appendix B.11 reports full load curves. Memory-budget and speculative-decoding comparisons further characterize deployment scope (Appendices B.8 and B.19).

6 Related work and positioning

Elastic memory and multi-model serving. vAttention separates physical KV allocation from stable virtual addresses (Prabhu et al., 2025). MuxServe periodically adapts per-model KV token-block quotas alongside prefill/decode scheduling (Duan et al., 2024). Prism combines its kvcached balloon driver with model placement and GPU-local priority admission (Xing et al., 2025; Yu et al., 2026). Our kvcached experiment compares elastic implementations, not Prism's complete scheduler. S-LoRA unifies adapter and KV paging around a shared base model; CrossPool manages KV and weight capacity for cold MoE models (Sheng et al., 2024; Ye et al., 2026). KVLease keeps heterogeneous weights resident and coordinates capacity across separate engine schedulers. Reservations, borrowing, and reclamation have classical precedents in virtual-machine memory management (Waldspurger, 2002); our contribution is a recoverable KV-capacity contract across independent engines, its workflow-demand analysis, and its complete-workflow evaluation.

Workflow and agent serving. Parrot and Agentix expose application dependencies and schedule LLM programs (Lin et al., 2024; Luo et al., 2026). Kairos combines workflow priorities with memory-aware dispatch (Chen et al., 2025). KVFlow, InferCept, and TokenCake manage cache reuse or retention across agent stages and interruptions (Pan et al., 2025; Abhyankar et al., 2024; Bian et al., 2025). Maestro combines stage-cost prediction, elastic provisioning, and weight caching with cross-cluster routing; SAGA schedules workflows with session affinity and KV reuse (Wang et al., 2026; Guo et al., 2026). MOSAIC optimizes MoA placement and concurrency and can bypass aggregation on consensus (Mitra et al., 2026). These works establish workflow-aware serving as a broad design direction. KVLease focuses on the physical-byte allocation contract between independently running, heterogeneous engines while holding the orchestration and its model computations fixed.

7 Conclusion

MoM workflows need capacity to reach the next stage even while earlier-stage engines remain busy. KVLease coordinates lending and return by stopping reuse of selected borrowed pages; capacity returns after references and GPU accesses finish. Its MoM analysis and replicated experiments connect this contract to improved complete-workflow performance across engines, model catalogues, and sustained traffic.

AI USE STATEMENT

AI tools supported both the research process and preparation of this manuscript. We used them to discover related literature, draft sections, and improve the clarity, organization, and wording of the paper. Research assistance covered exploring system designs and hypotheses, formulating mathematical claims and developing proofs for them, implementing and debugging code, and analyzing and interpreting recorded experiments. We also used AI tools to generate and refine conceptual figures; empirical plots were produced from measured data. Validation involved testing code, comparing reported results with experiment records, checking references against original sources, and reviewing mathematical assumptions and arguments. The authors retain responsibility for the final manuscript and all research decisions, technical claims, code, figures, and results.

REPRODUCIBILITY STATEMENT

Appendices A and B provide the analytical assumptions and proofs, experimental settings, paired replay protocol, individual replication results, memory-budget comparisons, and statistical definitions used in this paper. All reported measurements come from completed experiments.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 81–95. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/abhyankar24a.html.

Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. Tokencake: A kv-cache-centric serving framework for llm-based multi-agent applications. *arXiv preprint arXiv:2510.18586*, 2025.

Jinyuan Chen, Jiuchen Shi, Quan Chen, and Minyi Guo. Kairos: Low-latency multi-agent serving with shared llms and excessive loads in the public cloud. *arXiv preprint arXiv:2508.06948*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. MuxServe: Flexible spatial-temporal multiplexing for multiple LLM serving. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11905–11917. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/duan24a.html.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. Saga: Workflow-atomic scheduling for ai agent inference on gpu clusters. *arXiv preprint arXiv:2605.00528*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

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 ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Günter Last and Mathew Penrose. *Lectures on the Poisson Process*. Cambridge University Press, October 2017. ISBN 9781107458437. doi: 10.1017/9781316104477. URL https://doi.org/10.1017/9781316104477.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 2443–2459, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/luo.

Saptarshi Mitra, Yifan Zhang, Rachid Karami, Phyo Pyae Moe Aung, Nazmul Takbir, Sreetama Sarkar, Souvik Kundu, and Sitao Huang. Mosaic: Efficient mixture-of-agent scheduling via adaptive aggregation and inference concurrency. *arXiv preprint arXiv:2606.03014*, 2026.

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 D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 126246–126265. Curran Associates, Inc., 2025. doi: 10.52202/085713-4208. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/b7971d31a7d5eb0f1eed2f8f6f368195-Paper-Conference.pdf.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, pp. 1133–1150, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400706981. doi: 10.1145/3669940.3707256. URL https://doi.org/10.1145/3669940.3707256.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, April 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6399. URL http://dx.doi.org/10.1609/aaai.v34i05.6399.

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. Slora: Scalable serving of thousands of lora adapters. In P. Gibbons, G. Pekhimenko,

and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 296–311, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/906419cd502575b617cc489a1a696a67-Paper-Conference.pdf.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1348–1362. IEEE, 2025.

Carl A. Waldspurger. Memory resource management in VMware ESX server. In *5th Symposium on Operating Systems Design and Implementation (OSDI 02)*, Boston, MA, December 2002. USENIX Association. URL https://www.usenix.org/conference/osdi-02/memory-resource-management-vmware-esx-server.

Jinghao Wang, Xiao Zhou, Xiaoyang Sun, Yihui Zhang, Yilong Li, Tianyu Wo, Xu Wang, Chunming Hu, and Renyu Yang. Maestro: Workload-aware cross-cluster scheduling for llm-based multi-agent systems. In *46th IEEE International Conference on Distributed Computing Systems, ICDCS 2026, Seoul, Republic of Korea, June 22-25, 2026*, pp. 250–260. IEEE, 2026. doi: 10.1109/2575-8411.2026.00031. URL https://doi.org/10.1109/2575-8411.2026.00031.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 33944–33963, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5434be94e82c54327bb9dcaf7fca52b6-Paper-Conference.pdf.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Jiarong Xing, Yifan Qiao, Simon Mo, Xingqi Cui, Gur-Eyal Sela, Yang Zhou, Joseph Gonzalez, and Ion Stoica. Towards efficient and practical gpu multitasking in the era of llm. *arXiv preprint arXiv:2508.08448*, 2025.

Zhuoren Ye, Tianyu Wo, Dinghao Xue, Mingming Zhang, Yuchen Teng, Chunming Hu, and Renyu Yang. Crosspool: Efficient multi-llm serving for cold moe models through kv-cache and weight disaggregation. *arXiv preprint arXiv:2606.24506*, 2026.

Shan Yu, Yifan Qiao, Mingyuan Ma, Yangmin Li, Shuo Yang, Xinyuan Tong, Yang Wang, Zhiqiang Xie, Yuwei An, Shiyi Cao, Ke Bao, Deepak Vij, Xiaoning Ding, Yichen Wang, Qingda Lu, Zhong Wang, Gao Gao, Harry Xu, Junyi Shu, Jiarong Xing, and Ying Sheng. Prism: Cost-Efficient Multi-LLM serving via GPU memory ballooning. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 75–93, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL https://www.usenix.org/conference/osdi26/presentation/yu-shan.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue 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 *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

A PROOFS AND ANALYTICAL MODEL

A.1 MULTIPLEXING AND CO-ACTIVITY

Proposition 3 (Multiplexing and co-activity)**.** *$1 \le \Gamma \le M$, with both bounds attainable. If phase p activates models A_p with $d_m(p) = w_m \mathbf{1}\{m \in A_p\}$, then*

$$\Gamma = \frac{\sum_{m \in \cup_p A_p} w_m}{\max_p \sum_{m \in A_p} w_m}. \tag{10}$$

If all M models appear with equal positive weights, $\Gamma = M/\omega$, where $\omega = \max_p |A_p|$.

Proof of Proposition 3. For each phase p, $\sum_m d_m(p) \le \sum_m \max_q d_m(q)$, giving $\Gamma \ge 1$. Each individual peak is at most C^*_{pool}, so summing gives $\Gamma \le M$. Simultaneous equal peaks attain one. Rotating a single equal-sized hotspot across all models attains M. Under the co-activity model, each model appearing in any phase has peak w_m, and the largest simultaneous footprint is $\max_p \sum_{m \in A_p} w_m$. Substitution gives the weighted formula and its equal-weight specialization. □

The finite phase set is nonempty and the demand trace is nonzero. Models with zero demand can be omitted from the active catalogue; including them preserves the upper bound but not necessarily its attainability for that trace.

A.2 MEMORY-FEASIBILITY DECOMPOSITION

For prescribed $d_m(p) = \Lambda v_m(p)$, a fixed split accommodates every demand if and only if $\Lambda v_m(p) \le S_m$ for all (m, p). This gives Λ_S. To accommodate load Λ, any fixed split needs at least $\Lambda \sum_m \max_p v_m(p)$ bytes. Allocating $S_m = C \max_p v_m(p) / \sum_j \max_p v_j(p)$ attains that lower bound and proves the formula for Λ_{best}. A pool accommodates every phase if and only if $\Lambda \max_p \sum_m v_m(p) \le C$. Dividing these expressions proves Equation 6. This result concerns a prescribed demand trajectory, not a scheduling algorithm's stability region.

One way to estimate a phase footprint is $d(p) = U^\top \lambda(p)$, where $\lambda_c(p)$ is the class arrival rate and $U_{c,m}$ is the mean byte-seconds contributed to model m by one class-c request. U includes context lengths and residence times as well as the call graph. Holding U fixed gives the fluid sizing model. Under load, queueing and batching can change residence times; direct measurements evaluate that regime.

A.3 RECOVERABLE OVERFLOW

Proof of Equation 7. The identity $[x]_+ - [-x]_+ = x$ gives

$$O - I = \sum_m (d_m - S_m) = \sum_m d_m - C.$$

For nonnegative O, I, $O - [O - I]_+ = \min\{O, I\}$. Substitution proves the claim pointwise; integration proves the byte-second result. □

The distinction from Γ is strict. For a constant demand $(2, 0)$ and slices $(1, 1)$, $\Gamma = 1$ but $R = 1$: correcting shape helps even without temporal multiplexing. Conversely, a rotating trace $(1, 0), (0, 1)$ has $\Gamma = 2$ but $R = 0$ under slices $(1, 1)$: overprovisioning removes pressure despite a sizing opportunity.

A.4 STATIC SEPARATION AND RETURN

Consider fluid phases of duration P with constant demand satisfying $\sum_m d_m \le C$. Define useful held-capacity service as $\int \sum_m \min\{d_m, u_m(t)\}\, dt$, for allocation $u_m(t)$.

Proposition 4 (Static separation and return cost)**.** *Every fixed split has a phase in which it serves at most $1/M$ of an instantaneous pool. The equal split serves at least $1/M$ on every such phase, making M the optimal worst-case static ratio. A borrower that never returns capacity can leave another engine with zero service for arbitrarily long. An allocator that reaches a demand-satisfying allocation within $\tau < P$ of each phase change achieves at least $1 - \tau/P$ of ideal byte-time service in that phase.*

Proof of Proposition 4. Let m^* be a model with minimum fixed slice. Then $S_{m^*} \le C/M$. A phase with demand C only on m^* yields fixed-split service $PS_{m^*} \le PC/M$, versus PC for an instantaneous pool. For the equal split, if every $d_m \le C/M$, all demand is accommodated. Otherwise $\sum_m \min(d_m, C/M) \ge C/M \ge \sum_m d_m/M$. Thus the equal split attains the matching worst-case bound.

For non-returning borrowing, let one engine first claim all C bytes and keep reusing them while its demand persists. A second engine cannot acquire any capacity, for an arbitrarily long horizon, whereas a positive fixed slice would give it positive byte-time service. This is an isolation counterexample; it need not reduce the borrower's own utilization.

Finally, after a recovery time of at most τ, all feasible phase demand is accommodated for at least $P - \tau$. Service is nonnegative before recovery. Total service is therefore at least $(P - \tau) \sum_m d_m$, compared with $P \sum_m d_m$ for the ideal pool. For a zero-demand phase both quantities vanish; otherwise division gives $1 - \tau/P$. □

This last result is conditional on full demand recovery. Per-model guarantees, by themselves, do not imply that condition. Floors, finite allocation quanta, device-completion delays, and live references determine actual recovery, which the experiments measure.

A.5 Page-return contract

Proof of Proposition 2. By time δ_c, the selected groups are marked. Since no new references are admitted, every reference belongs to the finite initial set. All end within L after marking. The last-reference event therefore occurs by $\delta_c + L$. Completion of the final fence and retirement within δ_f then releases each selected group's bytes by $\delta_c + L + \delta_f$. □

Reference lifetime includes downstream GPU use, not just CPU-side release. A complete isolation guarantee additionally requires enough reclaimable groups, continued servicing of the control loop, and progress of waiting claims. The implementation rounds drain targets to whole groups and does not forcibly unmap live data.

A.6 Workflow covariance and statistical sizing

Proof of Proposition 1. For a marked Poisson process, Campbell's first-moment formula gives $\mathbb{E}[\sum_r f(t_r, \xi_r)] = \int \lambda(s) \mathbb{E}[f(s, \xi)]\, ds$. Its covariance formula is $\mathrm{Cov}(\sum_r f(t_r, \xi_r), \sum_r g(t_r, \xi_r)) = \int \lambda(s) \mathbb{E}[f(s, \xi) g(s, \xi)]\, ds$ (Last & Penrose, 2017). Apply these formulas with $f = h_m(t - s)$ and $g = h_n(t - s)$, and sum over independent classes. Total covariance over T gives $\mathcal{S} + \mathcal{R}$. Since footprints are nonnegative, $Q_{mn} \ge 0$. If two models never hold memory simultaneously within any request, $Q_{mn} = 0$ almost everywhere. Each covariance matrix is positive semidefinite, as are its average and the covariance of the mean path. □

For rectangular call footprints, call i holds b_i bytes on model $m(i)$ during $I_i = [s_i, e_i)$. Then

$$\int h_m(u) h_n(u)\, du = \sum_{i:m(i)=m} \sum_{j:m(j)=n} b_i b_j |I_i \cap I_j|.$$

This makes stage overlap explicit. Parallel debate branches starting together overlap for the shorter branch's duration. A proposer and its subsequent aggregator have zero overlap if their KV lifetimes end at their call boundaries. Persistent cached prefixes require additional footprints and do not satisfy that premise automatically.

Proof of Equation 4. Stationary Poisson superposition gives $\mu = \Lambda\bar{u}$ and $\Sigma = \Lambda W$, where $\bar{u} = \sum_c \pi_c \int H^{(c)}$. Divide the numerator and denominator of Γ_k by $\sqrt{\Lambda}$ and take the limit. Since $W_{mn} \geq 0$, $\mathbf{1}^\top W \mathbf{1} \geq \operatorname{tr} W$, with strict inequality exactly when an off-diagonal entry is positive. The covariance triangle inequality and Cauchy–Schwarz also give $1 \leq \Gamma_0 \leq \sqrt{M_{\text{active}}}$ for a nonzero process. □

For $k = \sqrt{(1-\varepsilon)/\varepsilon}$, Cantelli's inequality makes each static slice $\mu_m + k\sigma_m$ sufficient for its own overflow probability to be at most ε; the pooled denominator controls aggregate overflow at ε. This is a comparison of marginal sizing rules, not equal system-wide overflow guarantees: the static union bound is $M\varepsilon$. Under a Gaussian approximation, one instead uses $k = \Phi^{-1}(1-\varepsilon)$.

Empirical moment check. We reconstruct class templates from recorded call start/end times and prompt/output token counts, linearly interpolating KV growth over each call. Class-rate paths use the configured Poisson rate or a 10-second histogram for trace-driven arrivals. The convolution is evaluated on a one-second grid and compared with independently collected occupancy telemetry from the same executions, at the lightest tested rate. Across nine experiment replications, predicted versus measured cross-model correlations have Pearson coefficients of 0.97–1.00; mean absolute correlation errors are 0.04–0.16 on B200 and 0.05–0.13 on the three-model RTX GPU. The other RTX GPU supplies only one pair, so no across-pair correlation coefficient is reported.

At Gaussian $\varepsilon = 0.01$, the template-based sizing ratios differ from measured-moment ratios by less than 0.02 in all 12 per-GPU cases. B200 template ratios are 1.07–1.13, measured-moment ratios 1.07–1.12, and setting the measured off-diagonal covariance to zero gives 1.38–1.45. This checks the workflow covariance model using observed residence times; it is not a held-out forecast of latency or a validated 1% overflow guarantee. The analytical Poisson assumption applies exactly to the synthetic arrivals and approximately to the trace-driven moment comparison.

Transfer across seeds. To test more than reconstruction, templates are taken from another seed's lightest-load execution and applied to the target arrival rate. Same-seed pairs are excluded. The 36 source–target–rate comparisons span six target jobs, eighteen job–rate targets, and six template recordings on B200; repeated uses of a target are not independent replications. Pearson correlation across model-pair correlations ranges from 0.956 to 0.999 at the lightest rate and 0.925 to 0.996 at higher rates. Median absolute correlation error is 0.093, with a maximum of 0.257. The test evaluates demand-correlation transfer, not a forecast of queueing latency: template residence times remain fixed while actual service times can change under load.

B Experimental protocol and reproducibility

B.1 Configurations and scope

The B200 equal-token split is approximately 13.4 GiB for Gemma, 1.9 for gpt-oss, 6.4 for Granite, and 5.1 each for Llama and Mistral (rounded). In each paired comparison, the exact byte budget is identical. Model weights remain resident. NVIDIA Multi-Process Service (MPS) is enabled for all engines; tensor and pipeline parallelism are one. Maximum sequence length and batched-token budget are 8192; maximum concurrent sequences per engine is 64. vLLM uses full and piecewise CUDA graph capture (`FULL_AND_PIECEWISE`), asynchronous scheduling, prefix caching, and tuned mixture-of-experts (MoE) kernels for gpt-oss. Its hybrid KV manager is disabled in both configurations, storing sliding-window layers in full. The two RTX cards each have their own budget; capacity does not migrate between GPUs.

The primary measured configuration enables idle lending, blocking-priced floors, guarantees, and drain. Fixed-retention controls preserve the startup cache shares and remove only their adaptation. The separate controls do not retroactively change the configuration of the primary results. Floors update every 10 seconds by at most 2% of local pool capacity, with per-engine minima of half the initial floor. Drain is polled every 0.2 seconds; marks may be cleared after 10 seconds without a deficit. The page-return proposition assumes marks remain in force for the episode in question. For a parallel call group, each unfinished branch accumulates branch-seconds after the first branch finishes. The controller sums these times by model, scores each by accumulated time divided by floor bytes, and transfers at most $0.02C$ from a lowest-scoring donor to a highest-scoring recipient,

retaining at least half each initial floor. A zero floor has infinite score; ties use engine index, and an all-zero window makes no transfer. The occupancy control substitutes call-seconds in flight. These are retention heuristics, evaluated separately from the return contract.

For free capacity F, unmet demand n_m, and below-guarantee demand h_m, the proportional rule assigns integer headroom $\lfloor F h_m / \sum_j h_j \rfloor$ when $\sum_j h_j \geq F > 0$. Otherwise, below-guarantee needs are covered and residual capacity is shared over residual demand. When all needs fit, an engine may also use slack not needed by peers. The actual claim rechecks headroom under the ledger lock, so concurrent admission cannot spend the same bytes twice. Demands do not authorize live-page revocation.

B.2 Dataset sources

The primary workloads use GSM8K mathematical word problems (Cobbe et al., 2021), the MATH-500 subset of MATH competition problems (Hendrycks et al., 2021), MMLU-Pro knowledge and reasoning questions (Wang et al., 2024), and GPQA graduate-level questions (Rein et al., 2024). Additional dataset comparisons use ARC-Challenge science questions (Clark et al., 2018) and WinoGrande commonsense questions (Sakaguchi et al., 2020). These datasets supply inputs for the model-call workflows; the reported objective is serving efficiency.

B.3 Recorded execution and outcome accounting

Recording fixes the model calls, prompts, dependencies, and generated lengths used in replay. Replay executes those calls through the engine while preserving the graph, so every compared allocation policy processes the same work. It is a serving-efficiency comparison, not a new answer-quality benchmark. In two validation comparisons, live execution and replay have p95 latencies of 38.8 versus 38.7 seconds and 39.3 versus 44.5 seconds, respectively, with matching output lengths. Dataset examples vary task content; they do not establish task-quality improvements.

Requests are measured from user-request arrival to completion, including waiting between calls. Failed and timed-out requests remain in the denominator for SLO and throughput; aggregate latency charges unsuccessful arrivals the 600-second deadline. Throughput divides completed user requests by the full measured duration including completion of outstanding requests. Independent-call diagnostics remove dependencies while preserving work volume; their completion objective differs from complete-workflow execution and is labeled accordingly.

Table 1 uses five seeds for the B200 collaborative workload and three for the routing and original RTX cohorts. The appendix includes the two additional RTX seeds separately. Additional studies report their own replication counts and all tested rates. The reported benefits concern recoverable memory pressure; light-load and high-capacity controls locate this operating regime.

B.4 Elastic baseline accounting

We evaluate kvcached at commit `944ed53`, using the same model pool, prompts, arrival sequence, and cache budget. A separate process reserves surplus device memory, accounting for kvcached's own device reserve and already mapped pages, so its available KV capacity matches the other configurations. For all four B200 configurations, attention tensors use token–head–dimension ordering, as required by kvcached. The backend is `FLASH_ATTN`, except that gpt-oss uses `TRITON_ATTN`. That backend choice is held fixed within this experiment and differs from the primary performance experiments. The elastic comparison therefore stands on its own paired evidence.

Variation across replications is summarized with paired percentage changes and a two-sided Student t interval using $n - 1$ degrees of freedom for n paired seeds. Figure whiskers instead show the minimum and maximum across replications. At the lowest B200 kvcached rate, the paired mean improvement is 29% with a wide interval $[-71, +13]$ percent; at the lightest RTX rate, the paired change is -0.24% $[-0.68, +0.19]$. The abstract's elastic-baseline claim refers to the higher-load points with intervals reported in the main text.

B.5 Ablation and safety checks

Cumulative ablation uses fixed slices, idle lending, added retention floors, added guarantees, and added targeted drain. Each platform uses three replications, with identical workloads for every configuration within a replication. The figure reports all five configurations, allowing the performance and interference tradeoff of each addition to be read directly.

Finite-state verification couples the actual shared-budget code with a modeled page lifecycle and explores claims, rollback, release, standby reuse, reclamation, rescue, retirement, changing demand, and crash accounting. The largest reported configuration has three engines and four abstract pages, reaching 135,552 states with zero violations. The exhaustive result concerns these finite lifecycle configurations; GPU execution is checked separately through output-equivalence tests. Output-equivalence checks with fixed batches compare outputs before load, after load, and after idle; reported comparison runs pass their applicable 15–45 cases. Native values and algorithms remain unchanged, while concurrent batch scheduling can differ.

Table 2: B200 matched elastic comparison. kvcached → KVLease; seed labels abbreviate 202609xx. Five seeds at the two higher rates, three at the lowest. The idle-lending ablation in Figure 3 retains three seeds.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1.5	19	121.6 → 64.9	0.49 → 0.84	1.02 → 1.22	−46.6
1.5	24	81.0 → 70.7	0.73 → 0.84	1.19 → 1.30	−12.7
1.5	25	80.5 → 58.4	0.78 → 0.98	1.19 → 1.31	−27.4
2	19	127.1 → 87.2	0.59 → 0.69	1.33 → 1.54	−31.4
2	24	142.1 → 101.3	0.54 → 0.58	1.33 → 1.56	−28.7
2	25	115.4 → 80.2	0.62 → 0.79	1.38 → 1.61	−30.5
2	26	138.3 → 92.7	0.45 → 0.66	1.32 → 1.55	−33.0
2	27	120.3 → 82.9	0.58 → 0.74	1.40 → 1.63	−31.1
2.5	19	182.5 → 137.3	0.42 → 0.42	1.45 → 1.71	−24.8
2.5	24	179.2 → 126.3	0.44 → 0.45	1.50 → 1.83	−29.5
2.5	25	163.7 → 135.4	0.43 → 0.45	1.48 → 1.70	−17.3
2.5	26	204.6 → 149.7	0.38 → 0.45	1.42 → 1.71	−26.9
2.5	27	156.3 → 120.9	0.48 → 0.54	1.55 → 1.80	−22.6

Table 3: RTX matched elastic comparison. kvcached → KVLease; seed labels abbreviate 202609xx. Five seeds at the two higher rates, three at the lowest. The idle-lending ablation in Figure 3 retains three seeds.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
0.5	19	80.4 → 80.3	0.72 → 0.72	0.40 → 0.41	−0.0
0.5	24	79.1 → 78.8	0.85 → 0.85	0.41 → 0.41	−0.4
0.5	25	86.7 → 86.5	0.75 → 0.75	0.42 → 0.42	−0.3
1	19	123.2 → 113.0	0.61 → 0.71	0.70 → 0.76	−8.3
1	24	127.8 → 103.6	0.62 → 0.70	0.68 → 0.76	−19.0
1	25	114.0 → 102.8	0.67 → 0.71	0.72 → 0.77	−9.8
1	26	170.1 → 116.8	0.57 → 0.67	0.60 → 0.74	−31.4
1	27	130.8 → 109.1	0.61 → 0.71	0.68 → 0.74	−16.6
1.5	19	232.8 → 172.7	0.47 → 0.44	0.77 → 0.93	−25.8
1.5	24	224.1 → 162.7	0.49 → 0.49	0.78 → 0.89	−27.4
1.5	25	176.1 → 133.5	0.59 → 0.62	0.89 → 1.03	−24.2
1.5	26	232.5 → 155.0	0.43 → 0.45	0.76 → 0.96	−33.4
1.5	27	187.0 → 133.3	0.56 → 0.61	0.85 → 1.03	−28.7

B.6 Control loop and deficit coverage

The ledger's online demand $\widehat{d}_m$ estimates bytes the engine can use under its concurrency limit. In vLLM, it includes running requests and the first waiting requests that fit the remaining sequence slots. Each request contributes its current tokens plus its remaining declared output allowance, capped by the maximum sequence length and rounded up to logical blocks. The byte total adds two physical

pages per KV buffer for mapping alignment. This uses current scheduler state and declared limits, not future arrivals or realized output lengths. It differs from both the analytical footprint d_m and the telemetry estimator below. Claims and releases update the ledger under the shared lock. The controller has three events:

1. **Claim:** read current committed bytes and needs; compute free capacity F and below-guarantee needs h_m; allocate headroom to h_m first and then residual needs; atomically charge an allowed claim before mapping its pages. Roll back a failed mapping.
2. **Floor update (10 s):** collect completed and ongoing branch-wait charges; rank b_m/f_m; transfer at most $0.02C$ from a low-scoring floor to the highest-scoring floor, respecting the minima. Publish targets to local engines.
3. **Return poll (0.2 s):** compute D; assign targets to borrowers by excess above guarantee; mark eligible page groups to prevent reuse; retire a group after its last reference and device completion; release its charge for waiting claims.

Page eligibility and prefix reuse. The vLLM adapter excludes pinned groups and selects groups with the fewest live blocks. It masks every logical block touching a selected page against new allocation and removes its prefix-cache lookup entry, including entries for still-live blocks; existing references retain their contents. Only physical pages untouched by unmarked blocks count toward return. Targets are rounded down to whole page groups. A claim requires its full mapping quantum to fit the current headroom; byte priority does not imply immediate admission.

For $B = \sum_m [u_m - G_m]_+$ and $A = \sum_m [G_m - u_m]_+$, accounting gives

$$B - A = \sum_m u_m - \sum_m G_m = C - F - \sum_m G_m \geq -F.$$

The queued below-guarantee deficit is at most A, hence $B \geq D$. There are enough excess committed bytes in aggregate to cover the deficit. Physical eligibility and live references determine when those bytes can be returned. When $\sum_m h_m \geq F$, the claim rule assigns no residual headroom to above-guarantee borrowing, protecting the available bytes for the returning engines.

B.7 Pressure measurements and demand estimator

At one-second telemetry samples, a blocked engine has queued calls and insufficient local KV headroom. Idle capacity in its peers is measured relative to the configured slices. To estimate offered pressure, let $x_m(t)$ be the referenced-KV occupancy inferred from reported utilization, $q_m(t)$ the waiting-call count, and $\bar{p}_m$ the mean prompt-KV bytes per call, computed from token counts and that model's KV bytes per token. We use

$$\widetilde{d}_m(t) = x_m(t) + q_m(t)\bar{p}_m.$$

This estimate includes queued prompts but omits their future decode growth. It is a pressure proxy, not an observation of the unconstrained counterfactual: scheduling changes call residence times and downstream release times. For each GPU, substitute $\widetilde{d}$ into Equation 7; sum recoverable bytes across GPUs and normalize the time average by total KV capacity. All inputs come from the fixed-slice execution. Across 30 paired workload/rate/replication comparisons, this normalized proxy has Spearman correlation 0.77 with p95 improvement, versus 0.10–0.13 for peak-sizing ratios. These comparisons reuse workload settings; the association is descriptive.

B.8 Memory-budget sweep

The KV budget is an explicit experimental control, not the physical HBM size. It represents residual capacity after resident weights, engine workspaces, and other allocations. The following sweep varies this resource while preserving the five-model catalogue, workload, and generation settings. Each budget uses one paired replication; it measures sensitivity rather than a replicated scaling result. At 2 workflows/s, KVLease with 32 GiB reaches 61.9 seconds; fixed slices require 64 GiB to approach that latency (66.4 seconds). At 64 GiB and 1.5 workflows/s, both are near 53 seconds. The measured benefit follows memory pressure and persists at higher load even as the budget doubles.

Table 4: B200 memory-budget sweep. End-to-end p95 seconds, equal-token fixed slices → full KVLease. All arrivals complete.

KV budget (GiB)	1.5 workflows/s	2.0 workflows/s	2.5 workflows/s
16	186.5 → 82.8	242.1 → 135.7	355.0 → 192.6
24	120.8 → 56.3	167.3 → 77.0	236.6 → 147.8
32	89.4 → 55.1	129.6 → 61.9	181.6 → 98.1
48	59.2 → 53.7	81.7 → 58.0	124.2 → 82.8
64	52.9 → 53.4	66.4 → 57.5	96.7 → 75.4

B.9 Instrumented allocator overhead

Profiles time allocation, availability checks, block release, and page-group selection on the engine scheduler thread. Across the profiled non-speculative configurations, these routines consume 0.14–1.4 ms per engine step and 4–37% of one CPU core summed over five engines. Blocking retirement fences are included in allocation time and total at most 1.85 seconds across engines per approximately 300-second window. Background reclaimer CPU time is outside this instrumentation. These costs are already included in the end-to-end measurements; the profile is not a standalone map/unmap or contention microbenchmark. MPS uses its default compute-sharing configuration in all comparisons. Increased batch size can change compute interference even with identical MPS settings.

B.10 Primary per-replication results

Table 1 and the cumulative ablation use separate paired executions. All primary KVLease configurations enable lending, retention, guarantees, and return. The original B200 three-seed mechanism subset is retained for telemetry analysis; the latency table includes two further seeds at rates 1.5–2.5. Seed labels below abbreviate 20260919, 20260924, etc. Rates denote arriving workflows/s; arrows denote baseline → KVLease. Completed/s includes backlog completion. Every arrival completes in these primary comparisons.

The original RTX cohort maps unused virtual addresses to a shared guard page to protect padded kernel reads; live KV always has distinct physical backing. Two additional seeds use ordinary mapping. We report the cohorts separately, since changing this implementation option can affect overhead even though the allocation policy is unchanged.

Table 5: B200 collaborative workload: five seeds at the main rates; three at light load.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	43.8 → 43.3	0.99 → 0.99	0.86 → 0.87	−1.1
1	24	45.7 → 43.6	1.00 → 1.00	0.96 → 0.96	−4.4
1	25	48.2 → 43.5	1.00 → 1.00	0.88 → 0.93	−9.8
1.5	19	89.4 → 55.1	0.71 → 0.98	1.09 → 1.27	−38.4
1.5	24	80.1 → 56.1	0.70 → 0.97	1.18 → 1.34	−30.0
1.5	25	57.6 → 49.7	0.96 → 0.99	1.25 → 1.36	−13.7
1.5	26	75.4 → 49.7	0.79 → 0.98	1.22 → 1.36	−34.0
1.5	27	67.1 → 51.0	0.92 → 0.97	1.30 → 1.38	−24.0
2	19	129.6 → 61.9	0.58 → 0.93	1.30 → 1.64	−52.3
2	24	143.6 → 70.4	0.48 → 0.88	1.31 → 1.67	−51.0
2	25	114.0 → 67.2	0.58 → 0.92	1.37 → 1.68	−41.0
2	26	133.2 → 83.1	0.41 → 0.67	1.33 → 1.60	−37.6
2	27	124.4 → 67.8	0.47 → 0.88	1.36 → 1.71	−45.5
2.5	19	181.6 → 98.1	0.38 → 0.59	1.43 → 1.89	−46.0
2.5	24	167.2 → 98.7	0.42 → 0.63	1.53 → 2.00	−41.0
2.5	25	172.9 → 119.1	0.41 → 0.49	1.44 → 1.80	−31.1
2.5	26	190.4 → 128.0	0.38 → 0.55	1.45 → 1.81	−32.8
2.5	27	156.0 → 84.3	0.45 → 0.77	1.55 → 1.99	−45.9

Table 6: B200 routing workload: three paired seeds.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1.5	19	$38.0 \rightarrow 33.4$	$1.00 \rightarrow 1.00$	$1.32 \rightarrow 1.36$	-11.9
1.5	24	$71.4 \rightarrow 50.5$	$0.86 \rightarrow 1.00$	$1.17 \rightarrow 1.30$	-29.3
1.5	25	$63.0 \rightarrow 42.8$	$0.94 \rightarrow 1.00$	$1.21 \rightarrow 1.31$	-32.0
2	19	$96.5 \rightarrow 57.1$	$0.81 \rightarrow 0.97$	$1.44 \rightarrow 1.69$	-40.8
2	24	$81.1 \rightarrow 49.3$	$0.83 \rightarrow 1.00$	$1.52 \rightarrow 1.71$	-39.1
2	25	$100.9 \rightarrow 62.0$	$0.79 \rightarrow 0.94$	$1.45 \rightarrow 1.68$	-38.5
3	19	$138.0 \rightarrow 94.7$	$0.78 \rightarrow 0.80$	$1.89 \rightarrow 2.23$	-31.4
3	24	$116.6 \rightarrow 75.6$	$0.83 \rightarrow 0.88$	$2.03 \rightarrow 2.39$	-35.1
3	25	$163.4 \rightarrow 91.3$	$0.75 \rightarrow 0.82$	$1.75 \rightarrow 2.21$	-44.1

Table 7: RTX collaborative workload: original three-seed cohort with guard-page mapping.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
0.5	19	$85.9 \rightarrow 80.6$	$0.69 \rightarrow 0.72$	$0.38 \rightarrow 0.40$	-6.1
0.5	24	$85.4 \rightarrow 79.1$	$0.78 \rightarrow 0.85$	$0.40 \rightarrow 0.41$	-7.3
0.5	25	$95.8 \rightarrow 86.9$	$0.67 \rightarrow 0.75$	$0.39 \rightarrow 0.42$	-9.2
1	19	$188.9 \rightarrow 113.2$	$0.53 \rightarrow 0.71$	$0.57 \rightarrow 0.76$	-40.1
1	24	$197.2 \rightarrow 103.6$	$0.56 \rightarrow 0.69$	$0.55 \rightarrow 0.76$	-47.5
1	25	$193.1 \rightarrow 104.4$	$0.39 \rightarrow 0.71$	$0.56 \rightarrow 0.77$	-45.9
1.5	19	$368.0 \rightarrow 172.2$	$0.38 \rightarrow 0.44$	$0.58 \rightarrow 0.91$	-53.2
1.5	24	$356.5 \rightarrow 165.2$	$0.36 \rightarrow 0.46$	$0.59 \rightarrow 0.93$	-53.7
1.5	25	$287.2 \rightarrow 135.5$	$0.50 \rightarrow 0.61$	$0.67 \rightarrow 1.01$	-52.8

Table 8: RTX collaborative workload: two additional seeds with ordinary mapping.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
0.5	26	$90.1 \rightarrow 85.9$	$0.76 \rightarrow 0.83$	$0.40 \rightarrow 0.43$	-4.7
0.5	27	$71.9 \rightarrow 71.6$	$0.88 \rightarrow 0.88$	$0.42 \rightarrow 0.42$	-0.4
1	26	$236.1 \rightarrow 118.3$	$0.47 \rightarrow 0.67$	$0.51 \rightarrow 0.74$	-49.9
1	27	$179.1 \rightarrow 107.9$	$0.57 \rightarrow 0.71$	$0.57 \rightarrow 0.74$	-39.8
1.5	26	$316.8 \rightarrow 150.0$	$0.38 \rightarrow 0.46$	$0.64 \rightarrow 0.95$	-52.7
1.5	27	$300.8 \rightarrow 136.1$	$0.39 \rightarrow 0.58$	$0.65 \rightarrow 1.02$	-54.8

B.11 Additional algorithm and dataset results

The aggregation-only workload keeps three proposers followed by one aggregator for every request. At 4.5 workflows/s, p95 falls by 52–56% across three seeds, with all requests meeting the 60-second SLO. Its light-load points show that algorithm identity alone does not determine the speedup. The task-specific mixes use the primary orchestration classes with different prompt datasets. MMLU-Pro/GPQA and ARC-Challenge/WinoGrande each have two seeds. Tables report all tested rates and seeds in these studies. They measure serving performance for fixed work, not changes in answer quality.

Table 9: Three-proposer aggregation: fixed slices → KVLease. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
2.25	19	36.3 → 36.0	1.00 → 1.00	2.13 → 2.13	−0.9
3	19	38.1 → 37.0	1.00 → 1.00	2.75 → 2.76	−2.8
3	24	41.7 → 39.1	1.00 → 1.00	2.76 → 2.80	−6.3
3	25	39.0 → 34.7	1.00 → 1.00	2.73 → 2.73	−10.8
4.5	19	83.8 → 39.9	0.65 → 1.00	3.54 → 4.10	−52.4
4.5	24	87.6 → 38.9	0.57 → 1.00	3.50 → 4.04	−55.6
4.5	25	92.5 → 40.5	0.48 → 1.00	3.53 → 4.11	−56.3

Table 10: MMLU-Pro and GPQA: fixed slices → KVLease. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1.5	24	113.8 → 59.9	0.47 → 0.95	1.09 → 1.33	−47.3
1.5	25	77.4 → 45.9	0.77 → 1.00	1.23 → 1.34	−40.7
2	24	194.7 → 86.1	0.32 → 0.73	1.20 → 1.65	−55.8
2	25	120.5 → 54.2	0.40 → 0.99	1.37 → 1.73	−55.1
2.6	24	233.8 → 127.9	0.33 → 0.38	1.38 → 1.93	−45.3
2.6	25	205.1 → 108.8	0.35 → 0.48	1.38 → 1.89	−47.0

Table 11: ARC-Challenge and WinoGrande: fixed slices → KVLease. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
2.1	19	59.6 → 48.1	0.95 → 0.96	1.73 → 1.79	−19.3
2.1	24	34.3 → 33.1	0.97 → 0.97	1.87 → 1.87	−3.4
2.75	19	98.1 → 47.9	0.67 → 0.96	2.03 → 2.25	−51.2
2.75	24	46.9 → 38.2	0.96 → 0.97	2.31 → 2.36	−18.6
3.6	19	132.1 → 58.5	0.50 → 0.95	2.33 → 2.89	−55.8
3.6	24	134.0 → 52.0	0.48 → 0.96	2.49 → 3.06	−61.2

B.12 Direct allocation-policy controls

Table 12 changes only the retention signal while keeping lending, guarantees, and targeted return enabled. Omitting the adaptive controller preserves the initial cache-retention shares; it does not disable prefix caching. Every arrival completes in these comparisons. The result supports the simpler contract and does not establish a benefit from join-wait adaptation.

A separate periodic controller recomputes hard quotas from recent demand every 10 or 30 seconds, using the same elastic implementation and page-return mechanism. It prevents borrowing above those quotas. This is an internal policy control, not an implementation of MuxServe. In seed 20260919, at B200 rates 1.5/2/2.5 workflows/s, p95 is 54.0/62.4/100.3 seconds for 10-second updates, 55.0/64.7/96.5 for 30-second updates, and 53.8/60.0/97.9 for KVLease. On RTX at rates 0.5/1/1.5, the corresponding values are 80.3/114.0/171.8, 79.9/116.6/171.2, and 80.5/113.3/169.2 seconds. The RTX replication (seed 20260924) gives 78.7/109.2/166.0, 78.6/105.1/160.1, and 78.4/102.8/158.8 seconds. All arrivals complete. Comparable results with periodic quotas locate the contribution in recoverable capacity across engines, rather than a uniquely necessary feedback rule.

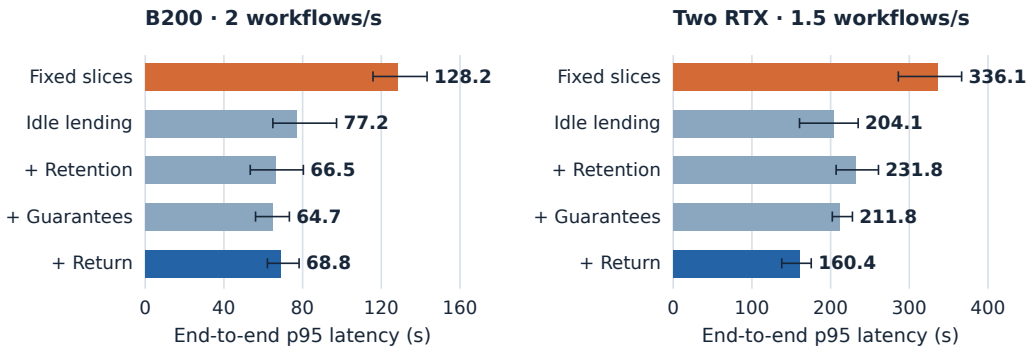

Figure 5: **Lending unlocks capacity; return controls interference.** Cumulative ablation at 2 workflows/s on B200 and 1.5 workflows/s on RTX, means over three replications, with whiskers spanning their minimum and maximum. These are separate paired runs from Table 1. The full policy prioritizes both aggregate completion and recovery of lenders' capacity.

Table 12: Full allocator with different retention signals. End-to-end p95 in seconds; paired within each row.

GPU	Seed	Workflows/s	Fixed shares	Join wait	Call occupancy
B200	20260919	2.0	58.5	61.0	60.3
B200	20260924	2.0	66.5	69.6	77.5
RTX	20260919	1.5	179.9	177.7	178.4
RTX	20260924	1.5	165.9	170.7	172.8

B.13 Profile quality and recoverable guarantees

The demand-fitted comparator is a heuristic constructed from realized occupancy on the evaluated mix. For each model, the fit takes the time-weighted mean live KV bytes, applies a minimum footprint, and normalizes these values to sum to one. For the B200 profile below, the minimum is 12,288 tokens times each model's KV bytes per token. It is an in-sample diagnostic, not a proven optimal static allocation. At the highest primary rate, its p95 versus default KVLease is 81.7 versus 98.2 and 85.2 versus 96.9 seconds on B200 (seeds 20260919/24, 2.5 workflows/s); RTX gives 140.7 versus 174.7 and 144.0 versus 164.0 seconds (1.5 workflows/s). At moderate load their latencies are close. This agrees with the distinction between partition shape and temporal sharing: a well-shaped static partition can capture much of the former.

Table 13 compares using the same profile as a hard partition or as guarantees that allow borrowing. The first column is a separate heavy-load diagnostic on the fitting mix; the second applies the seed-20260919 profile to a Llama/Mistral-heavy mix with Azure conversation timing. Results are paired within a column, with every arrival completed. The B200 profile fractions, in Gemma/gpt-oss/Mistral/Llama/Granite order, are approximately $(0.379, 0.183, 0.048, 0.048, 0.342)$. Giving both methods the same information isolates the role of a hard ceiling versus recoverable borrowing. This

Table 13: The same profile as a static partition or recoverable guarantees. End-to-end p95 in seconds; one paired replication per column.

Allocation	Fitting mix, 2.5 workflows/s	Shifted mix, 3 workflows/s
Equal-token fixed	—	67.0
Profiled static	82.4	102.3
KVLease, default guarantees	98.8	44.5
KVLease, profiled guarantees	83.6	44.3

diagnostic tests one recorded change of workload mix. Profiled guarantees preserve the calibration priority while permitting borrowing beyond it. Calibration incorporates call residence times as well as the workflow graph.

B.14 RESIDENT CATALOGUE AND AVAILABLE MEMORY

The eight-model catalogue adds Phi-4, Granite-3.3-2B-Instruct, and Gemma-2-2B-it to the primary five models. The ten-model catalogue additionally includes Llama-3.2-1B-Instruct and OLMo-2-1124-7B-Instruct.

Startup accounting separates model weights, engine overhead and KV capacity. With ten resident models on B200, weights occupy 128.1 GiB and non-KV overhead 18.0 GiB, leaving approximately 33.0 GiB from 179.1 GiB of device memory. The 24-GiB pool uses 73% of this residual. The five-model catalogue leaves 94.4 GiB; its 88-GiB comparison therefore checks a substantially less constrained setting than the primary 32-GiB experiments. Table 14 holds the catalogue and workload fixed within each comparison; role assignments change with catalogue size, so it is not a controlled scaling curve across rows. The ten-model rate-1.5 setting has three seeds; its rate-1 setting has one. Eight models leave approximately 49 GiB, of which the tested pool uses 44 GiB. Five- and eight-model comparisons each have one seed. Every arrival completes.

Table 14: Ten resident models: fixed slices → KVLease. All tested seeds and rates.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	113.8 → 64.2	0.57 → 0.93	0.68 → 0.83	−43.6
1.5	19	214.7 → 103.6	0.45 → 0.63	0.75 → 1.10	−51.7
1.5	24	205.9 → 95.1	0.41 → 0.66	0.87 → 1.19	−53.8
1.5	25	175.8 → 83.7	0.46 → 0.83	0.87 → 1.23	−52.4

Table 15: Eight resident models, 44 GiB: fixed slices → KVLease. All tested seeds and rates.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	122.2 → 107.0	0.73 → 0.77	0.63 → 0.71	−12.4
1.5	19	231.3 → 171.0	0.60 → 0.61	0.71 → 0.88	−26.1

Table 16: Five resident models, 88 GiB: fixed slices → KVLease. All tested seeds and rates.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	51.8 → 52.4	1.00 → 1.00	0.84 → 0.84	1.0
1.5	19	69.1 → 71.7	0.87 → 0.86	1.20 → 1.19	3.8

B.15 ADDITIONAL DEPLOYMENT COMPARISONS

All comparisons hold total GPU KV capacity at 32 GiB. Equal GPU-memory fractions assign each model the same fraction of device memory; subtracting its weights and overhead gives model-specific KV slices (Gemma 4.21, gpt-oss 7.26, Mistral 7.92, Llama 6.44, Granite 6.18 GiB, rounded). This differs from equal token capacities. The comparison below retains the same total capacity and workload.

The offload baseline adds 4 GiB of host KV storage per model to fixed GPU slices through vLLM's native KV offloading connector. All configurations use FLASH_ATTN, except gpt-oss uses TRITON_ATTN; the offload configuration disables the hybrid KV manager for compatibility. This connector retains computed prefixes for reuse, rather than swapping active decode state. External-cache hit rates are 0–0.2% here; offload changes p95 by -3 to $+2\%$ relative to fixed slices. The result concerns this reuse pattern, not workloads with frequent reusable prefixes.

The vanilla-SGLang comparison uses SGLang with fixed capacity versus the vLLM-based KVLease implementation. It is an end-to-end system comparison. The separate two-seed within-SGLang experiment in Section 5 isolates installing the pool within that engine.

Table 17: Equal GPU-memory fractions: baseline → KVLease, all paired seeds and tested rates. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1.5	19	115.1 → 54.4	0.61 → 0.98	0.98 → 1.27	−52.7
1.5	24	113.1 → 55.3	0.61 → 0.97	1.09 → 1.34	−51.1
2	19	146.2 → 61.8	0.59 → 0.93	1.21 → 1.64	−57.7
2	24	180.4 → 82.5	0.54 → 0.79	1.16 → 1.65	−54.2
2.5	19	230.8 → 112.0	0.53 → 0.57	1.24 → 1.80	−51.5
2.5	24	209.6 → 97.0	0.52 → 0.67	1.34 → 2.00	−53.7

Table 18: CPU KV offload comparison: baseline → KVLease, all paired seeds and tested rates. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
2	19	167.7 → 87.4	0.49 → 0.72	1.17 → 1.55	−47.9
2	24	188.7 → 103.1	0.43 → 0.59	1.17 → 1.55	−45.4
2.5	19	247.2 → 143.2	0.35 → 0.44	1.23 → 1.69	−42.1
2.5	24	230.7 → 129.3	0.40 → 0.47	1.31 → 1.81	−44.0

Table 19: Vanilla SGLang comparison: baseline → KVLease, all paired seeds and tested rates. Seed labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
2	19	116.4 → 64.0	0.59 → 0.92	1.34 → 1.64	−45.0
2	24	133.4 → 84.0	0.50 → 0.79	1.35 → 1.65	−37.0
2	25	113.8 → 76.7	0.55 → 0.84	1.41 → 1.66	−32.6
2.5	19	164.2 → 103.9	0.40 → 0.62	1.51 → 1.87	−36.7
2.5	24	158.8 → 108.0	0.44 → 0.61	1.60 → 1.94	−32.0
2.5	25	179.5 → 124.3	0.40 → 0.49	1.47 → 1.74	−30.8

B.16 SUSTAINED ELASTIC COMPARISON

Each seed supplies 1,800 arrivals over 30 minutes at 1 workflow/s on two RTX GPUs. KVLease completes all arrivals in both runs; kvcached has 106 and 111 timeouts in seeds 19 and 24, respectively. Timed-out arrivals receive the 600-second deadline in latency accounting and remain SLO misses. Throughput includes the completion interval after arrivals stop. These finite-window results do not assume that offered rate equals long-run sustainable throughput.

Table 20: Thirty-minute runs: kvcached → KVLease. Each row includes all 1,800 arrivals, including baseline timeouts.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	$600.0 \to 119.4$	$0.31 \to 0.64$	$0.78 \to 0.96$	-80.1
1	24	$600.0 \to 193.7$	$0.30 \to 0.44$	$0.79 \to 0.97$	-67.7

B.17 WITHIN-ENGINE SGLANG COMPARISON

Both configurations use SGLang 0.5.10.post1, the five-model catalogue, a 32-GiB GPU KV budget, and the same replayed workflow calls. The gpt-oss engine uses the Triton attention backend with page size one in both configurations. This comparison isolates installing the pool inside SGLang; it is separate from comparing vLLM-based KVLease against vanilla SGLang. All arrivals complete, and each seed passes 45 fixed-batch output checks before load, after load, and after idle.

Table 21: Fixed-slice SGLang → SGLang with the KVLease pool. All tested rates and both seeds; labels abbreviate 202609xx.

Rate	Seed	p95 (s)	SLO fraction	Completed/s	Change (%)
1	19	$45.7 \to 45.1$	$0.99 \to 0.99$	$0.85 \to 0.86$	-1.3
1.5	19	$93.3 \to 66.1$	$0.67 \to 0.93$	$1.07 \to 1.21$	-29.1
1.5	24	$90.2 \to 64.6$	$0.61 \to 0.94$	$1.14 \to 1.32$	-28.4
2	19	$136.0 \to 87.0$	$0.52 \to 0.68$	$1.30 \to 1.52$	-36.0
2	24	$152.8 \to 93.6$	$0.44 \to 0.63$	$1.28 \to 1.57$	-38.8
2.5	24	$177.5 \to 116.5$	$0.42 \to 0.56$	$1.49 \to 1.86$	-34.4

B.18 ALLOCATION WAITS AND MECHANISM MEASUREMENTS

Below-guarantee episode duration measures consecutive telemetry samples with a queued engine below its guarantee and a peer above its own. The queued engine must also be below its sequence-concurrency limit. We average episode durations within each run, then across seeds; this measures persistence of the allocation mismatch, not individual-request latency or a promised return deadline.

The conditional page-return bound concerns selected page groups; these queueing episodes measure allocation mismatch across engines.

B.19 COMPATIBILITY WITH SPECULATIVE DECODING

Speculative decoding. Both configurations use EAGLE-3 with three draft tokens for gpt-oss, Llama, and Granite, and prompt n-gram speculation with four draft tokens and lookup lengths 2–5 for Gemma and Mistral. With these same settings in the baseline and KVLease, the pool retains a mean p95 reduction of 46% at the highest arrival rate for the collaborative B200 workload (two replications), 40% for the routing mix (three replications), and 59% on RTX (three replications). Memory allocation and faster per-model decoding are complementary.