KVRECLAIM: RECOMPUTE-AWARE PLACEMENT AND MIGRATION-FREE RECLAMATION FOR KV CACHES ON ZNS SSDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) inference increasingly relies on long contexts and persistent conversation histories, causing KV caches to grow rapidly and become a major bottleneck for scalable serving. SSD-backed KV caching provides a cost-effective means of extending scarce GPU and host memory, but conventional SSD accesses incur substantial KV recovery latency, especially as device-managed garbage collection (GC) induces data migration that can interfere with foreground KV I/O. Zoned Namespace (ZNS) SSDs provide an opportunity to eliminate such unnecessary migration by exposing data placement and space reclamation to the host. Based on this insight, we present *KVReclaim*, a host–SSD coordinated KV-cache management framework for ZNS SSDs that exploits KV semantics across the storage hierarchy. *KVReclaim* manages KV states at conversation granularity and flushes them into zones according to their runtime access behavior and recomputation overhead. During garbage collection (GC), it directly discards selected KV states instead of migrating them, thereby reducing GC-induced data movement. *KVReclaim* further dynamically selects between SSD-backed KV recovery and recomputation by comparing their estimated costs, thereby reducing recovery overhead under varying SSD I/O conditions. Experiments on long-context and multi-turn LLM workloads show that *KVReclaim* reduces E2E latency by 74.2% on average compared with state-of-the-art SSD-backed KV-cache management schemes.

1 INTRODUCTION

Large language models (LLMs) have been widely adopted in a variety of applications, including coding assistants (Yang et al., 2024), machine translation (Cui et al., 2025), and document summarization (Ravaut et al., 2024). Many of these applications rely on long contexts and persistent conversational histories, making the efficient serving of long-running LLM workloads increasingly challenging, as each decoding step must attend to an ever-growing context (Xie et al., 2026; Jeong & Ahn, 2025).

To avoid repeatedly recomputing representations for previously processed tokens, modern LLM serving systems maintain a key–value (KV) cache in GPU high-bandwidth memory (HBM) and host DRAM, enabling efficient reuse of cached states during subsequent decoding steps (Gao et al., 2024). However, KV-cache memory consumption grows linearly with context length and the number of concurrent requests, quickly exhausting the limited capacity of GPU HBM and host DRAM (Kwon et al., 2023). GPU HBM must also accommodate model weights, activations, and other runtime states, while provisioning sufficient host DRAM for large KV caches incurs substantial hardware costs. As a result, KV-cache capacity has become a major bottleneck for scalable LLM serving (Chen et al., 2025). To mitigate this bottleneck, recent studies propose *SSD-backed KV caching*, which offloads KV states to SSDs and reloads the required entries into GPU memory on demand (Qin et al., 2025; Hu et al., 2026). By leveraging the large capacity and low cost of SSDs, SSD-backed KV caching provides a practical solution for accommodating large KV caches and enabling long-context LLM inference (Sheng et al., 2023).

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.

Recent SSD-backed KV-cache systems improve performance through KV placement, prefetching, eviction, and request scheduling. For example, CachedAttention and FlashGen organize KV states across HBM, DRAM, and SSD and prefetch or recover KV states to overlap storage accesses with computation (Gao et al., 2024; Jeong & Ahn, 2025). IMPRESS (Chen et al., 2025) exploits the importance of KV states to improve their placement and reduce SSD I/O, while Bidaw (Hu et al., 2026) coordinates request scheduling and KV eviction with storage-tier information and predicted future accesses. Despite these optimizations, existing systems largely treat the SSD as an opaque storage device and leave physical space reclamation to the SSD itself.

This abstraction can substantially degrade LLM inference performance. SSD access latency is orders of magnitude higher than that of GPU HBM and host DRAM, and can vary considerably under different runtime I/O conditions, resulting in long and unpredictable KV-loading stalls (Qureshi et al., 2023; Qiu et al., 2025). More importantly, conventional SSDs perform garbage collection (GC) transparently in the background. When reclaiming a block, SSD GC must migrate its still-valid pages to another physical location before erasing the block. These migrations generate additional internal read and write traffic, which competes with foreground KV-cache accesses for SSD bandwidth and can further degrade inference performance.

Such GC-induced data migration is particularly wasteful for KV caches. A KV state that is considered valid by the SSD may already be a candidate for eviction at the host, or may be inexpensive to recompute when needed again (Zhang et al., 2023; Hu et al., 2026; Yao et al., 2025). Consequently, the SSD may migrate a KV state during GC only for the host KV-cache manager to evict it shortly afterward. This unnecessary migration results from a fundamental mismatch: the host makes KV eviction decisions based on application-level semantics, whereas conventional SSDs make GC decisions based only on physical page validity. Because conventional SSDs expose only a black-box block interface, the host cannot inform the SSD which valid KV states can be safely discarded, and the SSD cannot incorporate KV semantics into its reclamation decisions (Bjørling et al., 2021).

Zoned Namespace (ZNS) SSDs provide an opportunity to bridge this gap by exposing space reclamation to the host. Unlike conventional SSDs, where the device independently determines when physical space is reclaimed through GC, ZNS SSDs organize storage into zones and require the host to explicitly reset a zone to reclaim the physical space occupied by that zone. In other words, the host determines when a zone becomes reclaimable and, by issuing a zone reset, determines when its physical space is actually reclaimed (Bjørling et al., 2021; Han et al., 2021). This host-controlled reclamation eliminates the black-box GC behavior of conventional SSDs and allows the host to incorporate application-level KV semantics into space-reclamation decisions.

Based on this insight, we propose *KVReclaim*, a recomputation-aware KV-cache management framework for ZNS SSDs that coordinates KV placement, eviction, reclamation, and recovery. *KVReclaim* first performs drop-aware KV placement by co-locating conversations with similar drop costs within the same zones. This organization increases the likelihood that a zone becomes fully reclaimable when low-cost KV states are evicted, allowing the host to directly reset the zone without migrating remaining valid KV states. When storage space needs to be reclaimed, *KVReclaim* globally identifies low-cost conversations and evicts them to create empty zones, thereby avoiding unnecessary GC migration and the associated I/O interference. Furthermore, *KVReclaim* performs cost-aware KV recovery by dynamically comparing the runtime cost of reading a KV state from the SSD with the cost of recomputing it, and selects the lower-cost option. Together, these mechanisms reduce reclamation-induced interference while adapting KV recovery to changing SSD I/O conditions. In summary, this paper makes the following four contributions:

- We identify the mismatch between KV eviction and device-level space reclamation as a source of SSD-induced interference, and propose *KVReclaim*, a ZNS-based KV-cache management framework that incorporates KV semantics into storage reclamation.
- We present drop-aware KV placement, which co-locates conversations with similar drop costs so that host-side KV eviction can be coordinated with zone reclamation, avoiding unnecessary migration of valid KV states.
- We design cost-aware KV recovery, which dynamically chooses between SSD reads and KV recomputation based on their runtime costs, thereby adapting KV recovery to changing SSD I/O conditions.

- We evaluate *KVReclaim* on long-context and multi-turn LLM workloads. Compared with state-of-the-art SSD-backed KV-cache systems, it reduces E2E latency by 74.2% on average and mean KV recovery latency by 65.4%.

2 Background and motivation

2.1 KV Cache and SSD-backed KV Cache

Autoregressive LLM inference proceeds in two stages: prefill and decode. During prefill, the model processes the input prompt and stores the key and value states of all input tokens in the KV cache. These cached states preserve the attention information of preceding tokens and can be reused during subsequent token generation. During decode, the model generates one token at a time. To generate each new token, the attention module accesses the key and value states of preceding tokens from the KV cache (Kwon et al., 2023). The key and value states of the newly generated token are then appended to the KV cache for use in subsequent decoding steps. By retaining and reusing these states, the KV cache avoids repeatedly recomputing the preceding context during autoregressive generation, thereby improving the overall inference performance (Kwon et al., 2023; Gao et al., 2024). However, the limited capacity of HBM and DRAM prevents them from storing all KV states, particularly for long-context conversations that generate a large volume of KV data.

To further improve inference performance under limited HBM and DRAM capacity, SSD-backed KV caching incorporates SSDs into the storage hierarchy to provide additional capacity for KV states. When upper-tier memory becomes constrained, selected KV states are offloaded to SSDs and retained for future reuse. Once these states are needed again, they are recovered from SSDs back to the upper memory tiers for subsequent inference. The high latency and limited bandwidth of SSDs, however, can introduce substantial overhead when KV states are transferred across storage tiers. Numerous studies have focused on optimizing cross-tier KV management, which can be broadly classified into three categories:

Optimizing cross-tier KV management. Gao et al. (Gao et al., 2024) proposed *CachedAttention*, which uses the future request order to guide KV prefetching and eviction, thereby overlapping KV transfers with computation. Jeong et al. (Jeong & Ahn, 2025) proposed *FlashGen*, which reorders requests according to their GPU access requirements for reducing head-of-line blocking caused by KV transfers. Hu et al. (Hu et al., 2026) proposed *Bidaw*, which schedules requests according to KV residency and recovery cost while using generated responses to guide KV eviction between host memory and SSDs, thereby reducing KV recovery stalls.

Reducing KV Data Movement. Chen et al. (Chen et al., 2025) proposed *IMPRESS*. By identifying important KV states based on similarity across attention heads, it reorganizes these states into fewer storage chunks. This organization allows only the required chunks to be retrieved from SSDs, thus reducing unnecessary data movement. Gao et al. (Gao et al., 2025) proposed *HCache*, which stores compact intermediate activations instead of complete KV states. The corresponding KV states are reconstructed during recovery, thereby trading GPU computation for reduced SSD data movement.

Optimizing the I/O Path. Qiu et al. (Qiu et al., 2026) proposed *Tutti*, which batches KV transfers while allowing GPU threads to directly submit asynchronous SSD I/O. As a result, it reduces CPU overhead on the KV I/O path. Jeong et al. (Jeong et al., 2026) proposed *DUAL-BLADE*, which dynamically selects between the kernel page-cache path and direct NVMe access. Directly accessed KV states are placed in contiguous LBA regions, thereby improving transfer efficiency. Zheng et al. (Zheng et al., 2026) proposed *SolidAttention*, which consolidates selected KV states into coarse-grained storage blocks. These blocks are speculatively prefetched based on KV access locality. This prefetching can enhance SSD access efficiency.

Despite these optimizations, existing systems largely treat SSDs as black-box storage devices and focus on host-visible KV management. The physical placement and reclamation of KV data remain independently managed by the SSD. As a result, SSD garbage collection may migrate KV states that will soon be evicted or can be recomputed at low cost, introducing unnecessary data movement and interference with foreground KV I/O.

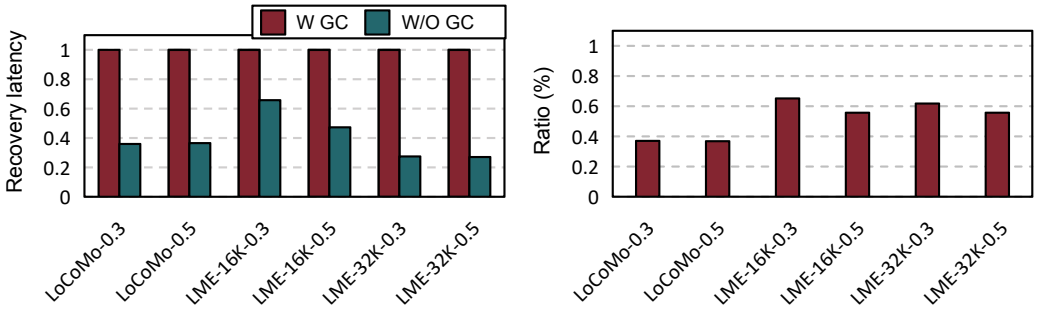

Figure 1: Average recovery latency and the eviction ratio of migrated pages when using *FlashGen* with (W-GC) or without page move (W/O-GC).

2.2 ZNS SSD

Due to the out-of-place update nature of NAND flash, SSDs rely on garbage collection (GC) to reclaim space occupied by obsolete data. When free space becomes scarce, GC selects a victim block and migrates its remaining valid pages before erasing the victim block. Such GC operations introduce additional internal data movement and can interfere with foreground I/O, thereby increasing I/O latency and reducing performance predictability (Cho et al., 2024; Yi et al., 2024).

To mitigate such GC-induced interference, Zoned Namespace (ZNS) SSDs expose storage-management responsibilities to the host rather than relying entirely on the conventional device-managed block interface (Yi et al., 2024; Kim et al., 2025). The storage space is organized into zones that follow sequential-write constraints and are explicitly managed by host software. The host can therefore control data placement and zone reclamation, allowing physical space management to account for application and workload characteristics (Yi et al., 2024; Kim et al., 2025).

2.3 Motivations

To quantify how GC affects I/O performance, we conduct experiments using *FlashGen*, an SSD-backed KV cache management scheme. The experimental setup and benchmarks are described in Section 4.2. Figure 1 (a) compares the recovery latency of *FlashGen* with and without GC.

As shown, *FlashGen* without GC reduces the recovery latency by `59.9%` compared with the case with GC. This degradation is mainly caused by the contention between GC-induced data migration and foreground I/O requests for shared SSD resources.

Figure 1 (b) shows the fraction of data migrated during GC that is subsequently evicted from the KV cache. As shown, such data accounts for between `36.8%` and `65.1%` of the total data migrated during GC. This indicates that a considerable portion of GC-induced data movement is spent on KV states that have only a short remaining lifetime in the cache. Although these KV states are still valid when GC is triggered, they are soon evicted and therefore provide little benefit from being migrated to a new physical location. Such unnecessary migration not only consumes SSD bandwidth and internal resources, but also increases contention with foreground I/O requests.

These observations motivate our research toward a solution that minimizes GC-induced data movement while maintaining efficient KV recovery. Specifically, the severe interference caused by GC migration highlights the necessity of reducing unnecessary data movement during GC. Meanwhile, the large fraction of migrated KV states that are subsequently evicted suggests that preserving all valid KV states during GC is inefficient. However, aggressively discarding valid KV states may incur additional recovery overhead when they are accessed again, particularly for KV states with high recomputation cost. To this end, *KVReclaim* organizes and places KV states according to their drop costs, selectively discards low-cost KV states during GC instead of migrating them, and dynamically chooses between SSD reads and recomputation for subsequent recovery based on the current SSD I/O load.

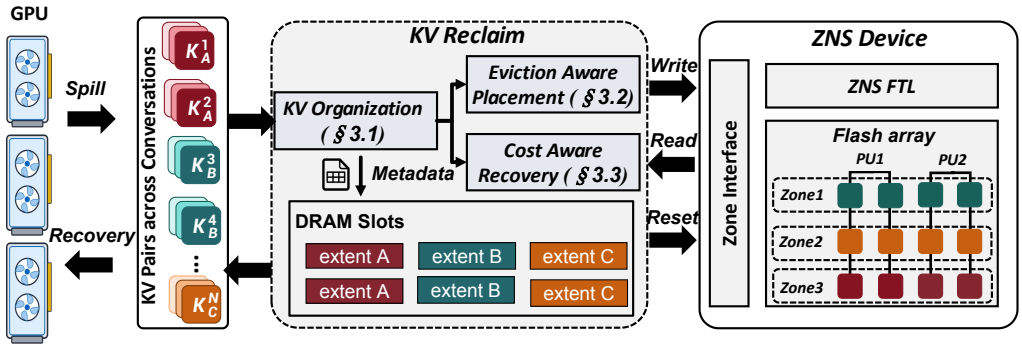

Figure 2: Overview of *KVReclaim*. Note that the KV entries represented by the same color correspond to the same conversation.

3 System Design and Implementation

3.1 Overview

We propose *KVReclaim*, a host–SSD coordinated KV cache management scheme that uses ZNS SSDs as a host-managed storage tier. Its core idea is to couple conversation-level KV eviction with host-controlled zone reclamation, allowing reclaimable zones to be created without migrating valid KV.

As shown in Figure 2, *KVReclaim* groups conversations with similar eviction costs into the same placement cohorts. During reclamation, it evaluates candidate zones based on the eviction cost of affected conversations and the resulting reclamation benefit, and directly resets zones made fully invalid by the selected eviction. For later accesses, *KVReclaim* chooses between SSD reads and recomputation according to current storage and GPU delays.

3.2 Conversation-Aware Migration-Free Zone Reclamation

KVReclaim couples conversation-level eviction with zone reset to reclaim space without migrating valid KV. Selecting a candidate zone determines the conversations whose valid KV must be evicted to make that zone resettable. Evicting these conversations invalidates their cached KV across all zones where they reside, potentially making the selected zone and additional zones fully invalid and directly resettable.

To evaluate each candidate, *KVReclaim* first estimates the eviction cost of the affected conversations. We define DropCost as a lightweight approximation of this conversation-level cost, combining access activity with recomputation cost. Access activity reflects the likelihood that discarded KV will be needed again, while recomputation cost represents the penalty of regenerating it.

The access component reflects both frequency and recency, as defined in Equation 1.

$$R(c) = \frac{1}{\bar{T}_c + \epsilon} \exp\left(-\frac{\Delta t_c}{\bar{T}_c + \epsilon}\right). \tag{1}$$

Here, $\bar{T}_c$ is the average interval between consecutive accesses, and Δt_c is the elapsed time since the latest access. The inverse interval reflects access frequency, while the exponential term reduces the weight as the inactive period grows relative to the conversation's typical access interval.

The recomputation component depends on the recomputation depth and preceding prefix length. For each depth d, *KVReclaim* uses an affine model to estimate the cumulative execution cost through that layer, as shown in Equation 2.

$$\hat{p}_d(i) = a_d + b_d i. \tag{2}$$

Here, a_d is the intercept, while $b_d i$ represents the additional cost associated with the preceding prefix length. The model parameters are fitted online using ordinary least squares (OLS), with the fitting statistics updated through an exponential moving average using a new-sample weight of 0.1.

For recovery range r, each recomputed token at position i uses the model associated with its deepest required layer d_i. The request-level recomputation cost is given by Equation 3.

$$T_{\text{recompute}}(c, r) = \sum_{i \in r} \hat{p}_{d_i}(i). \tag{3}$$

At the conversation level, these estimates are smoothed over time using Equation 4.

$$\overline{T}_{\text{recompute}}(c) \leftarrow (1 - \alpha)\overline{T}_{\text{recompute}}(c) + \alpha T_{\text{recompute}}(c, r). \tag{4}$$

We set $\alpha = 0.1$ and initialize the value from the current recomputation estimate when no recovery history is available.

The access activity and recent recomputation cost are combined into DropCost in Equation 5.

$$\text{DropCost}(c) = R(c)\,\overline{T}_{\text{recompute}}(c). \tag{5}$$

A lower DropCost indicates a lower estimated eviction cost. The metric is used for relative comparison rather than as an absolute latency estimate.

With conversation-level DropCost available, ***KVReclaim*** evaluates each candidate zone. For zone z, $\mathcal{C}(z)$ denotes the conversations with valid KV in that zone. Reclaiming z evicts these conversations and invalidates their cached KV across all zones where they reside, which may make additional zones fully invalid. We denote all resulting resettable zones by $\mathcal{R}(z)$.

To rank candidate zones, ***KVReclaim*** balances the aggregate DropCost of the affected conversations against the reclamation benefit, reflected by the number of resettable zones and their existing invalid space. The resulting ReclaimScore is defined in Equation 6, where $I(z')$ denotes the existing invalid space of zone $z' \in \mathcal{R}(z)$ and S_{zone} denotes the zone capacity.

$$\text{ReclaimScore}(z) = \frac{\sum_{c \in \mathcal{C}(z)} \text{DropCost}(c)}{|\mathcal{R}(z)|} \left(1 - \frac{\sum_{z' \in \mathcal{R}(z)} I(z')}{|\mathcal{R}(z)| S_{\text{zone}}}\right). \tag{6}$$

The first factor relates the eviction cost to the number of zones released, while the second favors candidates with more existing invalid space. ***KVReclaim*** selects the candidate with the lowest ReclaimScore.

Once selected, all cached KV of the conversations in $\mathcal{C}(z)$ is invalidated, and every zone in $\mathcal{R}(z)$ is reset without relocating the discarded KV. Conversation histories and lightweight metadata are retained so that DropCost can be recomputed when a conversation becomes active again.

In summary, ***KVReclaim*** couples conversation-level eviction with host-controlled zone reclamation, enabling one eviction decision to make one or more zones directly resettable without migrating valid KV.

3.3 Reclamation-Friendly KV Placement

The effectiveness of conversation-aware reclamation depends on how conversations are distributed across zones. Mixing conversations with very different eviction costs can make a candidate zone involve both cheap and expensive evictions, limiting the quality of reclamation choices. ***KVReclaim*** therefore co-locates conversations with similar DropCosts to create more favorable reclamation candidates.

Conversations with similar DropCosts are grouped into the same cohort. Each cohort maintains a zone sequence containing zero or more filled predecessor zones and one active zone for new KV writes. When the active zone fills, it becomes a predecessor zone and a free zone becomes the new active zone. ***KVReclaim*** maintains at most n cohorts.

When KV is spilled, an existing cohort affinity is reused and the new KV is appended to that cohort's active zone. Without an affinity, a new cohort is created while fewer than n cohorts exist; otherwise, placement is determined by the current DropCost.

Each cohort is represented by a CohortDrop, as defined in Equation 7, where $B(c, g)$ denotes the valid KV bytes of conversation c associated with cohort g, including those stored in predecessor zones.

$$\mathrm{CohortDrop}(g) = \frac{\sum_{c \in g} B(c, g)\, \mathrm{DropCost}(c)}{\sum_{c \in g} B(c, g)}. \tag{7}$$

A conversation without an affinity is assigned to the cohort whose CohortDrop is closest to its own DropCost. The affinity is retained while valid cached KV remains, so subsequent spills continue to use the same cohort. If reclamation removes all cached KV of the conversation, the affinity is cleared and the next spill performs cohort selection again using its latest DropCost.

3.4 Cost-Aware KV Recovery

When a conversation is accessed, some required KV may no longer reside in DRAM. ***KVReclaim*** first checks DRAM and then the ZNS SSD. If no valid stored copy remains, the missing KV must be recomputed from the conversation history. With an SSD copy available, ***KVReclaim*** selects between reading the stored KV and recomputing it.

The recomputation cost $T_{\mathrm{recompute}}(c, r)$ is obtained from Equation 3. Because recomputation may wait when the GPU is busy, ***KVReclaim*** also estimates the current GPU waiting time as $T^{\mathrm{GPU}}_{\mathrm{queue}}$.

The read path includes storage queueing, delay from pending or ongoing zone reset operations, and the SSD read service time. Because zone reset in ZNS is explicitly controlled by the host, ***KVReclaim*** observes when reset operations are issued and tracks the current storage request queue. This visibility allows the queueing and reset delays to be estimated accurately.

Based on these estimates, ***KVReclaim*** chooses the read path when the condition in Equation 8 holds.

$$T^{\mathrm{SSD}}_{\mathrm{queue}} + T_{\mathrm{reset}} + T_{\mathrm{service}}(c, r) \le T^{\mathrm{GPU}}_{\mathrm{queue}} + T_{\mathrm{recompute}}(c, r). \tag{8}$$

Here, $T^{\mathrm{SSD}}_{\mathrm{queue}}$ denotes the waiting time caused by queued storage requests, T_{reset} captures the reset-induced delay, and $T_{\mathrm{service}}(c, r)$ denotes the SSD read service time. If the condition does not hold, the required KV is recomputed from the conversation history.

4 Experiments and Discussions

4.1 Workloads and Baselines

Datasets and Models. We evaluate on LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2024), which cover long-running multi-turn conversations and long-context memory workloads, respectively. For LoCoMo, we preserve the original conversation and session structure. For LongMemEval, we follow its official custom-history construction pipeline and generate two workloads with 16K and 32K token budgets, denoted as LME-16K and LME-32K.Non-evidence filler sessions are removed as needed while preserving the task-specific evidence sessions. We evaluate Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct, which have different GQA configurations and thus substantially different KV-cache footprints.

Baselines. We compare against *CachedAttention* (Gao et al., 2024) and *FlashGen* (Jeong & Ahn, 2025), two representative SSD-backed KV-cache systems that manage historical KV states across HBM, DRAM, and SSD. They provide direct baselines for evaluating ***KVReclaim*** under the same tiered-storage setting.

4.2 Experimental Setup

We generate execution traces on a single NVIDIA H200 GPU and replay them using FEMU (Li et al., 2018), which provides controllable conventional SSD and ZNS SSD models. Following prior KV-cache management schemes, all systems use the same HBM management policy with

Table 1: End-to-end recovery latency and recomputation counts. CA, FG, and KR denote CachedAttention, FlashGen, and KVReclaim, respectively.

Benchmark	B	E2E P50 (s) ↓			E2E P95 (s) ↓			E2E Mean (s) ↓			Recovery #	Recompute # (Proactive)		
		CA	FG	**KR**	CA	FG	**KR**	CA	FG	**KR**		CA	FG	**KR**
Llama-3.1-8B-Instruct														
LoCoMo	0.3	1381.86	1566.12	**277.61**	9877.59	9193.11	**7212.87**	2478.43	2454.51	**1168.57**	3001	707 (0)	818 (338)	646 (108)
	0.5	1212.39	1061.52	**162.14**	5263.69	5992.77	**1555.33**	1746.63	1788.65	**581.44**	3001	260 (0)	538 (273)	248 (123)
LME-16K	0.3	130.35	131.17	**16.33**	867.66	1011.32	**709.64**	246.66	276.99	**165.40**	596	230 (0)	294 (57)	304 (31)
	0.5	97.69	57.91	**5.64**	788.69	648.63	**615.19**	206.18	160.66	**115.29**	596	186 (0)	206 (63)	217 (51)
LME-32K	0.3	404.74	377.32	**12.99**	2936.81	3022.13	**2032.04**	857.53	854.32	**485.29**	643	161 (0)	178 (17)	151 (12)
	0.5	273.21	280.51	**26.39**	3724.97	3054.27	**2573.40**	789.01	659.86	**397.89**	643	192 (0)	164 (14)	154 (6)
Qwen2.5-7B-Instruct														
LoCoMo	0.3	773.35	506.95	**199.20**	2388.57	2149.55	**784.40**	961.63	748.49	**262.93**	3001	0 (0)	0 (0)	0 (0)
	0.5	322.70	328.08	**36.18**	1461.32	1604.68	**491.83**	494.46	520.80	**113.95**	3001	0 (0)	0 (0)	0 (0)
LME-16K	0.3	10.37	9.47	**2.21**	226.97	247.52	**214.27**	49.64	50.05	**41.31**	596	48 (0)	68 (15)	68 (20)
	0.5	4.23	4.56	**0.86**	49.42	117.40	**17.34**	14.31	20.43	**3.81**	596	0 (0)	4 (4)	9 (9)
LME-32K	0.3	78.92	73.14	**2.38**	1058.20	1936.19	**834.14**	226.95	305.84	**101.22**	643	55 (0)	91 (0)	54 (0)
	0.5	42.12	36.14	**1.57**	304.36	319.26	**7.58**	93.51	93.68	**2.92**	643	0 (0)	0 (0)	0 (0)
Norm. Geo. Mean (%)		100.00	90.34	**9.86**	100.00	112.73	**43.61**	100.00	101.37	**34.90**	–	–	–	–

KV-residency budgets $B \in \{0.3, 0.5\}$, representing different memory-pressure levels. During trace replay, each system is configured with a 4-GiB host-DRAM cache and 16 GiB of emulated raw NAND flash. The flash model contains two channels with four LUNs per channel, and uses read, program, and erase latencies of 40 μs, 200 μs, and 2 ms, respectively. Each run starts from a fresh FEMU instance to isolate device state across experiments. Detailed device parameters are provided in the supplementary material.

4.3 Main Results

4.3.1 Overall Performance

We first examine the performance under different memory pressures by varying the KV-residency budget B. End-to-end (E2E) latency is our primary metric because it captures the request-visible impact of KV cache management, including KV recovery, recomputation, storage queuing, and space reclamation. We report P50 and P95 latency to characterize typical and tail performance, respectively, and use mean recovery latency to quantify the overhead of restoring KV. As shown in Table 1, *KVReclaim* consistently achieves the lowest E2E latency across the evaluated models, workloads, and memory pressures. Based on the normalized geometric mean, *KVReclaim* reduces E2E mean latency by 65.1% over *CachedAttention* and 65.6% over *FlashGen*, while reducing E2E P95 latency by 56.4% and 61.3%, respectively.

In several Qwen configurations, no recomputation is triggered because the workload intensity is too low to create sufficient storage pressure. These results suggest that the performance gains of *KVReclaim* stem not only from cost-aware recovery and reduced reclamation interference, but also from better exploiting the internal parallelism exposed by the ZNS data layout for large sequential KV accesses. As storage pressure increases, *KVReclaim* further preserves this advantage by organizing KV states with similar reclamation characteristics and reclaiming zones without migrating valid KV data, thereby reducing interference with foreground I/O. Its cost-aware recovery policy additionally selects recomputation when the SSD path becomes more expensive. Together, these mechanisms enable *KVReclaim* to better exploit ZNS SSD parallelism and its host-visible ability to control data placement and reclamation.

4.3.2 Eviction Efficiency

We next evaluate the effectiveness of KV placement and storage eviction by measuring the GPU time spent on forced recomputation after valid KV copies have been removed from storage. This metric captures the penalty introduced when storage reclamation discards KV that are later accessed again. It excludes proactive recomputation, where a valid SSD copy is still available and the system actively chooses recomputation over SSD recovery. A lower value therefore indicates that the placement and eviction policies are more effective at avoiding costly KV eviction.

Table 2: GPU time (s) spent on forced recomputation caused by storage eviction. CA, FG, and KR denote CachedAttention, FlashGen, and KVReclaim, respectively. Lower is better.

Mechanism	Llama-3.1-8B-Instruct						Qwen2.5-7B-Instruct					
	LoCoMo		LME-16K		LME-32K		LoCoMo		LME-16K		LME-32K	
	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5
CA	45595.38	18517.88	12342.96	8578.87	12179.63	20137.54	0.00	0.00	2366.18	0.00	5322.51	0.00
FG	29252.13	18724.38	10673.91	7915.38	11836.57	16575.93	0.00	0.00	2230.33	0.00	8563.73	0.00
KR	**28898.89**	**6460.98**	**10458.09**	**6884.57**	**10339.78**	**15838.11**	**0.00**	**0.00**	**2215.34**	**0.00**	**4964.55**	**0.00**

Table 3: Ablation results on LoCoMo with Llama-3.1-8B-Instruct. Evict and Cohort denote DropCost-guided eviction and DropCost-based cohort placement, respectively. Lower is better for all metrics.

Benchmark	B	E2E P50 (s) ↓				E2E P95 (s) ↓				E2E Mean (s) ↓			
		Base	Evict	Cohort	**Full**	Base	Evict	Cohort	**Full**	Base	Evict	Cohort	**Full**
Llama-3.1-8B-Instruct													
LoCoMo	0.3	426.47	362.19	461.97	**277.61**	7836.22	7340.04	**7144.26**	7212.87	1382.83	1430.19	1221.86	**1168.57**
	0.5	267.65	252.88	213.62	**162.14**	3429.84	3217.24	1663.47	**1555.33**	805.17	801.99	656.80	**581.44**

Table 2 shows that *KVReclaim* consistently incurs the lowest forced recomputation overhead in configurations where storage eviction causes subsequent recomputation. For Llama-LoCoMo with $B = 0.5$, *KVReclaim* reduces the forced recomputation GPU time by about 65% compared with both *CachedAttention* and *FlashGen*. The zero values in several Qwen configurations occur because the storage pressure is insufficient to trigger eviction that leads to forced recomputation. Overall, the lower recomputation overhead indicates that DropCost-guided placement and eviction selection better organize and remove low-cost KV states, which reduces the penalty of storage reclamation.

4.3.3 Ablation Study

We evaluate the contribution of the major mechanisms in *KVReclaim* using Llama-3.1-8B-Instruct and LoCoMo. We consider both $B = 0.3$ and $B = 0.5$ to examine the mechanisms under different KV-residency budgets. Table 3 compares the base configuration with DropCost-guided eviction, DropCost-based cohort placement, and the full design. We report E2E P50, E2E P95, and mean recovery latency to capture their effects on request latency and KV recovery.

At $B = 0.3$, DropCost-guided eviction reduces E2E P50 and P95 by 15.1% and 6.3%, respectively, while DropCost-based cohort placement reduces P95 by 8.8% and mean recovery latency by 11.6%. The full design provides the largest reduction in P50 and recovery latency, lowering them by 34.9% and 15.5% over the base configuration, respectively. Its P95 is slightly higher than that of the cohort variant, indicating that the mechanisms affect median and tail latency differently. At $B = 0.5$, the benefits become more consistent: the full design reduces P50, P95, and mean recovery latency by 39.4%, 54.7%, and 27.8%, respectively. Overall, the results show that DropCost-guided placement and eviction provide complementary benefits, while their combination with the remaining components of *KVReclaim* yields the best overall latency and recovery performance.

5 Conclusion

We present *KVReclaim*, a host–SSD coordinated KV-cache management framework for ZNS SSDs. *KVReclaim* couples conversation-level KV eviction with host-controlled zone reclamation, uses reclamation-friendly placement to improve eviction decisions, and dynamically selects between SSD recovery and recomputation.

Across long-context and multi-turn workloads, *KVReclaim* reduces normalized geometric-mean E2E P50 latency by 90.1% and 89.0%, and E2E P95 latency by 56.4% and 61.3%, over *CachedAttention* and *FlashGen*, respectively.

AI USE STATEMENT

We used OpenAI's GPT models solely to improve the grammar, clarity, and readability of this manuscript. The models were not used to formulate the research problem, develop the proposed method, design or conduct experiments, generate or analyze results, or draw scientific conclusions. All AI-assisted revisions were reviewed and verified by the authors, who take full responsibility for the final content of this paper.

REFERENCES

Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. ZNS: Avoiding the block interface tax for flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 689–703. USENIX Association, 2021.

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

Sungjun Cho, Beomjun Kim, Hyunuk Cho, Gyeongseob Seo, Onur Mutlu, Myungsuk Kim, and Jisung Park. AERO: Adaptive erase operation for improving lifetime and performance of modern nand flash-based ssds. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024.

Menglong Cui, Pengzhi Gao, Wei Liu, Jian Luan, and Bin Wang. Multilingual machine translation with open large language models at practical scale: An empirical study. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 5420–5443. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.280.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126. USENIX Association, 2024.

Shiwei Gao, Youmin Chen, and Jiwu Shu. Fast state restoration in LLM serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, pp. 128–143. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696072.

Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 2021.

Shipeng Hu, Guangyan Zhang, Yuqi Zhou, Yaya Wei, Ziyan Zhong, and Jike Chen. Bidaw: Enhancing Key-Value caching for interactive LLM serving via bidirectional computation–storage awareness. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 101–116. USENIX Association, 2026.

Bodon Jeong, Hongsu Byun, Youngjae Kim, Weikuan Yu, Kyungkeun Lee, Jihoon Yang, and Sungyong Park. DUAL-BLADE: Dual-path NVMe-direct KV-cache offloading for edge LLM inference. In *46th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 620–630. IEEE, 2026. doi: 10.1109/2575-8411.2026.00065.

Jinwoo Jeong and Jeongseob Ahn. Accelerating LLM serving for multi-turn dialogues with efficient resource management. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25)*, pp. 1–15. Association for Computing Machinery, 2025. doi: 10.1145/3676641.3716245.

Minwook Kim, Seongyeop Jeong, and Jin-Soo Kim. ZRAID: Leveraging zone random write area (ZRWA) for alleviating partial parity tax in ZNS RAID. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025.

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

Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 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.

Shi Qiu, Weinan Liu, Yifan Hu, Jianqin Yan, Zhirong Shen, Xin Yao, Renhai Chen, Gong Zhang, and Yiming Zhang. GeminiFS: A companion file system for GPUs. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 221–236. USENIX Association, 2025.

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

Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, C. J. Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-Mei Hwu. GPU-initiated on-demand high-throughput storage access in the BaM system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 325–339. Association for Computing Machinery, 2023. doi: 10.1145/3575693.3575748.

Mathieu Ravaut, Aixin Sun, Nancy Chen, and Shafiq Joty. On context utilization in summarization with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2764–2781. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.153.

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 (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31094–31116. PMLR, 2023.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

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. USENIX Association, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. doi: 10.52202/079017-1601.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached

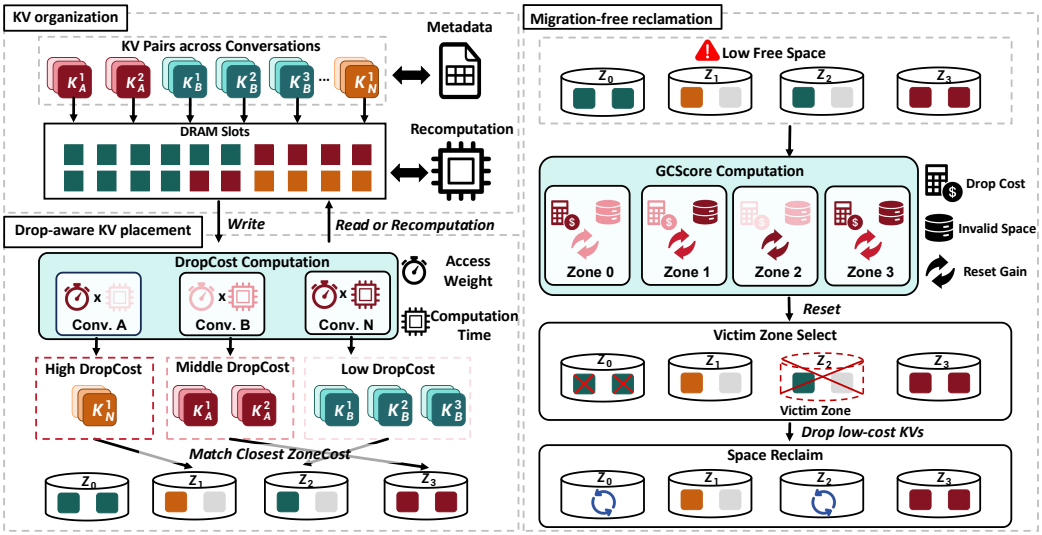

Figure 3: Detailed workflows of *KVReclaim*. It groups conversations with similar DropCost into the same placement cohorts, and dynamically selects the recovery path between read from SSDs and recomputation

knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, pp. 94–109. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696098.

Shushu Yi, Shaocong Sun, Li Peng, Yingbo Sun, Ming-Chang Yang, Zhichao Cao, Qiao Li, Myoungsoo Jung, Ke Zhou, and Jie Zhang. BIZA: Design of self-governing block-interface ZNS AFA for endurance and performance. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 313–329, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Xinrui Zheng, Dongliang Wei, Jianxiang Gao, Yixin Song, Zeyu Mi, and Haibo Chen. SolidAttention: Low-Latency SSD-based serving on Memory-Constrained PCs. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 67–82. USENIX Association, 2026.

A Appendix

A.1 Detailed Workflows of KVReclaim

This section provides additional illustrations of the major mechanisms in *KVReclaim*. Figure 3 details how drop-aware placement organizes conversations across zones and how it selects the victim zone to reclaim space without data migration.

B Additional Experimental Results

B.1 Spill-Side Performance

We first examine spill-side performance. Table 4 reports the P50, P95, and mean end-to-end latency of spill operations across the evaluated workloads and KV-residency budgets. These metrics reflect both typical and tail delays encountered when KV data is moved to the storage tier and space reclamation is required.

Table 4: Spill end-to-end latency and counts under the 10-minute arrival window. CA, FG, and KR denote *CachedAttention*, *FlashGen*, and *KVReclaim*, respectively. Lower is better for all latency metrics.

Benchmark	B	Spill E2E P50 (s) ↓			Spill E2E P95 (s) ↓			Spill E2E Mean (s) ↓			Spill #
		CA	FG	**KR**	CA	FG	**KR**	CA	FG	**KR**	
Llama-3.1-8B-Instruct											
LoCoMo	0.3	1375.25	1552.79	**274.42**	9805.66	9122.20	**7144.00**	2457.09	2436.60	**1156.93**	3001
	0.5	1207.77	1056.10	**160.08**	5202.74	5931.18	**1553.08**	1733.16	1775.13	**576.60**	3001
LME-16K	0.3	125.90	124.99	**13.42**	810.18	957.93	**661.79**	227.34	256.37	**146.06**	596
	0.5	92.79	49.67	**5.06**	745.03	597.78	**565.83**	189.52	146.07	**101.76**	596
LME-32K	0.3	386.21	369.08	**12.13**	2905.81	3011.46	**2030.67**	834.18	831.03	**469.20**	643
	0.5	261.80	270.12	**25.13**	3585.99	2917.00	**2466.27**	754.28	630.05	**373.14**	643
Qwen2.5-7B-Instruct											
LoCoMo	0.3	771.88	506.84	**197.19**	2381.00	2146.23	**782.31**	957.77	745.31	**261.59**	3001
	0.5	321.11	326.97	**36.15**	1457.96	1602.12	**491.01**	492.36	518.59	**113.35**	3001
LME-16K	0.3	8.78	9.53	**1.02**	197.78	213.23	**179.39**	44.31	45.33	**36.18**	596
	0.5	3.31	3.35	**0.66**	49.19	115.52	**16.33**	12.82	18.70	**2.88**	596
LME-32K	0.3	76.95	69.83	**2.02**	958.58	1844.80	**752.79**	215.96	290.73	**93.22**	643
	0.5	41.63	34.77	**1.25**	303.07	316.88	**7.50**	91.23	91.74	**2.55**	643
Norm. Geo. Mean (%)		100.00	90.14	**8.89**	100.00	112.88	**43.18**	100.00	101.65	**33.63**	–

KVReclaim consistently reduces spill latency across the evaluated workloads and KV-residency budgets. Based on the normalized geometric mean, it reduces spill E2E P50 latency by 91.1% and 90.1% over *CachedAttention* and *FlashGen*, respectively. The reductions in P95 latency are 56.8% and 61.7%, while the mean latency decreases by 66.4% and 66.9%. The improvement is particularly pronounced at P50, although substantial reductions are also observed at the tail. We next examine the reclamation activity associated with these spill operations.

B.2 Space-Reclamation Efficiency

To characterize the storage-side reclamation behavior, Table 5 reports the number of block erase operations performed by each system. Whenever reclamation is triggered, *KVReclaim* requires fewer erase operations than both baselines. Across all configurations, it performs 10,176 block erases, compared with 31,096 for *FlashGen* and 32,504 for *CachedAttention*. This corresponds to reductions of 67.3% and 68.7%, respectively. Several Qwen configurations incur no block erase in any system and are therefore excluded from the per-configuration reduction calculation.

The reduction is consistent with the interaction between reclamation-friendly placement and conversation-aware reclamation. Placement groups conversations with similar eviction costs, which avoids mixing inexpensive and costly eviction choices within the same candidate zones. Once a zone is selected, conversation-wide invalidation can make the selected zone and additional zones directly resettable. Under the same replay workloads, the resulting layout requires substantially fewer erase operations to sustain storage reclamation, indicating higher space-reclamation efficiency.

B.3 Recovery-Path Breakdown

Finally, we examine how requests are served after their KV is needed again. Table 6 separates recovery requests into DRAM hits, SSD reads, and recomputation. For recomputation, we further distinguish proactive recomputation, where a valid SSD copy remains and the system chooses recomputation, from forced recomputation caused by the absence of a valid stored copy.

Across all 16,960 recovery requests, *KVReclaim* serves 7,550 from DRAM and 7,559 from the SSD, while 1,851 requests are recomputed. Of these recomputations, 360 are proactive and 1,491 are forced. *FlashGen* performs 2,361 recomputations, including 781 proactive and 1,580 forced cases, whereas all 1,839 recomputations in *CachedAttention* are forced. Thus, despite selectively

Table 5: Number of block erase operations under different workloads and KV-residency budgets. CA, FG, and KR denote *CachedAttention*, *FlashGen*, and *KVReclaim*, respectively. Lower is better.

Method	LoCoMo		LME-16K		LME-32K	
	$B = 0.3$	$B = 0.5$	$B = 0.3$	$B = 0.5$	$B = 0.3$	$B = 0.5$
Llama-3.1-8B-Instruct						
CA	4,720	2,080	4,856	3,232	10,976	4,192
FG	4,232	1,832	5,224	2,816	10,072	4,464
KR	**1,080**	**808**	**1,552**	**1,160**	**2,872**	**1,200**
KR vs. CA	77.1%	61.2%	68.0%	64.1%	73.8%	71.4%
KR vs. FG	74.5%	55.9%	70.3%	58.8%	71.5%	73.1%
Qwen2.5-7B-Instruct						
CA	0	0	936	128	1,384	0
FG	0	0	1,104	112	1,240	0
KR	**0**	**0**	**704**	**64**	**736**	**0**
KR vs. CA	–	–	24.8%	50.0%	46.8%	–
KR vs. FG	–	–	36.2%	42.9%	40.6%	–

Table 6: Breakdown of KV recovery paths. Recomp. reports the total number of recomputations, with proactive and forced recomputations shown in parentheses as P/F. CA, FG, and KR denote *CachedAttention*, *FlashGen*, and *KVReclaim*, respectively.

Benchmark	B	Recovery #	CA			FG			**KR**		
			DRAM	SSD	Recomp. (P/F)	DRAM	SSD	Recomp. (P/F)	DRAM	SSD	Recomp. (P/F)
Llama-3.1-8B-Instruct											
LoCoMo	0.3	3001	668	1626	707 (0/707)	1059	1124	818 (338/480)	**1047**	**1308**	**646 (108/538)**
	0.5	3001	934	1807	260 (0/260)	1292	1171	538 (273/265)	**1278**	**1475**	**248 (123/125)**
LME-16K	0.3	596	121	245	230 (0/230)	148	154	294 (57/237)	**77**	**215**	**304 (31/273)**
	0.5	596	151	259	186 (0/186)	217	173	206 (63/143)	**122**	**257**	**217 (51/166)**
LME-32K	0.3	643	144	338	161 (0/161)	155	310	178 (17/161)	**113**	**379**	**151 (12/139)**
	0.5	643	152	299	192 (0/192)	209	270	164 (14/150)	**124**	**365**	**154 (6/148)**
Qwen2.5-7B-Instruct											
LoCoMo	0.3	3001	1485	1516	0 (0/0)	1711	1290	0 (0/0)	**1731**	**1270**	**0 (0/0)**
	0.5	3001	1966	1035	0 (0/0)	2191	810	0 (0/0)	**2034**	**967**	**0 (0/0)**
LME-16K	0.3	596	226	322	48 (0/48)	345	183	68 (15/53)	**215**	**313**	**68 (20/48)**
	0.5	596	332	264	0 (0/0)	420	172	4 (4/0)	**250**	**337**	**9 (9/0)**
LME-32K	0.3	643	195	393	55 (0/55)	346	206	91 (0/91)	**243**	**346**	**54 (0/54)**
	0.5	643	272	371	0 (0/0)	374	269	0 (0/0)	**316**	**327**	**0 (0/0)**
Total	–	**16,960**	6,646	8,475	1,839 (0/1,839)	8,467	6,132	2,361 (781/1,580)	**7,550**	**7,559**	**1,851 (360/1,491)**

discarding cached KV during reclamation, *KVReclaim* does not increase forced recomputation. Its forced recomputation count is 5.6% lower than *FlashGen* and 18.9% lower than *CachedAttention*.

The 360 proactive recomputations also show that the recovery policy does not simply favor recomputation whenever an SSD copy is available. Instead, the SSD and recomputation paths are selected according to their estimated runtime costs. This breakdown complements the reclamation results by showing that the reduction in storage-management overhead is not obtained by indiscriminately discarding KV and shifting the resulting work to the GPU.