RETHINKING KV CACHE OFFLOADING: BANDWIDTH SCALING AND DECODE CONTINUITY

Anonymous authors
Paper under double-blind review

ABSTRACT

KV cache offloading trades data movement for memory capacity in language model serving. As offload bandwidth increases, should systems continue to prioritize minimizing transferred bytes? We study this question using a storage-only NVLink peer as a controlled surrogate for a high-bandwidth remote tier, with inference resources held fixed. A bandwidth sweep reveals a reversal: request-granular migration begins slower than two fine-grained policies but becomes faster as transfer cost falls. Load sweeps and runtime analysis connect this reversal to a broader design concern: preserving useful decoding while recovering memory capacity. We formulate three design rules and instantiate them in CARGOKV: amortize migration interruptions, recover headroom with few logical actions, and bound migration activity. Across three dense models and six workload regimes, comparisons with LayerKV, NEO, ZeRO-Inference, and FlexGen reveal both gains and boundaries. At the highest tested static latency loads, CARGOKV reduces P99 time per output token by 24.3–49.3% relative to SOTA. The resulting guideline is conditional and portable: a fast offload path should be designed around the exposed cost of interrupting decoding, alongside the bytes it moves. We will make the data and code publicly available upon acceptance.

1 INTRODUCTION

Autoregressive language models retain keys and values of preceding tokens to avoid recomputation. This KV cache grows with sequence length and concurrency, competing with weights and runtime state for GPU memory. Efficient allocation reduces waste (Kwon et al., 2023), but cannot eliminate the capacity required by live contexts. Under capacity pressure, exact offloading must decide which state remains near computation, which moves to another tier, and when suspended computation resumes.

The familiar answer emphasizes transfer efficiency. On a bandwidth-constrained path, unnecessary bytes directly delay inference, motivating carefully staged layer-, tensor-, or block-level movement (DeepSpeed Team, 2022; Sheng et al., 2023; Xiong et al., 2024). Fine granularity can improve residency and overlap, but it also introduces scheduling, bookkeeping, and synchronization boundaries. A faster link reduces one side of this trade-off without necessarily reducing the other. The question is therefore not simply whether offloading becomes faster, but *whether the organization of offloading should change as bandwidth scales*.

We approach this question as a design study of future high-bandwidth offload paths. A storage-only peer GPU connected through NVLink supplies a controllable remote tier; separate GPUs execute inference. This construction provides *transfer-rate equivalence along the bandwidth dimension*: it lets us study the balance between moving KV state and coordinating that movement as transfer cost falls. The surrogate exposes this transition without assuming the cost, latency, or contention characteristics of a future memory product.

Our analysis follows three questions. First, does fine-grained migration keep benefiting from additional bandwidth? In the controlled sweep, request migration is initially slower but overtakes the tested finer policies. Second, why does this matter to serving? A load sweep on the same workload shows heavier token-latency tails and faster loss of SLO-goodput for the finer policy. Third, what runtime behavior accompanies this difference? The critical-path decomposition and serving telemetry

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.

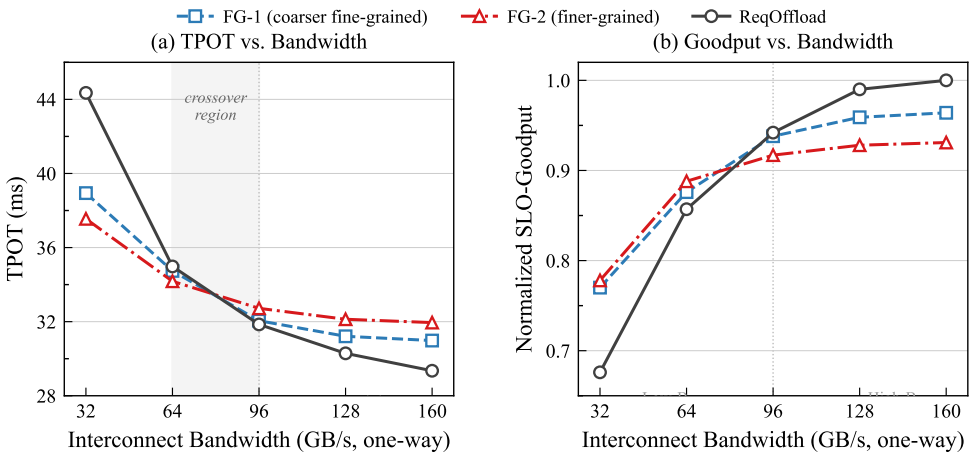

Figure 1: Controlled bandwidth sweep: TPOT (left) and SLO-goodput relative to native memory-fit service (right). The shaded interval brackets the observed policy reversal between 64 and 96 GB/s.

expose coordination costs and gaps that faster transfers do not eliminate. The progression moves from a policy reversal, to its service consequence, to the runtime mechanism that a design must address.

This progression motivates an *execution-continuity principle*: optimize the exposed cost of migration on the decode path, accounting jointly for transfer, coordination, and delayed progress. Request-level movement provides one realization; batching and overlapping finer transfers may provide another. Request-level preemption itself is established (Wu et al., 2026; Jiang et al., 2025). Our contribution is the bandwidth-dependent design argument and its evaluation, rather than a new claim to that primitive.

Our contributions are:

- **A bandwidth-dependent policy reversal.** Controlled bandwidth, load, and runtime studies show why finer migration can lose its advantage as transfer becomes cheaper.
- **Design principles for fast offload paths.** We derive rules for amortizing interruptions, recovering headroom with few actions, and regulating migration against decode progress.
- **An instantiation and evaluation.** CARGOKV implements the rules. Three-model, four-baseline comparisons and component studies quantify gains and boundaries; simulation explores conditional scale-out behavior.

2 EMPIRICAL STUDY: WHAT CHANGES WHEN BANDWIDTH INCREASES?

We first study migration policies before introducing a complete system. The three questions proceed from bandwidth scaling, through its service-level consequence, to the critical-path behavior that motivates a design. Here, REQOFFLOAD denotes only whole-request KV movement; victim selection, reload ordering, and feedback control are introduced in Section 3.

2.1 EXPERIMENTAL LENS AND SCOPE

The controlled study uses an eight-A100-80GB NVLink/NVSwitch server, with one GPU executing Llama-2-13B in FP16 and a second storing remote KV state without running inference kernels. Prompts contain 4096 tokens and generation is configured for 1024 tokens; the recorded configuration uses 2× RoPE scaling to accommodate the resulting sequence length. All policies share serving computation and the remote path. FG-1 migrates KV at layer granularity, FG-2 uses finer tensor granularity, and the request policy migrates a request's complete KV state. FG-1 and FG-2 are controlled policy variants inspired by conventional offloading designs, not measurements of unmodified product implementations. The recorded sweep uses one-way bandwidth labels of 32, 64, 96, 128, and 160 GB/s. The maximum label rounds a measured effective throughput of approximately 158 GB/s;

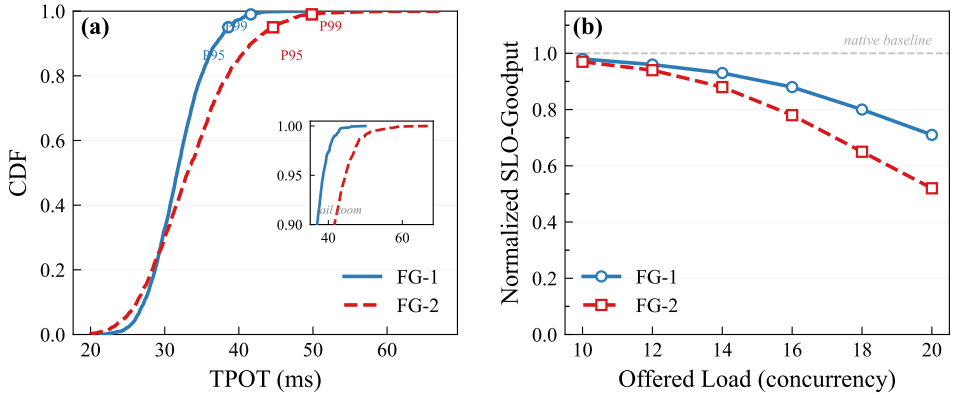

Figure 2: High-bandwidth service on the controlled workload. Left: TPOT distribution at a representative load. Right: SLO-goodput relative to native memory-fit service as outstanding requests increase. Offered load counts outstanding requests, including suspended ones; it is not GPU-resident concurrency.

it is distinct from the 175 GB/s nominal capacity of seven enabled links. Bandwidth is varied by changing the number of enabled NVLink links (Appendix A); the sweep characterizes policy ordering on this controlled path. We use "high bandwidth" operationally: transfer time has fallen enough that other exposed costs can materially influence policy choice. Matching this transfer-rate dimension does not match endpoint latency, access granularity, contention, or overlap.

2.2 RQ1: Does Finer Migration Continue to Benefit from More Bandwidth?

Figure 1 shows that all three policies improve as bandwidth rises, but by different amounts. At 32 GB/s, request-granular migration takes 44.35 ms/token, versus 38.94 for FG-1 and 37.55 for FG-2. The request policy is thus 13.9% and 18.1% slower. This is the expected cost of moving a larger unit when transfer is expensive. Near the middle of the sweep, the difference narrows and then changes sign. At the maximum setting, request migration takes 29.35 ms/token, versus 30.98 and 31.95, reductions of 5.3% and 8.1%. Normalized SLO-goodput changes in the same direction: the request policy rises from 0.676 to 1.000, versus 0.770 to 0.964 for FG-1 and 0.778 to 0.931 for FG-2. Fine-grained improvement diminishes rather than vanishing; FG-1 still improves from 31.21 to 30.98 ms/token between the last two settings. These controlled effects motivate, but do not determine, the larger differences between complete serving configurations.

Insight 1. The preferred migration organization can change with bandwidth. In this controlled path, paying for larger transfers becomes worthwhile when it sufficiently reduces other exposed costs; finer granularity is not uniformly optimal.

2.3 RQ2: How Does the Trade-off Appear Under Serving Load?

We retain the Llama-2-13B workload and compare FG-1 and FG-2 at the highest bandwidth setting, increasing outstanding requests beyond the native memory-fit ceiling $c^\star = 14$. Figure 2 shows that the distinction is most visible in the TPOT tail: FG-2 has a heavier upper tail even where much of the distributions is close. The companion load sweep shows the associated service loss. At load 14, normalized SLO-goodput is 0.93 for FG-1 and 0.88 for FG-2; at load 20 it is 0.71 and 0.52. The finer policy therefore delivers 26.8% less goodput at the latter point despite using the same fast transfer path. The shared high-bandwidth path therefore does not erase the service-level difference between the policies. Transfer throughput alone is an incomplete objective. Requests sharing a worker depend on the same decode schedule; coordination and loss of runnable work can delay generation beyond the request whose state moves. Aggregate distributions do not isolate each request's migration history, so we next examine the critical path.

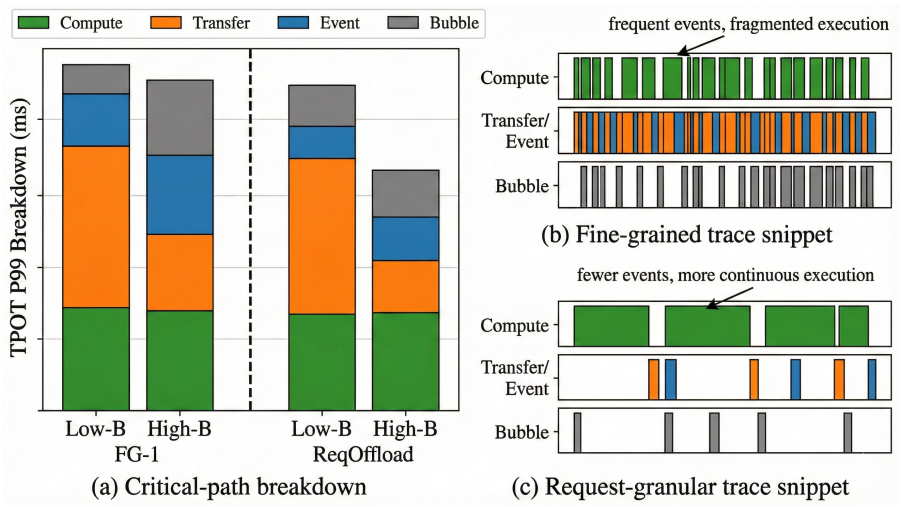

Figure 3: Critical-path comparison across bandwidth regimes (left) and representative execution patterns (right). The snippets illustrate fragmentation and continuity; their drawn lengths are not a separate latency measurement. "Event" denotes migration coordination, while "Bubble" denotes a gap in useful decoding.

Insight 2. The cost of migration is a service-level cost. Under memory pressure, a fine-grained policy can retain heavier decode tails and lose more SLO-goodput even on a fast link; protecting useful decode progress must accompany transfer optimization.

2.4 RQ3: Which Costs Remain After Transfer Becomes Faster?

Figure 3 separates the cost of moving state from the runtime activity around that movement. Faster transfers reduce a major component for both policies, but leave coordination and fragmented execution exposed in the fine-grained case. We use the following accounting lens for an observed decode interval:

$$T_{\text{decode}} = T_{\text{compute}} + T_{\text{transfer,exposed}} + T_{\text{coord,exposed}} + T_{\text{gap}}. \tag{1}$$

"Exposed" means that a cost lies on the observed path after overlap; Equation 1 is an accounting convention rather than an assumption that all raw copy and kernel durations add. A gap is an interval without useful decoding in that accounting. It need not be a hardware stall cycle, nor is every gap uniquely attributable to migration. The decomposition shows two different responses to faster movement. With FG-1, the reduction in transfer time leaves a larger fraction of the observed path in event overhead and gaps. With ReqOffload, fewer migration boundaries preserve longer compute intervals, so more of the transfer improvement reaches useful decoding. The execution snippets illustrate this distinction in time: many small transfers interleave short compute segments, whereas request movement concentrates migration activity and leaves longer uninterrupted segments. Together, the observations connect cheaper transfer (RQ1), service loss under load (RQ2), and the residual interruptions a policy must address (RQ3). Transferred bytes still matter, and noncontiguous KV blocks may require several physical copies per request. The implication is to optimize the *exposed effect* of movement, rather than migration size alone.

Insight 3. As transfer time shrinks, exposed coordination and execution gaps can limit the benefit of additional bandwidth. The design target is their combined effect on decoding, not a particular logical migration size by itself.

3 Design Principles for High-Bandwidth KV Offloading

The three insights suggest a change in objective: recovering memory should preserve useful decoding, not merely minimize transfer volume. This objective connects three decisions—migration granularity,

capacity recovery, and migration intensity. We first develop the principles governing these decisions, then instantiate them in a request-based controller.

3.1 From Empirical Insights to Design Rules

Rule 1: Amortize interruptions while accounting for extra bytes. As transfer becomes cheaper, a larger migration unit can be worthwhile if it avoids enough exposed coordination and interruption. For a coarse policy c and fine policy f, this trade-off can be expressed as

$$\frac{S_c - S_f}{B} < (H_f - H_c) + (W_f - W_c), \tag{2}$$

where S is transferred volume, B is effective bandwidth, H is exposed coordination and interruption cost, and W is policy-induced waiting. The comparison assumes comparable computation and exposed transfer; overlap changes the transfer term. It explains why the preferred granularity can change with bandwidth, without prescribing a universal unit. Whole-request movement amortizes migration boundaries; fusing fine-grained actions can also reduce H_f. Either choice must compensate for its byte and waiting costs.

Rule 2: Recover headroom with few logical actions. The second insight links migration organization to service under memory pressure. Capacity recovery should create room for sustained decoding without repeatedly interrupting the worker. For indivisible migration units with equal logical action cost, a local formulation is

$$\min_{V \subseteq \mathcal{A}} |V| \quad \text{subject to} \quad \sum_{r \in V} s(r) \geq \Delta, \tag{3}$$

where Δ is the capacity deficit and $s(r)$ the recoverable footprint. At a fixed snapshot, taking units largest-first minimizes their number (Appendix F). This objective captures action overhead; transferred bytes and future reload waiting still enter Rule 1's broader trade-off. Offload selection and reload ordering therefore serve different purposes: creating headroom and restoring progress.

Rule 3: Regulate migration by its effect on decoding. The third insight shows why available link capacity is insufficient guidance for migration intensity. Even short transfers can repeatedly disrupt useful execution. A controller should bound outstanding migration and reduce issuance when decoding is interrupted, while retaining enough capacity recovery to admit useful work. This couples memory pressure to execution progress: reclaiming space is valuable only insofar as it helps the worker continue serving requests.

3.2 A Request-Based Instantiation

CargoKV makes these principles concrete using whole-request KV migration (Figure 4). Moving a request suspends its decoding until restoration while other eligible requests continue. A high watermark triggers largest-first victim selection to cover the deficit to a low watermark. Reload candidates are ordered by remaining admission-time generation budget, $\widehat{r}(r) = \max(0, b_r - g_r)$, where b_r is the configured output budget and g_r the generated count. This online completion heuristic requires no knowledge of the eventual EOS position; it can postpone long requests and provides no starvation bound.

A per-epoch action budget, an in-flight cap, and reactive backoff implement Rule 3. Backoff reduces the budget when the decode-gap indicator exceeds its threshold. The in-flight cap can dominate issuance, so the guard ablation evaluates their combined effect. Defaults and this interaction appear in Appendix A; Algorithm 1 gives the selection policy.

The implementation extends vLLM 0.9.0 and PagedAttention with remote-state metadata and asynchronous migration. Requests transition among OnGPU, InFlight, and Offloaded; their retained KV values remain exact. Planned recovery and projected occupancy guide selection, while safe execution requires a consistent snapshot, retaining source state until copy completion, reserving reload space, and resuming only after restoration. Appendix A details the execution semantics. This instantiation lets us evaluate the rules together; the rules themselves do not require this storage endpoint or request-sized units.

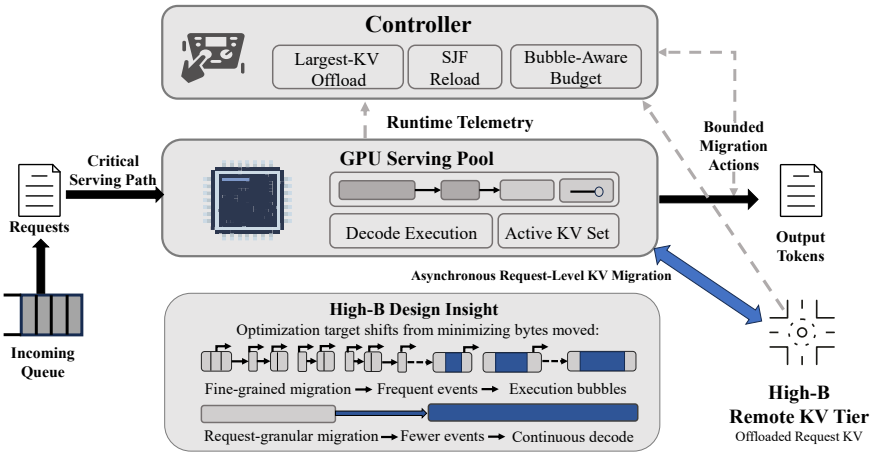

Figure 4: An instantiation of the design principles in CARGOKV. Serving workers execute inference; a storage-only peer GPU retains suspended requests' KV state. The diagram's SJF reload module uses remaining admission-time generation budget.

4 EVALUATION

We evaluate how the design rules improve tail latency and SLO-goodput under static and bursty load (RQ4), then explain the gains through runtime breakdowns and component ablations (RQ5).

4.1 EXPERIMENTAL SETUP

Fixed resource budget. The end-to-end experiments use the same eight-A100-80GB server, with four GPUs assigned to inference and one dedicated to remote KV storage. The storage peer runs no inference; three GPUs remain unused. All methods receive this allocation and link capacity: seven active links, nominally 175 GB/s one-way and approximately 158 GB/s measured effective throughput.

Models and workloads. We evaluate Qwen3-32B, Qwen2.5-72B, and Qwen1.5-110B (Yang et al., 2025; 2024; Qwen Team, 2024), with Llama-3-8B (Meta, 2024) included in the ablations. Static buckets use input/output token scales S=512, M=2048, and L=4096, yielding L-L, L-S, M-M, S-S, and S-L. Prompts are constructed from ShareGPT material (Chiang et al., 2023; Sun et al., 2024). The BurstGPT-derived shock–recovery workload (Wang et al., 2025) has mean input/output lengths of 2048 tokens and arrival CV near 3.0. Static plots index closed-loop in-flight concurrency; the burst sweep indexes peak active requests. Appendix A details workload construction.

Baselines. LayerKV and NEO are the primary KV-serving comparisons (Xiong et al., 2024; Jiang et al., 2025); ZeRO-Inference and FlexGen provide conventional offloading references (DeepSpeed Team, 2022; Sheng et al., 2023). LayerKV combines layer-wise KV management with serving scheduling, while NEO incorporates CPU attention and GPU–CPU pipelining. We compare their complete serving configurations under the common resource budget. The FG-1/FG-2 study and component ablations isolate design choices. Configurations appear in Appendix A.

Metrics and repetitions. We report request-level P99 time per output token (TPOT) and normalized SLO-goodput. Decode wall time includes suspension and restoration between the first and last generated tokens. The decode SLO is fixed to native no-offload P99 TPOT at the same model–workload's maximum memory-fit concurrency $c^\star$. SLO-goodput measures completed-request throughput meeting this decode SLO, normalized by the native reference throughput. This decode-side criterion excludes TTFT. Results average two real-node runs.

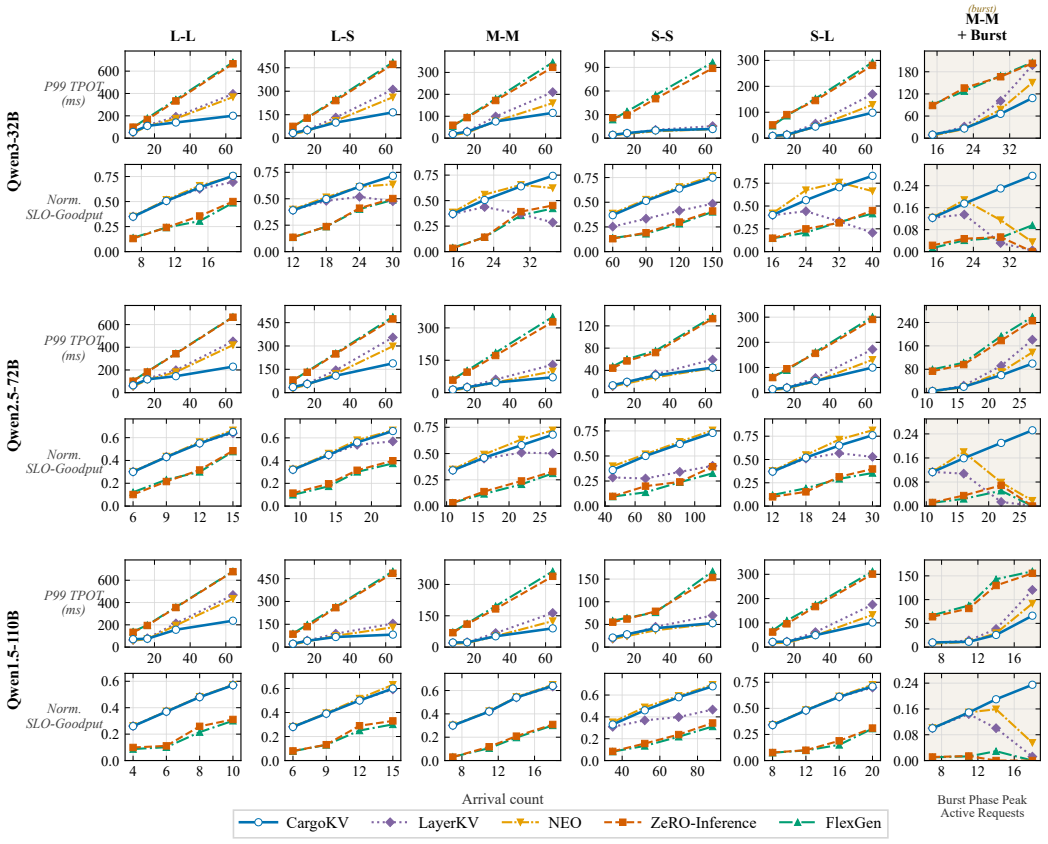

Figure 5: Five-method end-to-end comparison across three models, five static workload buckets, and one burst construction. TPOT and goodput panels use distinct load grids. Static axes labeled "Arrival count" denote closed-loop concurrent requests; the burst axis denotes peak active requests. Goodput is normalized to native memory-fit service.

4.2 RQ4: End-to-End Service Under Static and Bursty Load

Stronger tail-latency gains as pressure increases. At the largest static latency-load setting in each of the 15 model–workload panels, CargoKV reduces P99 TPOT by 24.3–49.3% relative to LayerKV. Relative to NEO, reductions reach 45.4% on L-L, while S-S remains close (2.1% higher TPOT). The benefit concentrates where longer-lived KV state intensifies capacity pressure. On Qwen2.5-72B/M-M, CargoKV, LayerKV, and NEO start at 14.10, 12.76, and 10.32 ms; at the largest load they reach 71.00, 129.30, and 98.46 ms. The ordering reverses as pressure grows: organizing recovery into fewer interruptions becomes increasingly valuable for sustaining decode progress.

Converting decode continuity into useful service. At the largest static goodput-load settings, CargoKV delivers 0.997–4.007× LayerKV's and 0.934–1.253× NEO's goodput. On Qwen2.5-72B/M-M at concurrency 27, the respective normalized values are 0.680, 0.502, and 0.721: CargoKV improves on LayerKV by 35.5%, while NEO achieves the highest goodput at this load. Migration organization and heterogeneous execution address complementary pressures on decode progress and attention computation. Against ZeRO-Inference and FlexGen, CargoKV reduces static-endpoint P99 TPOT by 60.3–87.9% and increases goodput by 34.0–132.0%, retaining 0.570–0.831 of native goodput. The runtime analysis below connects these service gains to lower exposed transfer and interruption costs.

Maintaining service under bursts. At the largest burst setting, CargoKV retains 0.276, 0.252, and 0.235 normalized goodput for 32B, 72B, and 110B. On 72B, LayerKV and NEO fall to approx-

Table 1: Ablations, normalized separately to full CARGOKV (1.00). Each cell is P99 TPOT / SLO-goodput; lower / higher is better.

Model	Workload	Full	No request unit	No min.-action	No guard
Llama-3-8B	M-M	1.00 / 1.00	1.23 / .79	1.09 / .92	1.08 / .93
	L-L	1.00 / 1.00	1.32 / .73	1.12 / .89	1.13 / .88
Qwen3-32B	M-M	1.00 / 1.00	1.28 / .75	1.12 / .89	1.11 / .90
	L-L	1.00 / 1.00	1.38 / .67	1.15 / .85	1.18 / .82

imately 0.0015 and 0.0180, leaving CARGOKV with a 23.4-percentage-point advantage over the stronger reference. At each model's latency endpoint, P99 TPOT is 45.1% lower than LayerKV and 27.9% lower than NEO. During demand shocks, capacity recovery competes with decoding. Few recovery actions and bounded migration intensity help sustain service, as the component effects below further demonstrate.

4.3 RQ5: RUNTIME EVIDENCE AND COMPONENT CONTRIBUTIONS

Most saved time comes from fewer interruptions. On Qwen2.5-72B/L-L, useful compute remains similar: 118 ms for CARGOKV and 120 ms for each conventional reference (Appendix D). Exposed transfer falls to 70 ms from 230/210 ms for ZeRO-Inference/FlexGen; event-plus-gap time falls to 40.98 ms from 315/338.20 ms. Of the total 436.02/439.22 ms reductions, event-plus-gap savings account for 62.8/67.7%. Thus, the larger contribution comes from recovering execution time between useful decode steps. M-M exhibits the same component ordering, connecting the continuity principle to both workload regimes.

Fewer actions preserve decode time as load grows. At the largest runtime-sweep load, CARGOKV issues 0.18 logical migration actions per output token, versus 1.36 for ZeRO-Inference and 1.72 for FlexGen—reductions of 86.8% and 89.5%. Its non-compute gap ratio is 0.127, versus 0.346 and 0.349, approximately 63% lower. The sweeps and timelines in Appendix D link fewer actions to longer useful decode intervals. Recovering capacity through fewer controller actions amortizes coordination overhead and leaves more of the execution window available for token generation.

Granularity, selection, and pacing contribute together. In Table 1, removing request granularity raises P99 TPOT by 23–38% and reduces goodput by 21–33%, the largest component effect. Its latency penalty increases from M-M to L-L for both models, matching the greater importance of migration organization under longer contexts and generation. Minimum-action selection further reduces interruptions: removing it raises TPOT by 9–15%. The stability guard contributes most on Qwen3-32B/L-L, where its removal increases TPOT by 18% and lowers goodput by 18%. These patterns connect the three rules: granularity amortizes each recovery episode, selection limits the actions needed, and pacing protects ongoing decoding as pressure intensifies.

5 DISCUSSION

The central implication is that more bandwidth can change what an offloading policy should optimize. When moving state is expensive, avoiding bytes has immediate value. As that cost falls, the intervals between useful decode steps become harder to ignore: a policy can transfer efficiently yet repeatedly interrupt the work it is meant to support. The observed policy reversal makes this design tension concrete. It motivates treating decode continuity as an explicit objective when designing future offload paths.

NVLink allows us to investigate this transition today. Holding inference resources fixed while varying the remote transfer path exposes how policy behavior changes as movement becomes cheaper. The storage-only peer is thus an experimental surrogate for a high-bandwidth tier, with equivalence confined to the transfer-rate dimension. The study addresses a class of fast offload paths, rather than forecasting a particular memory product. For another endpoint, latency, contention, and overlap change the balance in Equation 2; the same design question remains: how much useful execution is

lost while memory capacity is being recovered? This is the basis for carrying the insight beyond the testbed without treating different endpoints as interchangeable.

The resulting design space is broader than whole-request migration. A runtime might preserve finer residency decisions while batching their transfers or reducing synchronization boundaries. Such a design would pursue the same continuity objective through a different mechanism. CARGOKV provides one concrete starting point for examining this space: its gains demonstrate the promise of reorganizing migration, while NEO's competitive goodput shows that compute placement remains consequential. Future high-bandwidth designs should therefore consider migration organization and execution scheduling together, rather than inherit granularity choices solely from slower paths.

The evidence covers dense models and the reported context lengths; longer contexts increase the cost of coarse movement, and shared-fabric contention can alter its benefit. Fairness, deployment economics, and uncertainty beyond the two-run summaries remain unresolved. Scale-out results are conditional simulations (Appendix B). These limits bound the evaluated realization while leaving the design question open across future memory architectures.

6 RELATED WORK

From fitting state to organizing its movement. Memory-efficient serving has developed complementary ways to accommodate model and KV state. PagedAttention reduces allocation waste (Kwon et al., 2023); FlexGen and ZeRO-Inference exploit heterogeneous memory through placement and streaming (Sheng et al., 2023; DeepSpeed Team, 2022). For online KV serving, LayerKV couples layer-wise management with scheduling, while NEO combines offloading with CPU attention and heterogeneous pipelining (Xiong et al., 2024; Jiang et al., 2025). These approaches establish that placement, granularity, and execution must be considered together. Our study asks how their balance changes when bandwidth itself becomes the experimental variable. The contribution is the observed change in policy ordering and the continuity-oriented principles it motivates. Request preemption and migration are established mechanisms in FastServe and Llumnix (Wu et al., 2026; Sun et al., 2024); here they serve as tools for examining those principles.

Cache reuse and active-request continuity. Shared and hierarchical KV caches make retained state reusable across serving stages, requests, and memory tiers. Mooncake, LMCache, HiCache, and Strata develop this direction through disaggregation, cache management, and tiered data movement (Qin et al., 2025; Liu et al., 2025; Xie, 2025; Xie et al., 2026). Reuse reduces work needed by subsequent requests; our question concerns the progress of active requests when their state must move under capacity pressure. The two objectives meet at the migration path. Batching and overlap, already used in such cache systems, are alternative ways to reduce exposed movement overhead and can complement the proposed principles. Selective KV retrieval, as in InfiniGen (Lee et al., 2024), additionally changes which state is accessed; our realization retains KV values exactly.

What faster hardware makes visible. IX and KVell show that fast networking and storage shift attention toward software overhead (Belay et al., 2014; Lepers et al., 2019). We examine a specific consequence for autoregressive serving: coordination between KV movement and repeated decode steps can change which migration policy is preferable as bandwidth rises. The bandwidth sweep, service-level analysis, and instantiated rules connect this general systems observation to a concrete offloading design problem.

7 CONCLUSION

Increasing bandwidth changes the balance between moving KV state and keeping decoding productive. The controlled policy reversal, service degradation, and runtime evidence motivate three continuity-oriented rules, instantiated in CARGOKV. Their value lies in guiding migration organization on fast paths: account for exposed interruption alongside bytes, then select granularity and intensity according to the workload and runtime.

AI USE STATEMENT

We used generative AI tools solely to improve the grammar, wording, and readability of this manuscript. All AI-assisted revisions were reviewed by the authors to ensure that they preserved the intended meaning and technical accuracy. The authors take full responsibility for the final content of this manuscript.

REFERENCES

Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 49–65, Broomfield, CO, October 2014. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/belay.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

DeepSpeed Team. ZeRO-Inference: Democratizing massive model inference, September 2022. URL https://www.deepspeed.ai/2022/09/09/zero-inference.html. DeepSpeed technical blog.

Xuanlin Jiang, Yang Zhou, Shiyi Cao, Ion Stoica, and Minlan Yu. NEO: Saving GPU memory crisis with CPU offloading for online LLM inference. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/66a026c0d17040889b50f0dfa650e5e0-Abstract-Conference.html.

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*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/lee.

Baptiste Lepers, Oana Balmau, Karan Gupta, and Willy Zwaenepoel. KVell: the design and implementation of a fast persistent key-value store. In *SOSP 2019*, 2019. URL https://www.sigops.org/s/conferences/sosp/2019/program.html.

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, 2025. URL https://arxiv.org/abs/2510.09665.

Meta. Meta Llama 3 model card, April 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md. Model card; released April 18, 2024.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Qwen Team. Introducing Qwen1.5, February 2024. URL https://qwenlm.github.io/blog/qwen1.5/.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31094–31116. PMLR, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 5831–5841. Association for Computing Machinery, 2025. doi: 10.1145/3711896.3737413. URL https://doi.org/10.1145/3711896.3737413.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. FastServe: Iteration-Level preemptive scheduling for large language model inference. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 57–74, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/wu-bingyang.

Zhiqiang Xie. SGLang HiCache: Fast hierarchical KV caching with your favorite storage backends, September 2025. URL https://lmsys.org/blog/2025-09-10-sglang-hicache/.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical context caching for long context language model serving. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 1–16, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL https://www.usenix.org/conference/osdi26/presentation/xie-zhiqiang.

Yi Xiong, Hao Wu, Changxu Shao, Ziqing Wang, Rui Zhang, Yuhong Guo, Junping Zhao, Ke Zhang, and Zhenxuan Pan. LayerKV: Optimizing large language model serving with layer-wise KV cache management. *arXiv preprint arXiv:2410.00428*, 2024. doi: 10.48550/arXiv.2410.00428. URL https://arxiv.org/abs/2410.00428.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

A Experimental Configuration and Measurement Methodology

The evaluation combines a controlled granularity study, an end-to-end serving comparison, and supplementary mechanism analyses. Table 2 distinguishes their resource budgets and purposes.

Table 2: Experimental scope and resource allocation.

Study	Configuration	Role in the argument
Granularity study	Llama-2-13B, FP16; one inference GPU and one storage peer; 4096+1024 tokens	Compare policy behavior as transfer becomes cheaper
Main comparison	Three Qwen models; four inference GPUs and one storage peer; six workload regimes	Compare five complete serving configurations
Runtime analysis	Qwen2.5-72B; L-L and M-M breakdowns, L-L load sweep and timelines	Examine transfer, coordination, and decoding gaps
Component ablation	Llama-3-8B and Qwen3-32B; M-M and L-L	Examine request granularity, selection, and guards
Scale-out study	Qwen2.5-72B; 4–64 modeled serving pools	Explore conditional scaling beyond one node

A.1 Platform and Workloads

The real-node platform contains eight A100-80GB GPUs with NVLink/NVSwitch. The granularity study assigns GPU0 to inference and GPU1 to remote KV storage; the main evaluation assigns four GPUs to inference (TP=4, PP=1, DP=1) and one to storage. The peer executes no inference kernels. The implementation extends vLLM 0.9.0 and PagedAttention with request-state metadata and asynchronous KV movement. The Llama-2 study uses FP16 and $2\times$ RoPE scaling for the 4096-token prompt and 1024-token generation configuration.

The bandwidth sweep varies the number of enabled NVLink links. At the highest setting, five of twelve links are disabled, leaving seven active links: 175 GB/s nominal one-way capacity and approximately 158 GB/s effective throughput, rounded to 160 GB/s in the sweep.

Static workloads are constructed from ShareGPT material using input/output length scales S=512, M=2048, and L=4096, forming L-L, L-S, M-M, S-S, and S-L. The BurstGPT-derived construction uses an MGP arrival model with reported CV ≈ 3.0 to create clustered arrivals. Its shock phase increases the arrival rate from 5 to 35 requests/s during seconds 90–120; mean input and output lengths are each 2048 tokens. Static workloads use closed-loop control of in-flight concurrency; their load settings denote target concurrent requests. All methods replay the same burst arrival sequence, and the burst sweep is indexed by peak active requests. The 8/16/32/64 requests/s settings in the simulator calibration belong to a separate experiment.

A.2 Service Metrics and Runtime Traces

For a request producing $n_{\text{out}} > 1$ tokens, we define

$$\text{TPOT} = \frac{t_{\text{last}} - t_{\text{first}}}{n_{\text{out}} - 1}, \qquad T_{\text{SLO}} = \text{P99}(\text{TPOT}_{\text{native}}(c^\star)),$$

where $c^\star$ is the maximum native no-offload memory-fit concurrency for the model–workload pair ($c^\star = 14$ in the controlled Llama-2 study). Elapsed decode time includes suspension and restoration; TTFT is excluded. The native threshold is fixed across methods and loads. SLO-goodput counts completed requests satisfying $\text{TPOT} \le T_{\text{SLO}}$ per unit time and is normalized as $G_{\text{norm}} = G_{\text{SLO}}/G_{\text{native}}$, using native goodput at $c^\star$ under the same counting and timing rules.

Warmup is separate from the measured trace. After injection stops, outstanding requests drain to completion; the goodput denominator spans the full measurement interval corresponding to the counted completions, including draining. P99 TPOT and goodput are computed separately for each run and then averaged over two runs.

Runtime traces use `torch.profiler` CPU/CUDA activities and instrumented decode boundaries, analyzed in Perfetto. Within a common observation window, action intensity is the number of logical offload/reload actions divided by output tokens; physical copies are counted separately. The gap ratio divides non-compute gap time by observation time with unfinished decode requests. Compute, exposed transfer, event overhead, and gaps are attributed without double-counting overlaps. A P99 breakdown uses the component vector at the same quantile of total TPOT, rather than separately computed component quantiles. Timeline compute fractions describe the displayed windows.

A.3 Controller Configuration and Execution Semantics

Table 3: Default controller parameters.

Parameter	Value	Function
Δt	50 ms	Control epoch
$(\theta_{\text{hi}}, \theta_{\text{lo}})$	$(0.85, 0.75)$	Offload/reload hysteresis
$B_{\max}$	4	New migration actions per epoch
$K_{\max}$	2	In-flight migration cap
τ	0.10	Stall/gap indicator threshold
γ	0.75	Budget multiplier under backoff

The watermarks separate capacity recovery from restoration. Each 50 ms epoch permits four logical actions, with at most two offloads and reloads in flight jointly. When the gap indicator exceeds $\tau = 0.10$, backoff reduces the epoch budget to $\lfloor 0.75 \times 4 \rfloor = 3$. Completion callbacks refill available slots within the remaining epoch budget. Capacity checks include reserved reload space; an offload releases source memory only after completion. Requests become runnable after reload completion, with OnGPU, InFlight, and Offloaded tracking these transitions. Algorithm 1 gives the corresponding selection procedure.

Ablation configurations. *No request unit* replaces whole-request movement with layer-wise migration while preserving an equivalent parent-request action budget. *No min.-action* replaces largest-first victim selection with FCFS. *No guard* disables the action budget, in-flight cap, and backoff while retaining memory-safety checks. Each ablation otherwise uses the full configuration.

B Scale-out Study and Single-Pool Calibration

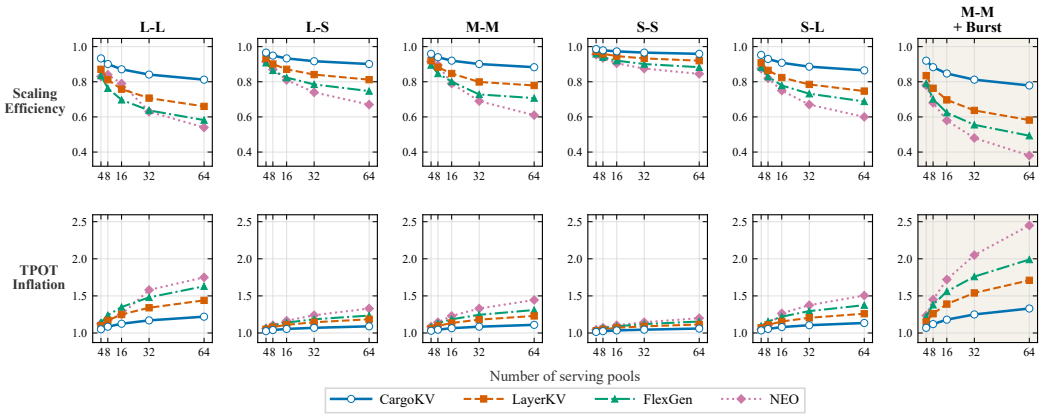

Figure 6: Simulated scale-out study on Qwen2.5-72B at a modeled 158 GB/s offload bandwidth. Each method uses its own single-pool reference. The top row is goodput scaling efficiency; the bottom row is P99 TPOT inflation. These are modeled serving pools, not a measured multi-node cluster.

Each modeled serving pool represents one fixed-budget serving instance with the compute budget of the real-node evaluation. The model scales the number of pools as $k \in \{4, 8, 16, 32, 64\}$, with $k = 1$

as reference, and fixes the one-way offload bandwidth at 158 GB/s. The remote tier is represented abstractly rather than as a particular device. All curves use Qwen2.5-72B and the six workload buckets; the modeled alternatives are FlexGen, LayerKV, and NEO.

Let $\mathrm{GP}^{\star}_{m,w}(k)$ denote the maximum SLO-goodput over the arrival-rate sweep for method m and workload w. The associated $\mathrm{TPOT}^{99}_{m,w}(k)$ is taken at that peak-goodput operating point. The scaling metrics are

$$\eta_{m,w}(k) = \frac{\mathrm{GP}^{\star}_{m,w}(k)}{k\,\mathrm{GP}^{\star}_{m,w}(1)}, \qquad \phi_{m,w}(k) = \frac{\mathrm{TPOT}^{99}_{m,w}(k)}{\mathrm{TPOT}^{99}_{m,w}(1)}.$$

The available calibration compares TPOT at four single-pool load settings and reports 7.69% MAPE. This does not validate goodput, contention, or scaling at 64 pools. The separate five-method real-node matrix in Figure 5 supplies the empirical comparison; simulation supplies a different question about modelled scaling.

Figure 6 reports scaling results across six workloads. At 64 pools, L-L scaling efficiency is 0.812 for CARGOKV versus 0.660 for the best reference, a 23.0% relative difference. Under bursts the values are 0.779 and 0.582 (33.8%). The corresponding best-reference TPOT inflation is 1.440 versus CARGOKV's 1.220 for L-L, and 1.710 versus 1.330 under bursts. Because every method uses its own denominator, these quantities measure scaling degradation.

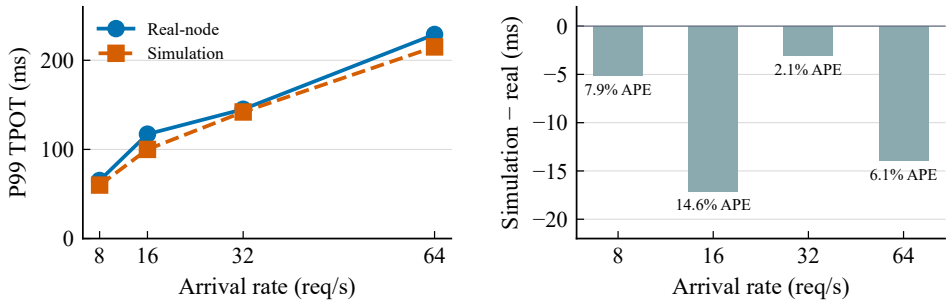

Figure 7: Existing single-pool calibration: real and simulated P99 TPOT at 8, 16, 32, and 64 requests/s (left); signed error and absolute percentage error (right). MAPE is 7.69%. These rate settings belong to calibration and should not be substituted for the closed-loop concurrency axes in the main matrix.

For each calibration point, the signed error is $\Delta = \mathrm{Sim} - \mathrm{Real}$ in ms/token, and $\mathrm{APE} = |\mathrm{Sim} - \mathrm{Real}|/\mathrm{Real} \times 100\%$. MAPE is the arithmetic mean of APE across the four points. The calibration checks tail-latency magnitudes at those points, with 2.1–14.6% individual absolute errors.

C CONTROLLER SENSITIVITY

The sensitivity study varies one parameter at a time on the BurstGPT-derived workload and normalizes results to the default controller. Epochs of 25/200 ms retain four actions per epoch, giving issuance ceilings of 160/20 actions/s. The action-budget and in-flight sweeps use 2/8 actions and 1/4 operations. Narrow/wide hysteresis uses $(\theta_{\mathrm{hi}}, \theta_{\mathrm{lo}}) = (0.825, 0.775)/(0.90, 0.70)$; strong/weak backoff uses $\gamma = 0.50/1.00$ with $\tau = 0.10$ fixed. The epoch variations change P99 TPOT by 1–6%, suggesting

Table 4: Controller sensitivity under the burst workload, normalized to the default configuration. Each pair follows the order of the settings in its row.

Parameter	Tested settings	P99 TPOT	SLO-goodput
Epoch	25 / 200 ms	1.01 / 1.06	.99 / .96
Watermarks	(.825, .775) / (.90, .70)	1.08 / 1.04	.95 / .97
Action budget	2 / 8 actions	1.05 / 1.07	.96 / .95
In-flight cap	1 / 4 operations	1.03 / 1.05	.98 / .97
Backoff	$\gamma = 0.50$ / 1.00	1.03 / 1.06	.97 / .96

that the reported behavior is not tied to a single control timescale. Both smaller and larger action budgets reduce goodput relative to the default, consistent with a balance between recovering capacity and disrupting decoding.

D RUNTIME EVIDENCE: BREAKDOWN, LOAD SWEEP, AND TIMELINE

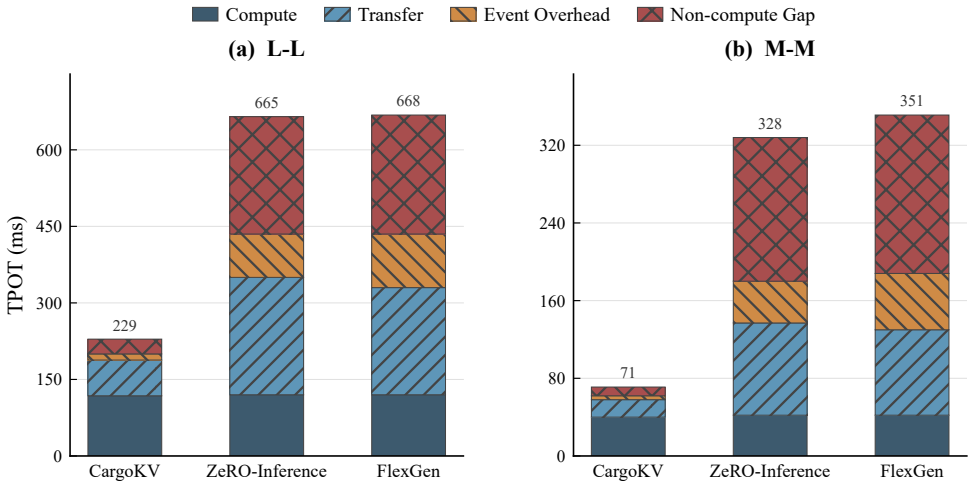

Figure 8: Qwen2.5-72B critical-path breakdown at heavy L-L and M-M operating points. Useful compute, exposed transfer, event overhead, and non-compute gaps are separated. Bar labels round the totals to milliseconds.

The L-L total is 228.98 ms for CARGOKV, 665 ms for ZeRO-Inference, and 668.20 ms for FlexGen. Similar compute contributions (118–120 ms) show that the large differences lie outside useful computation. The reduction comes from both lower exposed transfer time and lower event-plus-gap overhead. M-M provides a second operating point: totals are 71, 328, and 351.154 ms, respectively; CARGOKV's compute/transfer/event/gap components are 40/18/4/9 ms.

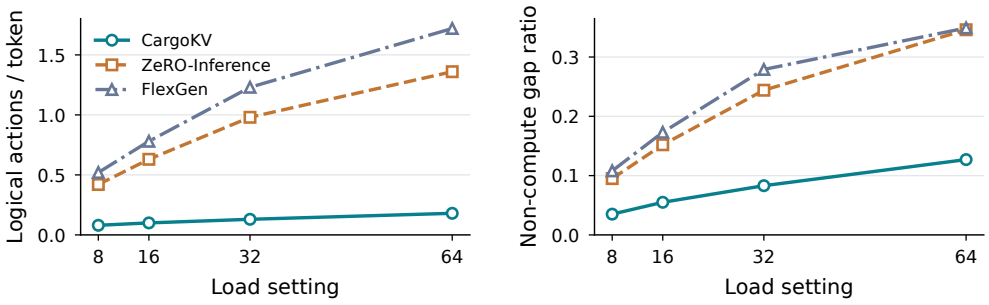

Figure 9: Qwen2.5-72B L-L load sweep: logical migration actions per output token (left) and non-compute gap ratio (right). A request action may comprise multiple physical copies; the two quantities are aggregate runtime signals.

Across the four recorded loads, CARGOKV's action count increases from 0.08 to 0.18 per token, versus 0.42 to 1.36 and 0.52 to 1.72. Its gap ratio increases from 0.035 to 0.127, versus 0.095 to 0.346 and 0.108 to 0.349. Thus the direction holds over the sweep, not just in the selected breakdown point. The two signals co-vary with load.

The selected windows give a temporal view of the same phenomenon: CARGOKV keeps longer intervals of useful decoding, while the conventional configurations spend more time in gaps. The

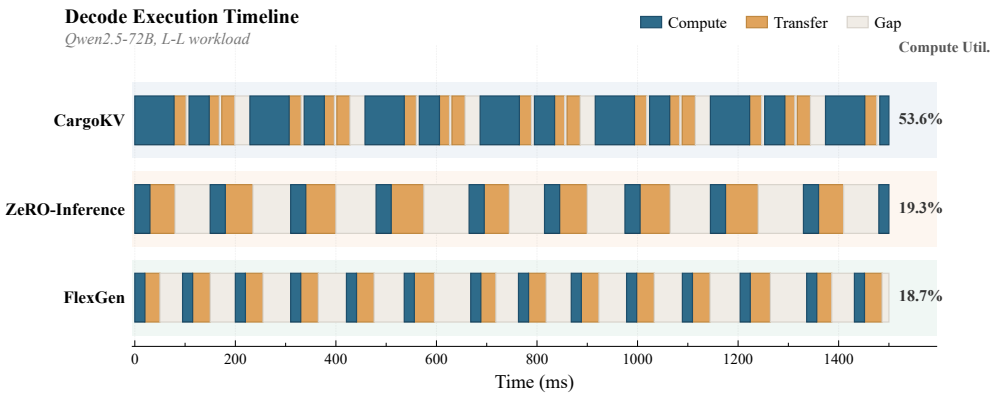

Figure 10: Representative Qwen2.5-72B L-L decode windows. Colored segments depict compute, transfer, and gaps. The displayed compute fractions summarize these selected windows; they are not GPU-wide utilization or population-level averages.

displayed compute fractions are 53.6%, 19.3%, and 18.7%. They must not be equated with the breakdown fractions, which use a different aggregation. These timelines illustrate execution organization and complement the load sweep.

E CONTROLLER SELECTION POLICY

Algorithm 1 starts at each epoch boundary; completion callbacks resume a paused selection loop within that epoch. M is allocated KV memory, Q is reserved reload space, P is the footprint of in-flight offloads, and N counts both migration directions. Eligible sets exclude in-flight requests. Issue operations update these counters atomically; completion releases the slot, commits the memory transition, and resumes the loop without resetting the epoch's issued-action count n. Reload order uses the remaining admission-time generation budget $\widehat{r}(r)$.

Algorithm 1 CARGOKV asynchronous selection with an epoch budget

Require: Eligible resident $\mathcal{A}$, offloaded $\mathcal{O}$, footprints $s(r)$, M, Q, P, N
if a new epoch begins **then**
  $n \leftarrow 0$; $b \leftarrow B_{\max}$
  If $\mathcal{B} > \tau$, set $b \leftarrow \lfloor \gamma B_{\max} \rfloor$
end if
$\widehat{M} \leftarrow M + Q$
if $\widehat{M} > \theta_{\mathrm{hi}} M_{\mathrm{cap}}$ **then**
  $\Delta \leftarrow \widehat{M} - \theta_{\mathrm{lo}} M_{\mathrm{cap}}$; $R \leftarrow P$
  for r in $\mathcal{A}$ sorted by decreasing $s(r)$ **do**
    if $R \geq \Delta$ or $n \geq b$ **then**
      break
    end if
    If $N \geq K_{\max}$, suspend until a completion opens a slot
    IssueOffload(r); $R \leftarrow R + s(r)$; $n \leftarrow n + 1$
  end for
else if $\widehat{M} < \theta_{\mathrm{lo}} M_{\mathrm{cap}}$ **then**
  for r in $\mathcal{O}$ sorted by increasing $\widehat{r}(r)$ **do**
    if $n \geq b$ **then**
      break
    end if
    If $N \geq K_{\max}$, suspend until a completion opens a slot
    $\widehat{M} \leftarrow M + Q$
    if $\widehat{M} + s(r) > \theta_{\mathrm{lo}} M_{\mathrm{cap}}$ **then**
      break
    end if
    IssueReload(r); $\widehat{M} \leftarrow \widehat{M} + s(r)$; $n \leftarrow n + 1$
  end for
end if

F LOCAL CAPACITY-RECOVERY PROPERTY AND ITS LIMITS

Let eligible request sizes be $s_1 \geq s_2 \geq \cdots \geq s_n > 0$. If a feasible prefix exists, define $k^\star = \min\{k : \sum_{i=1}^{k} s_i \geq \Delta\}$. Every subset of $k^\star - 1$ requests has total size at most $\sum_{i=1}^{k^\star - 1} s_i < \Delta$, proving minimum cardinality at this snapshot. Also,

$$0 \leq \sum_{i=1}^{k^\star} s_i - \Delta < s_{k^\star}.$$

This bound is a statement about planned bytes and equal-cost logical requests. An in-flight limit pauses issuance until a completion callback opens a slot; the remaining epoch budget then permits further selection. A deficit left after budget exhaustion carries into the next epoch. Planned offload bytes become free capacity only upon completion.