BOLT-KV: RETHINKING KV MOVEMENT IN CONFIDENTIAL LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) serving moves key-value (KV) caches between GPU and CPU memory to support long contexts and concurrent requests. While one request's cache moves in the background, other requests rely on small CPU–GPU transfers to continue decoding. Confidential computing (CC) protects data during execution and CPU–GPU communication, but can reduce transfer concurrency. On an H100 GPU with CC enabled, background KV movement reduces vLLM decoding throughput by roughly 30% relative to no movement, while the same background transfer workload leaves a non-CC comparison system near baseline. We identify two CC-associated interference modes: CUDA's legacy default stream can delay otherwise unrelated foreground operations, and bulk transfers can delay small transfers in the same direction even on separate streams. Foreground delay closely follows the time remaining until the active transfer completes, and stream isolation can shift the bottleneck to output readback. We present **BOLT-KV**, which removes avoidable legacy-stream coupling and divides selected KV movement into separately submitted, byte-bounded chunks. The resulting boundaries create opportunities for foreground work while preserving transfer order and completion dependencies. During GPU-to-CPU KV movement, BOLT-KV raises SGLang throughput from 78.5% to 99.7% of its no-movement baseline and reduces p99 inter-token latency from 101 to 8 ms. On vLLM's KV-store backend, throughput rises from 80.1% to 88.5% of its no-movement baseline, while p99 latency falls from 74 to 41 ms. BOLT-KV also reduces foreground disruption during the CPU-to-GPU KV restoration transfers, allowing protected KV movement to coexist more effectively with latency-sensitive confidential serving.

1 INTRODUCTION

Large language model (LLM) serving is increasingly constrained by key-value (KV) cache capacity, motivating systems that extend GPU memory with host memory, storage, or remote tiers (Kwon et al., 2023; Sheng et al., 2023; Qin et al., 2025a; Liu et al., 2025). KV tiering moves state in both directions: out of GPU memory when a request's cache is evicted, and back when it is restored, while other requests continue decoding.

Consider saving one request's KV cache to CPU memory while other requests generate tokens. The bulk cache transfer is optional to those requests' current decode steps, whereas they still require small readbacks from GPU memory to CPU memory (device-to-host, D2H) to produce their next outputs. Conventional serving therefore attempts to keep KV movement from delaying decoding by issuing it asynchronously and overlapping it with computation.

Confidential computing (CC) protects sensitive LLM inputs and state on shared cloud infrastructure, including CPU–GPU communication, but can make transfers more expensive and less concurrent, and cause transfer calls to block the issuing CPU thread (Park et al., 2026; Yin & Wang, 2026; Asad & Grunseid, 2026). On an H100 GPU with CC enabled, background D2H KV transfers reduce vLLM decoding throughput by roughly 30%, while the same background transfer workload leaves throughput near baseline on a comparison system without CC (NonCC). The question is where this interference enters the serving path and whether stream isolation is sufficient to remove it.

Our experimental study directly measures bulk transfers and small foreground operations without the serving software stack, revealing two distinct interference modes (Figure 1). First, CUDA's

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.

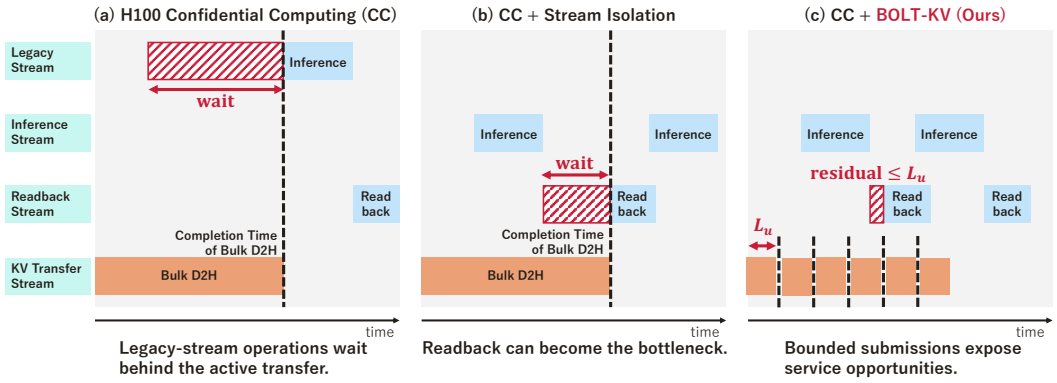


Figure 1: **Interference from background KV movement and BOLT-KV's bounded submissions.** (a) An active bulk D2H transfer blocks legacy-stream operations. (b) Stream isolation removes this coupling, but mandatory D2H readback can still wait behind a same-direction transfer. (c) BOLT-KV introduces boundaries between submissions, creating opportunities for readback to proceed without explicitly prioritizing foreground work; servicing readback delays subsequent KV movement.

legacy default stream can implicitly synchronize with blocking streams, causing otherwise unrelated foreground operations to wait behind an active transfer; we call this *legacy-stream-dependent interference* (Figure 1(a)). Second, even after inference is moved to an explicit non-blocking stream, a small D2H readback can still wait behind a bulk D2H KV transfer. We call this *same-direction occupation interference* (Figure 1(b)), and observe the symmetric effect during restoration, where bulk host-to-device (H2D) KV movement delays foreground H2D input transfers.

When a small foreground operation is delayed by an active bulk transfer, its added latency is approximately equal to the time remaining until that transfer completes. By contrast, small copies in the opposite direction on an explicit non-blocking stream do not show this pattern. In vLLM, when inference runs on an explicit non-blocking stream instead of the legacy stream, most of the movement-induced delay before inference disappears. The main bottleneck then shifts to the mandatory output readback. Together, these results indicate that foreground delay is tied to how long the current transfer remains active, rather than to aggregate transfer bandwidth.

We present **BOLT-KV** (*Boundary-Oriented Low-latency Transfers for KV Cache Movement*). BOLT-KV removes avoidable legacy-stream coupling when needed. It also divides selected KV movement into separately submitted chunks under a byte ceiling, which we call *bounded submissions*. As illustrated in Figure 1(c), BOLT-KV submits the next chunk only after the current chunk reaches a backend-specific issuance boundary. This creates opportunities for foreground work to proceed between chunks. BOLT-KV preserves transfer order and completion dependencies without deliberate idling or explicit foreground prioritization. It changes how selected KV movement executes, not whether state is preserved, restored, or recomputed.

We implement BOLT-KV in vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024). During D2H eviction, BOLT-KV improves serving throughput and reduces p99 inter-token latency (ITL) from 101 to 8 ms on SGLang and from 74 to 41 ms on vLLM's KV-store backend. We separately evaluate the H2D transfer component of restoring an 8K-token prefix while unrelated requests continue decoding. Compared with restoring the same KV cache in one coarse submission, BOLT-KV more than doubles foreground progress during recovery and substantially reduces tail latency. Recomputing the prefix KV cache (Reprefill) completes recovery sooner at this operating point, while BOLT-KV Restore better preserves foreground responsiveness, exposing a trade-off between recovery completion and disruption to concurrent serving.

Contributions. Our contributions are summarized as follows:

- **Problem discovery and characterization.** We identify two CC-associated interference modes in KV tiering transfers: legacy-stream-dependent and same-direction occupation interference. Foreground delay tracks the remaining duration of active movement, and stream isolation can shift the bottleneck downstream.

- **BOLT-KV.** We combine stream isolation with bounded submissions issued across backend-specific issuance boundaries. BOLT-KV preserves the selected KV data, transfer order, and logical completion dependencies without deliberate idling or a foreground-priority gate.
- **Coexistence of KV tiering transfers and latency-sensitive confidential serving.** BOLT-KV reduces interference from both D2H eviction and H2D restoration. During the measured 8K-token recovery, Reprefill completes sooner, whereas BOLT-KV Restore substantially reduces foreground tail latency and more than doubles foreground progress relative to one coarse restore.

More broadly, our results show that protected data movement should be treated as a first-class systems concern in confidential LLM serving, rather than as a background implementation detail. Controlling transfer granularity and issuance boundaries provides a practical path toward confidential memory hierarchies that extend GPU memory while preserving latency-sensitive serving.

2 Background and Related Work

KV Movement in LLM Serving. KV-cache capacity limits LLM serving (Kwon et al., 2023), motivating host-memory, storage, and remote tiers (Sheng et al., 2023; Qin et al., 2025a; Liu et al., 2025), as well as work on offloaded attention and retrieval (Hao et al., 2025; Mao et al., 2026; Wu et al., 2026; Liu et al., 2026), eviction and compression (Qin et al., 2025b; Staniszewski & Łańcucki, 2026), cache reuse and placement (Zheng et al., 2024; Yuan et al., 2026), benchmarking (Li et al., 2025), sparse offloading (Zhu et al., 2026) and MoE scheduling (Li et al., 2026). BOLT-KV instead studies interference caused by executing already-selected KV transfers under confidential GPU execution, without changing cache selection or the restore-versus-recompute policy.

Confidential GPU Execution. Prior work reports higher transfer cost, reduced concurrency, and host-side blocking for protected CPU–GPU communication (Park et al., 2026). Asad & Grunseid (2026) also evaluate SGLang with a CC-aware asynchronous D2H readback worker. Yin & Wang (2026) characterize this path as a serialized, high-setup-cost channel and study CC-aware scheduling and offload policies; PipeLLM instead overlaps encryption with computation (Tan et al., 2025). BOLT-KV addresses a complementary problem: how an already-selected bulk KV transfer delays foreground work, and how partitioning that transfer reduces the interference.

Interference and Granularity. CUDA streams order operations without guaranteeing concurrency, and asynchronous APIs may still block the host (NVIDIA Corporation, 2025a;b). Strait models input-transfer waiting but omits downstream-transfer interference for small outputs in its target workloads (Zhao & Georgantas, 2026); we show that small readbacks can remain a bottleneck behind protected bulk transfers. Sarathi-Serve chunks prefill computation (Agrawal et al., 2024), while ChunkFlow chunks H2D weight prefetching and explicitly yields to collectives at chunk boundaries (Meng et al., 2026). BOLT-KV instead combines stream isolation with bounded submissions issued across backend-specific issuance boundaries, without a foreground-detecting gate.

3 Problem Discovery and Formulation

This section aims to identify where optional KV movement interferes with latency-critical foreground execution under CC, what property of active movement predicts the resulting delay, and whether stream isolation removes or relocates the bottleneck.

KV-tiering systems optimize movement beyond GPU memory (Qin et al., 2025a; Liu et al., 2025), while GPU-CC studies expose serialized and host-blocking transfers (Park et al., 2026; Yin & Wang, 2026). What remains unclear is how these effects interact on the serving critical path. Using end-to-end serving and framework-independent probes, we identify two CC-associated interference modes with a shared residual-occupation signature.

3.1 CC-Associated Slowdown

We run vLLM (Kwon et al., 2023) serving Qwen3-8B[1] with four decoding requests and a fixed background D2H trace representative of KV eviction. Let $R = \Theta_M/\Theta_0$ denote serving throughput

[1] https://huggingface.co/Qwen/Qwen3-8B

(a) Small operations during bulk D2H.

Probe	Stream	p99 (ms)
H2D copy	non-blocking	**0.142**
H2D copy	legacy	88.44
Kernel launch	legacy	88.42
D2H copy	explicit	88.36

(b) Stream isolation shifts exposure in vLLM.

Metric	Stock	PBSI
Input-side excess (ms/iter)	0.960	**0.030**
Readback excess (ms/iter)	0.559	2.567

Table 1: Stream-dependent and same-direction interference are distinct; stream isolation reveals a downstream readback bottleneck. **(a)** Active host-call p99 in framework-independent probes. **(b)** Mean movement-induced host-visible excess at selected vLLM boundaries relative to matched no-movement controls. Rows in (b) are diagnostic, not additive components of iteration latency.

relative to the corresponding no-movement baseline. Under the same offered trace, CC throughput falls to $R = 0.701$, whereas a NonCC comparison remains at $R = 1.001$. Even under a pre-registered $4.5\times$-pressure NonCC control, throughput remains at $R = 0.981$ and active input-H2D p99 at 0.24 ms (Appendix A.2). The reported D2H host-call occupancy (the fraction of the window spent inside movement host calls) is 0.583 on CC and 0.022 on NonCC, despite identical offered work. During active D2H movement, foreground input-H2D p99 is 122.5 ms on CC but only 0.294 ms on NonCC[2]. These differences motivate framework-independent probes of where the delay enters foreground execution.

3.2 Two Interference Modes

The legacy default stream implicitly synchronizes with blocking streams; an explicit non-blocking stream avoids this coupling. We remove the serving framework and place a bulk mover and small victim operations in one CUDA context. Victims are H2D copies, D2H copies, and trivial kernel launches with no dependency on the moved data. We vary stream placement and measure host-call latency for operations issued while bulk movement is active. Table 1 summarizes the D2H-mover results; additional configurations appear in Appendix A.

Mode I: legacy-stream-dependent interference. During bulk D2H movement, H2D copies and kernel launches on the legacy CUDA stream incur approximately 88 ms p99 latency. Moving the H2D victim to an explicit non-blocking stream removes this stall. The affected operation therefore need not itself be D2H: legacy-stream coupling provides one route from protected movement into foreground execution.

Mode II: same-direction occupation interference. A small D2H copy remains delayed even on its own explicit stream. For movement unit u, let $[a_u, c_u)$ denote its recorded interval from submission entry to observed completion. For a foreground operation issued at t, define its remaining runtime-visible movement occupation as

$$\rho_u(t) = \begin{cases} c_u - t, & a_u \le t < c_u, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

For victims whose source is ready at issue, we model excess host-call latency as

$$X_o(t) = \beta_\kappa + \alpha_\kappa \rho_u(t) + \varepsilon_o(t), \tag{2}$$

where κ identifies the operation, direction, stream, and configuration. For explicit-stream D2H victims, $\alpha = 1.001$ with $R^2 \simeq 1$: across the tested victim sizes, excess latency closely tracks the remaining active D2H movement. Reversing the mover reverses the affected direction: explicit-stream H2D again gives $\alpha = 1.001$ and $R^2 \simeq 1$, whereas explicit-stream D2H shows no such dependence. Thus, Mode II is direction-specific: D2H KV storage can delay output readback, while H2D restoration can delay foreground input transfer.

Mode I and Mode II arise through different runtime conditions but share the same residual-occupation signature.

[2]The CC host uses driver 595.71.05 and the NonCC host 610.43.02; VM configurations also differ. We therefore describe the cross-system contrast as CC-associated rather than attributing it solely to enabling CC.

3.3 ISOLATION CAN SHIFT THE BOTTLENECK

To test whether Mode I alone explains the serving slowdown, we apply *Protection-Boundary-Aware Stream Isolation (PBSI)*. The inference path through sampling moves to one persistent explicit non-blocking stream, while output readback retains its existing stream and dependency on inference completion; the movement workload is unchanged.

Table 1b shows that input-side excess falls from 0.960 to 0.030 ms per iteration, while excess at the mandatory output-readback boundary rises from 0.559 to 2.567 ms. Thus, removing legacy-stream exposure can reveal a downstream same-direction bottleneck. These measurements show where delay is observed along the runtime path, but do not provide an additive causal decomposition of the total slowdown among runtime or underlying CC components.

A conditional residual-occupation model explains this pattern. If an upstream saving of δ advances a data-ready downstream operation by δ while the same movement interval remains active and its exposure coefficient remains $\alpha \simeq 1$, then the downstream wait increases by approximately δ. Appendix A.5 states these conditions formally. The factorial experiment in Section 5.1 tests the corresponding system-level consequence: stream isolation and bounded submissions address complementary sources of interference.

The key implication is that stream isolation changes one route of exposure without partitioning the active movement interval. This motivates movement-unit duration as a separate control lever.

3.4 PREDICTIONS

Suppose a logical movement job is divided into separately issued units. For an operation arriving during unit u,

$$0 < \rho_u(t) \leq L_u, \qquad L_u = c_u - a_u. \tag{3}$$

This bounds only the residual of the currently active unit, not total foreground waiting across subsequent units. Whether shorter units improve foreground latency depends on what the runtime services at their boundaries. We therefore make three empirical predictions.

P1 (Logical-job signature collapse). Bounded submissions should break the one-for-one dependence of foreground delay on the remaining duration of the full logical movement job. Any residual dependence should shift toward the scale of the active bounded unit (Section 5.1).

P2 (Complementarity). Stream isolation removes Mode I but leaves same-direction interference; bounded submissions shorten individual movement units but leave legacy-stream coupling intact. When both modes are present, their combination should exhibit a positive interaction, improving serving by more than the sum of their separate effects (Section 5.1).

P3 (Bounded bytes, unbounded tail). A byte ceiling limits submitted bytes per unit, not foreground waiting time, and service at the next boundary is not guaranteed. Tail latency should fall substantially while rare long waits may persist; no deterministic ITL bound follows (Section 5.2).

4 BOLT-KV

This section develops BOLT-KV to reduce foreground interference from already-selected KV movement under CC without changing the surrounding KV lifecycle policy. Guided by Section 3, BOLT-KV combines stream isolation to remove legacy-stream coupling (Mode I) with bounded submissions to shorten individual movement units (Mode II). It changes how selected KV movement executes while preserving the selected data, transfer order, and logical completion dependencies.

4.1 DESIGN PRINCIPLE: CONTROL THE ACTIVE SUBMISSION UNIT

On the measured confidential-transfer paths, the protected host call spans nearly the entire recorded movement interval (Appendix A.1). Dividing one API call into multiple internal descriptors does not create an additional API-return boundary for the mover. BOLT-KV therefore separates a logical movement job into bounded submissions, issuing each next unit only after its predecessor reaches the backend-specific issuance boundary defined in Section 4.3.

4.2 Stream Isolation

When latency-critical inference uses the legacy CUDA stream, BOLT-KV moves that path to one persistent explicit non-blocking stream. In vLLM, the path from input H2D through CUDA-graph replay and sampling uses this stream, while output readback retains its existing copy stream and dependency on inference completion.

This removes the legacy-stream route of Mode I without changing the readback operation itself. The stream is created before warmup and graph capture, and no additional synchronization is inserted between input transfer and computation. SGLang already uses explicit streams on the measured path, so no additional stream isolation is required there; its existing CC-specific readback worker is left unchanged. Implementation details are given in Appendix B.1.

4.3 Bounded Submissions and Issuance Boundaries

For each selected KV job, BOLT-KV partitions movement into ordered submissions under a byte ceiling $q_{\max}$ whenever backend granularity permits. Indivisible objects remain intact even if one exceeds the ceiling. The selected bytes, logical order, and completion dependencies are preserved.

BOLT-KV does not require a universal synchronization primitive. Instead, each backend defines an *issuance boundary* at which the next movement unit can be submitted. The mover is work-conserving: it issues the next unit as soon as the preceding one reaches this boundary, without deliberate idling or a foreground-priority gate. The concrete realization of this boundary differs across runtimes.

An issuance boundary determines when the next movement unit may be submitted; it does not redefine logical-job completion, which remains governed by the backend's existing completion semantics. These boundaries create opportunities for foreground service, but do not guarantee that foreground work is selected next.

4.4 Instantiation

We instantiate BOLT-KV in SGLang and vLLM; detailed issuance and completion semantics are given in Appendix B.2.

SGLang. SGLang (Zheng et al., 2024) already uses explicit streams on the measured path, so only bounded byte-addressable submissions are added. The mover advances at protected D2H host-call return; at 4 MiB, host-call p50 is 1.052 ms versus 1.056 ms including completion, supporting this issuance boundary.

vLLM. Controlled replay in vLLM (Kwon et al., 2023) waits for unit completion, whereas its KV-store backend advances after each protected `cuMemcpyBatchAsync` return while retaining its per-job completion event. For Qwen3-8B, indivisible 2.25 MiB KV blocks make the 4 MiB setting one block per submission; H2D restoration uses the same block-granular policy in reverse.

Design implication. BOLT-KV accommodates different backend semantics, providing a path toward confidential memory hierarchies that preserve foreground responsiveness. Issuance boundaries create service opportunities, not scheduling guarantees.

5 Empirical Evaluation

Section 3 established the interference characterization. Here we test P1 and P2 and the BOLT-KV mechanism (Section 5.1), measure end-to-end serving performance and P3 (Section 5.2), and evaluate H2D restoration under concurrent decoding (Section 5.3).

Setup. We evaluate the SGLang and vLLM instantiations from Section 4.4 on an NVIDIA H100 NVL with GPU confidential computing enabled. Unless otherwise stated, experiments use Qwen3-8B in BF16, and results use 20 independent launches per configuration, with percentiles computed within launch before across-launch aggregation. The headline experiments use $q_{\max} = 4\,\mathrm{MiB}$, fixed before serving measurements; Appendix B.3 reports a separate calibration audit, and Section 5.2 evaluates sensitivity to submission size.

Table 2: BOLT-KV collapses the logical-job occupation signature at SGLang's mandatory readback (across-launch mean ± SD). Slope and R^2 measure dependence of readback excess on the remaining duration of the logical movement job.

	Unbounded	**BOLT-KV**
Slope vs. remaining job	1.006 ± 0.008	$\mathbf{0.0021 \pm 0.0004}$
R^2	0.989 ± 0.027	$\mathbf{0.039 \pm 0.015}$
Active readback p99 (ms)	83.8 ± 19.0	$\mathbf{1.51 \pm 0.03}$

Table 3: Complementarity of stream isolation and bounded submissions on vLLM controlled replay. R: mean [95% CI]; no-movement ITL p99 is 8.2 ms. Max ITL: mean of per-launch maxima.

Inference	**Movement**	R	**ITL p99**	**Max ITL**
Legacy	Unbounded	0.695 [0.690, 0.699]	20.3 ms	852 ms
Legacy	Bounded	0.724 [0.721, 0.727]	36.1 ms	130 ms
PBSI	Unbounded	0.714 [0.711, 0.717]	131.0 ms	244 ms
PBSI	Bounded	**0.827** [0.822, 0.830]	**12.1** ms	**108** ms

5.1 Mechanism-Level Validation

We first test whether BOLT-KV changes the interference signatures identified in Section 3 and whether its two components address complementary mechanisms, corresponding to P1 and P2.

P1: logical-job signature collapse. P1 predicts that bounded submissions should collapse the logical-job-scale dependence identified in Section 3. Table 2 tests this prediction at SGLang's mandatory readback. With unbounded movement, readback excess has slope 1.006 against the remaining logical-job duration ($R^2 = 0.989$). With BOLT-KV, the slope falls to 0.002 ($R^2 = 0.039$), while active readback p99 falls from 83.8 to 1.51 ms. Residual dependence remains at the active-unit scale (slope 0.733, $R^2 = 0.44$), indicating that bounding shifts rather than eliminates movement interference. The reverse direction shows the same pattern (Figure 2a). With total movement bytes fixed, H2D victim p99 falls from 89 ms unbounded to 3.1, 2.14, 1.12, and 0.67 ms for 16, 8, 4, and 2 MiB submissions, respectively. Its slope against the remaining logical job falls from 1.001 to at most 0.004 (Appendix A.3).

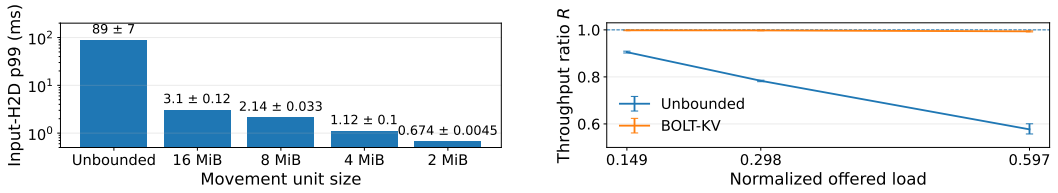
P2: stream isolation and bounded submissions are complementary. Table 3 crosses legacy/PBSI inference with unbounded/bounded movement on vLLM controlled replay. From the legacy/unbounded baseline, stream isolation alone increases R by 0.019 and bounded submissions alone by 0.029, whereas their combination increases R by 0.132, giving a positive interaction of approximately $+0.08$. The latency results show why. Stream isolation removes Mode I but leaves readback exposed to same-direction movement, increasing ITL p99 from 20.3 to 131.0 ms. Bounding shortens active movement units but leaves the legacy-stream path open. Only their combination addresses both modes, reducing ITL p99 to 12.1 ms versus 8.2 ms without movement.

Takeaway. BOLT-KV changes the scale and path of movement interference: bounded submissions collapse logical-job dependence toward the active-unit scale, while stream isolation removes a complementary legacy-stream path. Their combination addresses both interference modes.

5.2 End-to-End Serving Under D2H Movement

We next test whether the mechanism-level effects translate into end-to-end serving. SGLang uses controlled replay in which only optional movement execution changes, whereas vLLM exercises its KV-store backend under the runtime's existing KV lifecycle policy.

Table 4 shows substantial improvements in both runtimes. On SGLang, BOLT-KV raises R from 0.785 to 0.997 (95% CI $[0.995, 0.999]$) and reduces ITL p99 from 101.4 to 8.0 ms. On vLLM's KV-store backend, R increases from 0.801 to 0.885 and ITL p99 falls from 73.7 to 41.4 ms; the paired BOLT-KV/Stock throughput ratio is 1.106 ($[1.098, 1.112]$).



(a) H2D victim latency versus submission size.

(b) SGLang under increasing offered movement load.

Figure 2: **(a)** Smaller bounded submissions sharply reduce H2D victim p99 under fixed total movement bytes. **(b)** BOLT-KV sustains near-baseline foreground throughput as movement pressure increases, while unbounded movement degrades serving; throughput is normalized to the matched no-movement control and the 4 MiB operating point is fixed. Error bars and ± values denote across-launch mean ± SD of per-launch statistics.

Table 4: BOLT-KV improves serving throughput and tail latency under D2H movement at the 4 MiB operating point. SGLang uses controlled replay; vLLM uses its KV-store backend. Each runtime is compared with its own matched control. Max ITL: mean of per-launch maxima.

	SGLang		vLLM	
Metric	**Stock / Unbounded**	**BOLT-KV**	**Stock / Unbounded**	**BOLT-KV**
Throughput ratio R	0.785	**0.997**	0.801	**0.885**
ITL p99	101.4 ms	**8.0 ms**	73.7 ms	**41.4 ms**
Max ITL	217.5 ms	**28.9 ms**	280.2 ms	**69.2 ms**

The result is not specific to the 4 MiB operating point. A separate vLLM sweep finds lower ITL p99 at every tested bounded size (2.25–18 MiB), while throughput among bounded configurations decreases from $R = 0.801$ at 2.25 MiB to 0.733 at 18 MiB (Appendix C.3).

We also reproduce the throughput and ITL improvements on Llama-3.1-Nemotron-Nano-8B, whose different KV-block geometry lets the 4 MiB ceiling group two 2 MiB blocks per submission, indicating that the benefit is not specific to Qwen3-8B's single-block granularity (Appendix C.4).

Movement work and pressure. On SGLang, BOLT-KV completes essentially the same amount of movement as the unbounded baseline and, at high offered load, spends more time inside movement host calls. Despite this, its serving advantage persists as movement pressure increases (Figure 2b; Appendix C.2). The vLLM KV-store backend lacks equivalent completed-byte instrumentation, so we do not make the same accounting claim there.

P3: bounded submissions reduce tails but do not bound them. BOLT-KV reduces ITL p99 by 12.6× on SGLang and 1.8× on vLLM, but rare long delays remain. On SGLang, the mean per-launch maximum is 28.9 ms versus a 9.1 ms median because one launch contains an approximately 200 ms excursion; the 4 MiB H2D probe likewise records an approximately 200 ms maximum despite p99 1.12 ms. At the highest offered load, BOLT-KV movement lateness reaches p99 2.26 s and some launches retain backlog (Appendix C.2).

Takeaway. BOLT-KV improves end-to-end serving across SGLang and vLLM, with the benefit persisting under higher movement pressure. On SGLang, this occurs with essentially unchanged completed movement work, supporting partitioning rather than reduced movement as the explanation. Bounded submissions reduce tail latency but do not provide a deterministic latency bound.

5.3 H2D Restoration Under Concurrent Serving

We compare three recovery methods. *Reprefill* recomputes the KV cache of an 8K-token prefix with prefix caching disabled, using vLLM's default chunked prefill with an 8192-token budget. *Stock Restore* transfers the preserved KV cache in one coarse H2D submission, whereas *BOLT-KV Restore* transfers the same bytes using bounded submissions. Both restore arms write identical prepared KV bytes to a validated GPU scratch destination, so the experiment measures the transfer component of restoration rather than resumed attention.

Table 5: Recovery under concurrent serving. BOLT-KV reduces foreground disruption while exposing a trade-off with completion time. Recovery time is the mean; Max ITL is the mean of per-launch maxima over the fixed observation slots.

Metric	Reprefill	Stock Restore	BOLT-KV Restore
Recovery time (mean)	**457.6 ms**	1049.8 ms	799.5 ms
Progress during recovery	7.6%	9.9%	**22.7%**
Slot retention	**92.8%**	89.1%	89.8%
ITL samples > 100 ms	0.149%	0.149%	**0%**
Max foreground ITL	546.8 ms	652.4 ms	**47.3 ms**

Each launch contains four recovery events six seconds apart. We use 20 independent launches per method plus a no-recovery control. *Progress during recovery* measures foreground tokens produced during recovery relative to the control, while *slot retention* measures foreground tokens over the full fixed observation slots.

Table 5 reveals different rankings depending on the recovery objective.

- **Recovery completion.** Reprefill completes fastest. Median recovery times are 457, 690, and 761 ms for Reprefill, Stock Restore, and BOLT-KV Restore, respectively (Appendix C.5); the larger Stock mean is driven by one approximately 7 s stall. BOLT-KV Restore therefore does not minimize recovery time.
- **Aggregate foreground service.** Reprefill retains 92.8% of foreground service over the fixed slots, compared with 89.1% for Stock Restore and 89.8% for BOLT-KV Restore.
- **Foreground disruption during recovery.** BOLT-KV Restore more than doubles foreground progress relative to Stock Restore (22.7% versus 9.9%). It records no ITL above 100 ms and reduces mean launch-maximum ITL to 47.3 ms, compared with 652.4 ms for Stock Restore and 546.8 ms for Reprefill.

Takeaway. Recovery completion and foreground responsiveness are distinct objectives. Reprefill favors completion time and aggregate service, whereas BOLT-KV Restore shifts the trade-off toward foreground responsiveness, with greater progress during recovery and much smaller tail stalls.

Limitations. Our evaluation has several limitations that constrain the scope and generality of the conclusions. We evaluate one H100 NVL CC platform and selected workloads; Blackwell-generation platforms remain untested. The CC and NonCC systems differ in driver and VM configuration, so their contrast is CC-associated rather than causal, and we cannot locate the mechanism inside NVIDIA's closed CC implementation. SGLang uses controlled replay, and restoration is evaluated on scratch memory rather than resumed attention. Finally, bounded submissions reduce interference at the tested loads but do not provide a deterministic latency bound or demonstrate a higher maximum sustainable movement rate.

6 Conclusion

We identify two ways in which background KV movement can delay latency-sensitive serving under confidential computing. Legacy-stream coupling can stall otherwise unrelated operations, while bulk transfers can delay smaller foreground transfers in the same direction even on separate streams. In both cases, the added delay is closely tied to how long the active transfer remains. BOLT-KV addresses these effects by removing avoidable legacy-stream coupling and dividing selected KV movement into byte-bounded submissions. Across SGLang and vLLM, this improves serving throughput and tail latency without changing the surrounding KV lifecycle policy. During CPU-to-GPU KV restoration, BOLT-KV also improves foreground responsiveness, showing that recovery completion time and disruption to concurrent serving are distinct objectives.

More broadly, our results show that protected data movement should be treated as a first-class systems concern in confidential LLM serving. How transfers are divided and when the next transfer is issued directly affect foreground responsiveness. BOLT-KV shows that memory can be extended beyond the GPU without letting background KV movement dominate latency-sensitive serving.

AI USE STATEMENT

Generative AI tools were used for manuscript drafting, language editing, organization, and proofreading, and for feedback on experimental methodology and comparisons and assistance in interpreting and presenting results. All reported experimental measurements were obtained from system executions. The authors reviewed and verified all AI-assisted text, technical claims, experimental results, analyses, and citations, and take responsibility for the final content.

ETHICS STATEMENT

This work studies the performance of confidential LLM serving systems and does not involve human subjects or personal data. The experiments use public language models and synthetic or controlled serving workloads. BOLT-KV changes scheduling within an existing confidential-computing boundary and leaves the protected CPU–GPU transfer path unchanged (Appendix B.4).

REPRODUCIBILITY STATEMENT

Experimental configurations, seeds, and validity criteria for the reported measurements were fixed before data collection. Independent process launches are the statistical units, with uncertainty estimated at the launch level. Appendices A–C document the platforms, workloads, measurement procedures, implementation details, and statistical analyses.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134. USENIX Association, 2024.

Amean Asad and Ansgar Grunseid. Benchmarking confidential computing performance on NVIDIA blackwell GPUs, 2026. URL https://arxiv.org/abs/2608.26575v2.

Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. OmniKV: Dynamic context selection for efficient long-context LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ulCAPXYXfa.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Yan Li, Zhenyu Zhang, Zhengang Wang, Pengfei Chen, and Pengfei Zheng. Semantic parallelism: Redefining efficient moe inference via model-data co-scheduling. In *International Conference on Learning Representations*, 2026.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gkUyYcY1W9.

Guangda Liu, Chengwei Li, Zhenyu Ning, Jing Lin, Yiwu Yao, Danning Ke, Minyi Guo, and Jieru Zhao. FreeKV: Boosting KV cache retrieval for efficient LLM inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=wXAn7orB1H.

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

Yuzhen Mao, Qitong Wang, Martin Ester, and Ke Li. IceCache: Memory-efficient KV-cache management for long-sequence LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=yHxSKM9kdr.

Han Meng, Danny Willow Liu, and Dong Li. ChunkFlow: Communication-aware chunked prefetching for layerwise offloading in distributed diffusion transformer inference. *arXiv preprint arXiv:2605.11335*, 2026.

NVIDIA Corporation. *CUDA C++ Programming Guide*. NVIDIA Corporation, release 13.0 edition, 2025a. URL https://docs.nvidia.com/cuda/archive/13.0.0/cuda-c-programming-guide/index.html.

NVIDIA Corporation. *CUDA Runtime API*. NVIDIA Corporation, 2025b. URL https://docs.nvidia.com/cuda/archive/13.0.0/cuda-runtime-api/index.html. CUDA Toolkit 13.0.0.

Eunseong Park, Timo Thans, Vishnu Kumar Kalidasan, Qinghao Hu, and Wenjie Xiong. An empirical study of LLM serving in confidential GPUs. In *2026 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, pp. 311–322. IEEE, 2026. doi: 10.1109/ISPASS69572.2026.00038.

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, 2025a.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. CAKE: Cascading and adaptive KV cache eviction with layer preferences. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=EQgEMAD4kv.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, 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.

Konrad Staniszewski and Adrian Łańcucki. KV cache transform coding for compact storage in LLM inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2511.01815.

Yifan Tan, Cheng Tan, Zeyu Mi, and Haibo Chen. PipeLLM: Fast and confidential large language model services with speculative pipelined encryption. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 843–857, 2025. doi: 10.1145/3669940.3707224.

Wenbo Wu, Qingyi Si, Xiurui Pan, Ye Wang, and Jie Zhang. LouisKV: Efficient KV cache retrieval for long input-output sequences. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=6RJ8fZwm4P.

Hang Yin and Kevin Wang. The serialized bridge: Understanding and recovering LLM serving performance under blackwell GPU confidential computing. *arXiv preprint arXiv:2606.23969*, 2026.

Ying Yuan, Pengfei Zuo, Bo Wang, Zhangyu Chen, Zhipeng Tan, and Zhou Yu. DualMap: Enabling both cache affinity and load balancing for distributed LLM serving. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zCadrJ32Xn.

Haidong Zhao and Nikolaos Georgantas. Strait: Perceiving priority and interference in ML inference serving. *arXiv preprint arXiv:2604.28175*, 2026.

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 *Advances in Neural Information Processing Systems*, volume 37, 2024.

Zhui Zhu, Weichen Zhang, Zhenghan Zhou, Yunhao Liu, and Fan Dang. Dynamicinfer: Runtime-aware sparse offloading for llms inference on a consumer-grade gpu. In *International Conference on Learning Representations*, 2026.

APPENDIX

A MECHANISM CHARACTERIZATION DETAILS

This appendix gives the measurement conventions, controls, and model supporting Section 3. Each experimental block uses its own workload and reference measurements; absolute latencies are compared within a block.

A.1 MEASUREMENT CONVENTIONS

Timing boundaries. For a movement unit u, a_u is submission entry, r_u is host-call return, and c_u is observed completion. Host-call duration is $r_u - a_u$, post-return drain is $c_u - r_u$, and the runtime-visible duration is $L_u = c_u - a_u$. Observed completion is the completion-wait measurement boundary, not a hardware timestamp for the last transferred byte, and no interval is interpreted as physical Copy Engine ownership.

For the recorded D2H events above 128 MiB, the host-call fraction $(r_u - a_u)/L_u$ is 0.98–1.00. Thus, almost all of the recorded movement interval lies inside the host call in this configuration. This is an empirical property of the measured path, not a general equivalence between host return and device completion; it is consistent with prior observations of host blocking under GPU confidential computing (Yin & Wang, 2026).

Victim latency. Victim latency is measured from host-call entry to return. Completion waits are measured separately and, for explicit-stream victims, restricted to the victim's own stream; no device-wide synchronization is used. A sample is *active* if issued while bulk movement is in progress and *inactive* otherwise. Active-conditioned statistics are kept separate from unconditional and no-movement controls.

Residual-occupation predictor. For a victim issued at t_o during movement unit u, the predictor is $\rho_u(t_o) = c_u - t_o$ (Equation 1). The active unit is identified from mover timestamps at victim issue and is never selected retrospectively from the observed victim delay.

Statistics. The independent process launch is the statistical unit. Percentiles are computed within launch and then summarized across launches as specified by each experiment. Tables below distinguish means, medians, standard deviations, confidence intervals, and pooled descriptive statistics explicitly.

A.2 CC–NONCC COMPARISON

Four requests decode continuously with Qwen3-8B under vLLM while a background mover replays a fixed D2H trace (685 events, 8.0 GiB, fixed ordering and absolute release rule). The primary comparison preserves offered events, byte counts, ordering, and release rule across hosts.

A second NonCC arm was pre-registered to increase movement pressure until the reported D2H host-call occupancy matched the CC value of 0.583. The release-gap scale was selected from a frozen grid using disjoint calibration seeds. The most aggressive grid point reaches occupancy 0.099; the grid was not extended, so this arm is reported as a $4.5\times$-pressure control, not an occupancy-matched comparison. All 80 final launches on each host are valid and none were replaced.

Table 6 shows a large CC-associated contrast in foreground input-H2D latency, including against the $4.5\times$-pressure NonCC control. Active input-H2D p99 remains 0.24–0.29 ms on NonCC versus 122.5 ms on CC at the tested pressures.

The CC and NonCC hosts use drivers 595.71.05 and 610.43.02, respectively, and their VM configurations also differ. We therefore describe the cross-system contrast as CC-associated rather than attributing it solely to enabling CC.

Table 6: CC–NonCC serving comparison. R and ΔT_{iter} are means with 95% CIs over 20 matched launches, each movement arm normalized by its own contemporaneous no-movement control. Reported D2H occupancy is the median across launches of the fraction of the window spent inside movement host calls. Store-active input-H2D p99 is a pooled descriptive statistic (pooled n in parentheses; no launch-level CI).

Metric	CC	NonCC same trace	NonCC $4.5\times$ pressure
Serving throughput ratio R	0.701 [0.697, 0.706]	1.001 [1.000, 1.002]	0.981 [0.981, 0.981]
ΔT_{iter} (ms)	+3.64 [3.58, 3.70]	−0.005 [−0.010, −0.002]	+0.135 [0.132, 0.138]
Reported D2H occupancy	0.583	0.022	0.099
Store-active input-H2D p99 (ms)	122.5 (6291)	0.29 (8081)	0.24 (30787)

Table 7: Bulk D2H mover. Control is the across-launch median of within-launch host-call p50; active is the across-launch median of within-launch host-call p99. Slope and R^2 are from Equation 2 with ρ_u as predictor. n is the number of independent launches retained after pre-defined validity checks.

Victim	Stream	Control p50 (ms)	Active p99 (ms)	Slope	R^2	n
H2D, 4100 B	explicit	0.0466	0.1415	0.0000	0.0006	17
H2D, 4100 B	legacy	0.0218	88.4355	1.0007	1.0000	19
Kernel launch	legacy	0.0248	88.4235	1.0010	1.0000	19
D2H, 4100 B	explicit	0.0578	88.3594	1.0010	1.0000	18
D2H, 1 MiB	explicit	0.3282	90.1685	1.0012	1.0000	18

A.3 Framework-Independent Probes

Two studies place a bulk mover and small victim operations in one process and one primary CUDA context, with no event dependency between mover and victim and no producer–consumer dependency on the moved data. The first uses a bulk D2H mover (Table 7); the second reverses the mover to H2D and includes D2H reference probes (Table 8). Absolute latencies need not match across the two studies because they use separate release schedules.

Tables 7 and 8 show near-unit residual slopes for legacy-stream victims and explicit-stream same-direction copies. Explicit-stream opposite-direction copies have slopes near zero. Reversing the mover reverses the affected copy direction, while increasing the D2H victim size from 4100 bytes to 1 MiB leaves the near-unit slope unchanged.

Stream priority. The backend low-priority H2D configuration is delayed identically to the explicit-stream H2D victim (slope 1.0009, $R^2 = 1.0000$). Changing CUDA stream priority therefore does not mitigate the measured same-direction interference, consistent with the documented behavior that stream priorities affect compute-kernel scheduling but not H2D or D2H transfers (NVIDIA Corporation, 2025b).

Granularity and the limit of a byte ceiling. The H2D sweep in Figure 2a fixes total movement bytes and varies the submission ceiling. For the unbounded mover, the slope against remaining logical-job duration is 1.0009 ($R^2 = 1.0000$); every tested bounded ceiling gives slope ≤ 0.004 and $R^2 \leq 0.06$. Thus, the logical-job-scale signature disappears at every tested ceiling.

A byte ceiling does not, however, provide a latency bound. At 4 MiB, the across-launch mean of per-launch maximum victim latency is 12.8 ms while the median is 1.18 ms, because one of the 17 valid launches records an approximately 200 ms wait. The event is retained rather than trimmed and motivates prediction P3 in Section 3.4.

A.4 PBSI Bottleneck Relocation

PBSI is implemented as described in Appendix B.1. The movement workload is unchanged, and output readback retains its dependency on inference completion. Table 9 reports signed changes

Table 8: Bulk H2D mover, with D2H reference probes from the same study. Columns as in Table 7.

Victim	Stream	Control p50 (ms)	Active p99 (ms)	Slope	R^2	n
Bulk H2D mover						
H2D	legacy	0.0214	91.0159	1.0007	1.0000	19
H2D	explicit	0.0463	90.1310	1.0009	1.0000	18
H2D	backend low-priority	0.0465	90.5097	1.0009	1.0000	19
D2H	legacy	0.0312	90.7039	1.0007	1.0000	20
D2H	explicit	0.0578	0.1467	0.0000	0.0005	17
Kernel	legacy	0.0250	90.9436	1.0008	1.0000	18
Bulk D2H reference probes						
H2D	legacy	0.0217	89.4631	1.0006	1.0000	19
D2H	explicit	0.0582	88.3618	1.0009	1.0000	19

Table 9: PBSI latency decomposition (ms per iteration). Rows are selected host-visible boundaries; the input-side row uses the loop-level input/preparation boundary (`pi`), not the step-level H2D-copy metric. Boundary changes can include waits inherited from predecessors and do not sum to the iteration-time change.

Metric	Stock	PBSI
Mean iteration-time increase	$+3.6445$	$+3.3587$
Movement-induced penalty		
Input-side path	$+0.9604$	$+0.0303$
FULL graph replay	$+0.1201$	$+0.0142$
Sampler	$+0.8284$	$+0.0193$
`AsyncOutput.__init__`	$+0.5589$	$+2.5672$

in mean host-visible time per iteration at selected runtime boundaries relative to the matched no-movement configuration.

PBSI removes most upstream exposure: the input-side, graph-replay, and sampler penalties all decrease substantially. The mean iteration-time penalty, however, changes only from $+3.64$ to $+3.36$ ms, while the output-readback boundary increases from $+0.56$ to $+2.57$ ms. Thus, removing legacy-stream exposure reveals a downstream same-direction bottleneck. This experiment is separate from the factorial evaluation in Section 5.1.

A.5 Conditional Exposure Model

This section states the conditions behind the model in Sections 3.2 and 3.3 and proves the relocation proposition.

Definitions. For movement unit u with recorded interval $[a_u, c_u)$ and a foreground operation o issued at time t, residual occupation is $\rho_u(t)$ from Equation 1. Let $T_o(\mathcal{M})$ and $T_o(\varnothing)$ be the latency of o under optional movement and its matched no-movement reference, measured using the same timing boundary. The observed excess is

$$X_o = T_o(\mathcal{M}) - T_o(\varnothing).$$

For data-ready operations issued during movement, the local model is Equation 2,

$$X_o(t) = \beta_\kappa + \alpha_\kappa \rho_u(t) + \varepsilon_o(t),$$

with $\mathbb{E}[\varepsilon_o \mid \rho_u, \kappa] = 0$. The configuration index κ specifies operation, direction, stream, and measurement configuration.

Readiness adjustment. If an operation has an independent readiness wait g, the idealized incremental delay from movement is

$$\max(g, \rho) - g = [\rho - g]_+, \tag{4}$$

rather than ρ. All probe victims in Appendix A.3 are data-ready at issue ($g = 0$).

Proposition 1 (Conditional bottleneck relocation) *Let p precede q on a foreground dependency path, with q issued immediately upon p's completion. Suppose an intervention removes $\delta > 0$ of p's excess and advances q's issue time from t_q to $t_q - \delta$. Assume both executions use the same no-movement references, q is data-ready at both issue times, its coefficients α_q, β_q are unchanged, and the movement interval $[a_u, c_u)$ is fixed with $a_u \le t_q - \delta < t_q < c_u$. Then*

$$(X'_p + X'_q) - (X_p + X_q) = -(1 - \alpha_q)\delta + \Delta\varepsilon_q, \tag{5}$$

where $\Delta\varepsilon_q = \varepsilon'_q(t_q - \delta) - \varepsilon_q(t_q)$.

Proof 1 *By assumption, $X'_p - X_p = -\delta$. Since q is data-ready and the movement interval is fixed, $\rho_u(t_q - \delta) - \rho_u(t_q) = \delta$. Applying Equation 2 at the two issue times gives $X'_q - X_q = \alpha_q\delta + \Delta\varepsilon_q$. Adding the two changes yields Equation 5.*

If $|\Delta\varepsilon_q| \le \eta$, the change in total excess differs from $-(1-\alpha_q)\delta$ by at most η. Thus, when $\alpha_q \simeq 1$, an upstream saving can reappear as downstream waiting. The result requires the same active movement interval and a data-ready downstream operation. With an independent readiness wait g_q, readiness slack can absorb some or all of the issue-time advance.

Boundary measurements. Host-visible boundary measurements can include waits inherited from predecessors, so the entries in Table 9 are not additive components of iteration latency. The PBSI measurements are consistent with the relocation mechanism but do not supply the paired issue-time conditions needed to test Equation 5 quantitatively. Section 5.1 tests the corresponding system-level complementarity of stream isolation and bounded submissions.

What bounding does and does not bound. Equation 3 bounds the residual of the identified active unit. It does not guarantee service at the next boundary or bound waiting across successive units. A byte ceiling therefore provides no deterministic service-time or ITL bound.

B BOLT-KV Implementation Details

This appendix gives the implementation details supporting Section 4. The key distinction is between the boundary that permits the next movement unit to be issued and the condition that establishes completion of the logical job.

B.1 Stream Isolation

Protection-Boundary-Aware Stream Isolation (PBSI) moves the latency-critical vLLM inference path to one persistent explicit non-blocking CUDA stream S. The path includes input H2D, preparation kernels, model execution including CUDA graph replay, logits, and sampling. Keeping the dependent path on one stream preserves execution order without introducing an intermediate synchronization edge.

Output readback retains its existing copy stream. Its dependency on inference completion is retargeted from the legacy inference stream to S:

$$\texttt{output_copy_stream.wait_stream}(S). \tag{6}$$

The mandatory D2H operation and the inference–readback dependency are otherwise unchanged.

The prototype creates S during worker initialization, before warmup and CUDA-graph capture, so warmup, graph construction, replay, and steady-state inference use the same stream configuration. No PBSI modification is applied to SGLang: the measured path already uses explicit streams, and its existing CC-specific readback worker remains enabled and unmodified.

B.2 Bounded Submissions and Completion

For a logical movement job containing B bytes, BOLT-KV constructs ordered submission units with sizes b_k satisfying

$$\sum_k b_k = B, \qquad b_k \le q_{\max}, \tag{7}$$

Table 10: Backend-specific issuance and completion semantics. Host-call return is an issuance boundary where used, not a device-completion condition.

Implementation	Next-unit issuance boundary	Logical-job completion
vLLM controlled replay	Wait for the current unit to complete on the movement stream.	The final unit's stream completion establishes job completion.
SGLang controlled replay	Advance after the protected D2H host call returns.	Drain the movement stream after the final unit.
vLLM KV-store backend	Advance after the current `cuMemcpyBatchAsync` call returns.	Observe the original per-job completion event recorded after the final submission.
H2D restoration component	Advance after the current protected H2D host call returns.	Wait for completion on the H2D stream after the final submission.

whenever the backend's atomic granularity permits the ceiling. For a byte-addressable mover,

$$b_k = \min(q_{\max}, B - o_k), \qquad o_{k+1} = o_k + b_k, \quad o_1 = 0, \tag{8}$$

until the original range is covered.

For indivisible objects, consecutive objects are grouped without exceeding the ceiling whenever possible. If one object exceeds $q_{\max}$, it remains the minimum feasible unit and is reported as exceeding the configured ceiling. For Qwen3-8B, the evaluated vLLM KV-store backend exposes 2.25 MiB blocks, so a 4 MiB ceiling permits one block per submission. A block can contain multiple backend views and copy descriptors; the ceiling applies to their total submitted bytes.

Partitioning preserves the selected source and destination ranges, logical order, and the completion dependency covering all units of the job. These are per-job properties; movement accounting across serving runs is defined in Appendix C.

Work-conserving issuance. BOLT-KV issues the next unit as soon as its predecessor reaches the backend-specific issuance boundary, without intentional idling or a foreground-priority gate. This is the sense in which the mover is *work-conserving*; movement throughput is measured separately. Algorithm 1 summarizes the policy.

Algorithm 1 Bounded movement with backend-specific issuance boundaries

Require: Eligible job J, ceiling $q_{\max}$, backend $\mathcal{B}$
1: $\mathcal{U} \leftarrow \textsc{Partition}(J, q_{\max})$
2: **for** unit $u \in \mathcal{U}$ in logical order **do**
3: Issue u on the existing movement path
4: Advance after $\mathcal{B}$'s issuance boundary
5: **end for**
6: Retain $\mathcal{B}$'s logical-job completion dependency
7: Report completion only when that dependency is satisfied

Table 10 shows that the implementations share the same submission policy but not an identical synchronization primitive. An issuance boundary determines when the *next* unit may be submitted; it does not redefine logical-job completion.

A submission may contain multiple copy descriptors. BOLT-KV reaches the specified issuance boundary between calls rather than placing all descriptors in one large submission or immediately enqueueing the full sequence.

In the vLLM KV-store backend, the wrapper invokes the existing `copy_blocks` routine on successive groups of blocks. Each invocation issues a `cuMemcpyBatchAsync` call, and the original per-job completion event is recorded only after the final invocation. Host-call return therefore permits the next submission, while the existing event determines logical store completion.

The independent SGLang boundary probe in Appendix C.2 supports host-call return as the issuance boundary on that measured path: for a 4 MiB D2H unit, host-call p50 is 1.052 ms versus 1.056 ms including completion. Detailed drain measurements appear in Table 12. These measurements do

not imply that host return generally establishes device completion. Applying BOLT-KV to another backend therefore requires validating its issuance boundary separately from its logical-job completion condition.

B.3 Operating Point and Calibration Audit

The headline experiments use $q_{\max} = 4\,\mathrm{MiB}$, fixed during preliminary development. The launch-level records underlying that preliminary choice are unavailable, so we do not reconstruct or retrospectively justify the choice from later measurements.

To assess submission granularity independently of end-to-end serving, we conduct a separate calibration audit using a bulk D2H mover and a small latency-critical D2H victim. For each candidate ceiling, we run 20 independent launches and measure both victim latency and realized movement throughput, since finer submissions can reduce interference at the cost of additional submission overhead.

Let $L_p(q)$ denote victim p99 latency and $\mu(q)$ movement throughput for candidate ceiling q, with unbounded references $L_{p,0}$ and μ_0. For $\mathcal{Q} = \{2, 4, 8, 16\}\,\mathrm{MiB}$, the primary rule selects the largest candidate satisfying

$$L_p(q) \le 2\,\mathrm{ms}, \qquad \frac{\mu(q)}{\mu_0} \ge 0.95. \tag{9}$$

If no candidate satisfies both criteria, the pre-defined fallback admits candidates satisfying

$$\frac{\mu(q)}{\mu_0} \ge 0.80, \qquad \frac{L_{p,0}}{L_p(q)} \ge 5, \tag{10}$$

and selects the admitted candidate maximizing

$$U(q) = \frac{\mu(q)/\mu_0}{L_p(q)/L_{p,0}}. \tag{11}$$

Table 11 reports the audit measurements. Victim p99 is computed within each launch over 600 D2H victim calls and then averaged across launches. Movement throughput is normalized within each launch by its own unbounded reference before averaging. Brackets denote 95% percentile-bootstrap confidence intervals of the across-launch mean. The unbounded references are $L_{p,0} = 120.0\,\mathrm{ms}$ and $\mu_0 = 4.31\,\mathrm{GB/s}$.

Table 11: Calibration audit for submission-ceiling selection.

Submission ceiling	Victim p99 (ms)	$\mu(q)/\mu_0$
Unbounded	120.034 [119.973, 120.107]	1.000
2 MiB	1.283 [1.277, 1.288]	0.847 [0.846, 0.848]
4 MiB	2.266 [2.261, 2.272]	0.898 [0.897, 0.899]
8 MiB	4.230 [4.222, 4.239]	0.946 [0.945, 0.948]
16 MiB	8.158 [8.145, 8.172]	0.965 [0.961, 0.967]

No candidate satisfies both primary criteria. The 2 MiB setting meets the latency criterion but not the throughput-retention criterion; 16 MiB meets the latter but not the former. The 4 MiB and 8 MiB settings satisfy neither: 4 MiB exceeds the latency threshold by 0.27 ms, while 8 MiB misses the throughput threshold by 0.004. All four candidates satisfy the fallback constraints, which select 2 MiB by maximizing $U(q)$.

This later audit does not alter the headline configuration: the serving experiments retain the 4 MiB setting fixed before their measurements. Section 5.2 and Appendix C.3 separately evaluate sensitivity to submission size. The same 4 MiB headline ceiling is used for SGLang and H2D restoration; for Qwen3-8B on the vLLM KV-store backend, indivisible 2.25 MiB blocks make one block the effective submission unit under this ceiling.

B.4 Confidential-Computing Boundary

PBSI changes stream placement within the existing GPU execution context, and bounded submissions retain the existing protected CPU–GPU transfer path. Neither the additional CUDA stream nor the recovery scratch allocation defines a new security domain. BOLT-KV changes scheduling within the existing confidential-computing boundary.

C Additional Evaluation Details

This appendix gives the experimental settings, estimators, and additional controls supporting Section 5. Mechanism probes and BOLT-KV implementation details are described in Appendices A and B, respectively.

C.1 Experimental Setup and Estimators

All Section 5 experiments use an NVIDIA H100 NVL with GPU confidential computing enabled and driver 595.71.05. Unless otherwise stated, experiments use Qwen3-8B in BF16. The vLLM experiments use vLLM 0.28.0 and PyTorch 2.13.0. Serving experiments maintain four continuously decoding foreground requests unless stated otherwise.

Controlled-replay experiments fix logical movement sizes, ordering, and release rules within each operating point. Backend experiments instead obtain movement jobs from the runtime's KV-store policy, so identical request arrivals need not produce identical selected movement work.

Statistics. Timing conventions follow Appendix A.1. Foreground throughput Θ is the number of tokens delivered to the client during the observation window divided by the window duration. The independent process launch is the statistical unit; tokens, iterations, transfer calls, and recovery events within one launch are repeated observations rather than independent replicates.

For matched serving experiments, the movement/no-movement ratio is formed within each seed block before aggregation:

$$R_r = \frac{1}{n} \sum_{i=1}^{n} \frac{\Theta_{r,i}}{\Theta_{0,i}}. \tag{12}$$

Matched-block bootstrap resampling is used for paired ratios and contrasts; arm-specific summaries use independent-launch resampling. Percentiles are computed within launch before across-launch aggregation, using the estimator specified for each experiment family.

Movement accounting. For controlled replay, *offered* work has been released by the trace, *issued* work has begun submission, and *completed* work has satisfied the observed logical completion condition. Backlog is released work not yet observed complete. Windowed completions may include work released before the observation window, while work released near its end may complete afterward; completed/offered ratios are therefore not automatically cohort-completion fractions.

Factorial design. The component experiment in Section 5.1 crosses legacy/PBSI inference with unbounded/bounded movement over 20 matched seed blocks. Each movement arm is normalized by its corresponding no-movement control. The interaction is the difference between the bounded-minus-unbounded effects under PBSI and legacy inference, estimated as $+0.083$. Table 3 reports the four arm-level results. This experiment is separate from both the PBSI decomposition in Appendix A.4 and the KV-store integration below.

C.2 SGLang Validation and Movement Pressure

Configuration. We use the public `cc-fixes-rebased` branch at commit `64121c62a8dca7f5d1fdd39c8155ac3b8fc9da70`. The existing CC readback worker is enabled and unchanged in every arm; no PBSI modification is applied. The workload uses four streaming requests with 96-word prompts and 20-s measurement windows.

The replay mover accesses live KV buffers in the scheduler process and CUDA context. The frozen trace contains 685 logical events: 36 exceed 4 MiB and account for 82.2% of total bytes, and the

Table 12: Independent SGLang boundary probe. Durations are in milliseconds. Ratio p95 is the p95 of per-transfer post-return-drain/total-duration ratios.

Size	Host-call p50	Drain p95	Total p50	Ratio p95
4 MiB	1.052	0.0044	1.056	0.0042
499.5 MiB	120.872	0.0194	120.889	0.0002

Table 13: SGLang serving results over 20 matched launches, with 95% confidence intervals. ITL p99 is the mean of per-launch p99 values; Max ITL is the mean of per-launch maxima.

Metric	Unbounded	BOLT-KV
Serving ratio R	$0.7848\ [0.7834, 0.7859]$	$0.9972\ [0.9954, 0.9987]$
ITL p99 (ms)	$101.41\ [96.02, 107.74]$	$8.023\ [7.962, 8.072]$
ITL max (ms)	$217.51\ [212.91, 222.36]$	$28.91\ [8.85, 58.86]$

largest event is 499.5 MiB. Both movement arms preserve the absolute release rule and event order, changing only submission granularity.

Host-return boundary. Table 12 reports the independent contiguous-buffer probe supporting host-call return as the issuance boundary on this measured path.

The ratio is computed per transfer before taking its percentile. At 4 MiB it satisfies the pre-defined $p95(\mathrm{drain}/\mathrm{total}) \leq 0.10$ criterion. This supports host-call return as the issuance boundary in this configuration; it does not imply that host return generally establishes device completion.

Serving results. The serving comparison uses 20 matched seed blocks per arm. Table 13 reports the resulting launch-level statistics. The across-launch mean windowed completed movement is nearly identical: 25.76 GB for the unbounded arm and 25.77 GB for BOLT-KV.

At this operating point, the two arms receive nearly the same offered movement work. The unbounded arm has zero final logical backlog in all launches. The bounded arm has median final backlog zero and mean backlog 0.05 events, reflecting one residual event in one launch. We therefore do not interpret finite-window completion counts as exact equality of completed movement.

Readback exposure. The mechanism study uses 20 independent instrumented launches per movement arm. A readback is active when movement is in progress at readback issue. For launch i, let c_i denote its inactive readback baseline and define

$$X_i^{\max} = \max_{r \in \mathrm{active}_i} T_{\mathrm{copy}}(r) - c_i. \tag{13}$$

Table 14 reports across-launch mean $\pm$ SD.

For the bounded arm, regression against the remaining active submission unit retains a weaker relationship, with slope 0.733 ± 0.034 and $R^2 = 0.439 \pm 0.060$. We interpret this as residual unit-scale dependence rather than a precise scheduling law.

Rare long serving stalls remain possible after bounding: the bounded serving arm has a mean launch-maximum ITL of 28.9 ms but a median of 9.14 ms because an approximately 200 ms excursion is retained rather than trimmed. Readback-call excess is therefore not an upper bound on end-to-end ITL.

Pressure and schedule cost. The pressure study uses 20 matched launches per arm at each operating point. Offered load is normalized by the independently measured 499.5 MiB D2H rate of 4.3351 GB/s, giving normalized offered duties 0.149, 0.298, and 0.597. This is an offered byte-rate normalization, not measured Copy Engine utilization. Table 15 summarizes the resulting movement-schedule cost.

BOLT-KV remains close to its no-movement serving baseline across these points, while lateness grows substantially at the highest offered load. Nonzero final backlog is also observed there, so these finite operating points do not establish a maximum sustainable movement rate.

Table 14: SGLang readback statistics over 20 independent launches. Latency values are in milliseconds; values are mean ± SD.

Metric	Unbounded	BOLT-KV
Inactive readback p50	0.1325 ± 0.0048	0.1337 ± 0.0061
Active readback p99	83.76 ± 19.02	1.510 ± 0.026
Mean maximum excess	109.57 ± 14.69	1.512 ± 0.058
Slope vs. remaining logical event	1.0061 ± 0.0082	0.00207 ± 0.00042
R^2 vs. remaining logical event	0.9886 ± 0.0272	0.0388 ± 0.0149

Table 15: Movement-schedule cost on SGLang. Occupancy is the fraction of the observation window spent inside movement host calls, not physical engine utilization. Lateness is measured against the absolute release schedule.

Offered duty	Occupancy bulk	Occupancy BOLT	BOLT lateness p50 (ms)	BOLT lateness p99 (ms)	Final backlog mean (events)
0.149	0.134	0.160	0.135	522	0.00
0.298	0.320	0.355	0.137	1267	0.00
0.597	0.604	0.772	564.8	2255	0.10

Windowed byte accounting is phase-sensitive. At the highest pressure, the cross-arm offered-work ratio is approximately 0.956, and bounded-arm completed/offered accounting can exceed one because windowed completions may include work released outside the corresponding finite-window cohort. We therefore do not use such ratios as direct evidence of movement retention.

C.3 vLLM KV-Store Validation and Granularity

For Qwen3-8B, the KV-store integration in Table 4 uses the backend's indivisible 2.25 MiB blocks under the fixed 4 MiB operating point, yielding one block per bounded submission. Checks verify partition coverage, block ordering, and retention of the existing logical completion dependency. An offline check over 5,000 synthetic partitioning jobs reports no partition failures, and serving launches report no client errors or unexpected preemptions.

The integration uses 20 matched seed blocks. The serving ratio is 0.801 [0.797, 0.806] for Stock and 0.885 [0.881, 0.890] for BOLT-KV; the direct paired comparison gives a BOLT-KV/Stock ratio of 1.106 [1.098, 1.112]. The corresponding latency results are reported in Table 4.

Separate granularity study. This sensitivity experiment uses 20 matched launches per configuration, each with its own no-store control. It uses a heavier store-inducing workload than the headline integration: 26 background requests of about 3.5K prompt tokens (221 KV blocks each) released on an absolute 0.385-s schedule, compared with 52 requests of about 0.9K tokens (55 blocks each) under a back-pressured schedule in the headline integration. One unbounded submission therefore moves about 497 MiB rather than 124 MiB. Absolute Stock latency values from the two experiments are consequently interpreted only against their own contemporaneous controls. Table 16 summarizes serving performance across the tested submission sizes.

Every tested bounded size improves R and reduces ITL p99 relative to the unbounded arm. Among bounded configurations, smaller units favor throughput while ITL p99 remains similar across the tested range.

C.4 Second Model: Llama-3.1-Nemotron-Nano-8B

To test whether the headline effects extend beyond Qwen3-8B, we repeat the SGLang controlled-replay and vLLM KV-store experiments with Llama-3.1-Nemotron-Nano-8B[3]. The model uses a Llama-3.1 architecture with 32 layers and 2.0 MiB KV blocks, so the 4 MiB ceiling admits two blocks per submission. For SGLang, we replay the same frozen 685-event D2H trace (byte-identical,

[3] https://build.nvidia.com/nvidia/llama-3_1-nemotron-nano-8b-v1

Table 16: vLLM KV-store granularity study. Each configuration uses 20 matched launches and its own no-store control; ITL p99 is the mean of per-launch p99 values. This workload is separate from the headline KV-store integration.

Submission unit	Serving ratio R	ITL p99 (ms)
Unbounded	0.687	281.27
2.25 MiB	0.801	102.92
4.5 MiB	0.778	105.31
9 MiB	0.747	108.70
18 MiB	0.733	104.37

Table 17: Headline effects on a second 8B model (20 matched seed blocks per arm; all launches valid). For Nemotron, R is mean [95% CI], and paired throughput ratios are BOLT-KV/Stock within seed block. Max ITL is the mean of per-launch maxima. Qwen3-8B results use the same experiments as Table 4 and are included for reference.

		Qwen3-8B		**Nemotron-Nano-8B**	
Runtime	**Metric**	**Stock**	**BOLT-KV**	**Stock**	**BOLT-KV**
SGLang	R	0.785	0.997	0.779 [0.778, 0.781]	0.993 [0.991, 0.995]
	ITL p99 (ms)	101.4	8.0	88.5	8.0
	Max ITL (ms)	217.5	28.9	246.3	29.3
	Paired ratio	1.271		1.274 [1.271, 1.278]	
vLLM	R	0.801	0.885	0.932 [0.929, 0.934]	0.952 [0.949, 0.955]
	ITL p99 (ms)	73.7	41.4	35.4	23.3
	Max ITL (ms)	280.2	69.2	84.2	38.5
	Paired ratio	1.106		1.022 [1.018, 1.025]	

including SHA-256) with the same seeds, windows, and estimators. For vLLM, we retain the same request workload and runtime configuration while exercising the model's native KV-store lifecycle and block geometry.

We preregistered throughput improvement and ITL p99 reduction in each runtime before the second-model measurements. All four paired effects have the predicted direction, with 95% confidence intervals excluding the corresponding nulls. The paired BOLT-KV-minus-Stock ITL-p99 differences are -80.5 ms [-85.1, -74.9] for SGLang and -12.1 ms [-12.6, -11.7] for vLLM. Table 17 summarizes the headline results.

On SGLang, where the offered movement is byte-identical across models, unbounded throughput ratios are also similar ($R = 0.785$ versus 0.779), while BOLT-KV restores both models to within 1% of their respective no-movement baselines. The close agreement shows similar unbounded degradation and recovery toward baseline with BOLT-KV across both models under the same offered transfer workload.

On vLLM, the runtime realizes a different store workload because the two tokenizers produce different token counts for the same prompts. Stock interference is smaller for Nemotron ($R = 0.932$), and the corresponding improvement is also smaller: BOLT-KV provides a 2.2% paired throughput gain while reducing ITL p99 by $1.5\times$ and mean launch-maximum ITL by $2.2\times$. The direction nevertheless matches the pre-registered prediction.

The second model also exercises a different backend granularity: each bounded vLLM submission contains two 2.0 MiB KV blocks rather than the single 2.25 MiB block used by Qwen3-8B. This indicates that the byte-ceiling abstraction carries over to a submission that groups multiple backend blocks.

All launches are retained. In SGLang, one BOLT-KV launch exhibits a movement-lateness excursion with a p50 of 9.4 s, and three launches finish with one backlogged event. Separately, foreground launch-maximum ITL varies across launches: its mean is 29.3 ms and its median is 9.2 ms, indicating that rare foreground tail excursions remain.

Table 18: 8K recovery results over 20 independent launches per method. Bracketed values are 95% confidence intervals.

Metric	Reprefill	Stock Restore	BOLT-KV Restore
T_{ext} mean (ms)	457.6 [456.9, 458.3]	1049.8 [690.1, 1717.7]	799.5 [759.9, 868.0]
T_{ext} median (ms)	457.4	690.1	761.2
Foreground progress during recovery	7.64% [7.46, 7.83]	9.88% [1.92, 20.74]	22.67% [20.28, 26.66]
Fixed-slot foreground retention	92.78% [92.49, 93.09]	89.09% [88.15, 90.70]	89.81% [89.48, 90.15]
ITL samples > 100 ms	0.1490%	0.1485%	0
Mean launch maximum ITL (ms)	546.8 [546.0, 547.8]	652.4 [584.1, 687.1]	47.34 [46.95, 48.00]

C.5 8K Recovery Under Concurrent Decoding

Setup and validation. We compare no recovery, Reprefill with prefix caching disabled, Stock Restore, and BOLT-KV Restore while four unrelated requests decode. Both restore arms use the same selected KV layout and transfer path, with 576 host and scratch blocks, allocation-bounds checks, and restored-byte validation. The scratch destination avoids writes into foreground-owned KV, and restored state is not re-entered into attention. CPU KV preparation precedes the timed experiment, which excludes the preceding D2H store. The experiment therefore measures the H2D transfer component of recovery; eviction and restoration are evaluated separately.

Reprefill configuration. The frozen Reprefill arm uses vLLM 0.28.0 with chunked prefill enabled, `max_num_batched_tokens=8192`, prefix caching disabled, and `max_tokens=1` for the recovery request. The tokenized recovery input contains approximately 8381 tokens. Client ITL traces are consistent with a large first prefill chunk followed by a small remainder.

Protocol and metrics. The reported campaign contains 20 independent launches per arm, 80 launches in total. Each launch completes four recovery events on a common 6-s absolute schedule. All 80 launches are valid, with no replacement or censoring.

For event i, the fixed observation slot is $W_i = [t_i, t_i + 6\,\text{s})$. Let $N_{r,\ell,i}$ and $N_{0,\ell,i}$ denote foreground token counts for recovery arm r and its matched control in launch block ℓ. Fixed-slot foreground retention is

$$S_{r,\ell} = \frac{\sum_i N_{r,\ell,i}}{\sum_i N_{0,\ell,i}}. \tag{14}$$

Launch-level values are aggregated before statistical inference.

Progress during recovery separately compares foreground generation during the actual recovery intervals with the matched control expectation for those same intervals. It is not the fixed-slot retention $S_{r,\ell}$. Primary recovery latency T_{ext} spans action dispatch to observed completion, using the same external timing boundary for every recovery method.

Rare Stock stall and table conventions. One Stock Restore launch has mean external recovery time of approximately 7.0 s, compared with approximately 690 ms for a typical Stock launch. It completes all four events and is retained without replacement, producing the difference between Stock's mean and median in Table 18. Confidence intervals use launch-level bootstrap resampling, and maximum ITL denotes the mean of per-launch maxima over the fixed observation slots.

Smaller-budget comparison. After the frozen campaign, we preregistered an additional comparison that changed only `max_num_batched_tokens` from 8192 to 2048, with a matched control. Both attempted launches failed during initial prefill, before any recovery event, at the assertion `request.num_output_placeholders >= 0` in vLLM's async scheduler. The failure reproduced in 3/3 standalone runs without instrumentation, plugins, or concurrent requests. Diagnostic runs also failed at budget 4096, whereas budget 8192 completed normally. The two aborted launches are retained separately from the 80 valid launches reported above; none of those 80 launches was replaced. We therefore report no recovery comparison against Reprefill with a smaller prefill budget.