STRIDEKV: ACCELERATING BATCHED LONG-CONTEXT DECODING VIA PERIODIC REUSE OF RETRIEVED KV CACHE

Anonymous authors
Paper under double-blind review

ABSTRACT

Query-aware KV cache retrieval enables long-context inference by keeping the full KV cache in CPU memory and maintaining only a small GPU-resident working set of entries relevant to each query, but repeatedly selecting and loading these entries can dominate decoding time. We study how often this retrieval needs to occur in synchronous batched decoding. A natural approach is adaptive: reuse the previous working set and perform a correction, an immediate retrieval for the current query, only when the query changes substantially. However, when requests in a batch execute in lockstep, different requests need corrections at different layers and steps, so the batch as a whole must perform corrections far more often than any single request; as the batch grows, the throughput gain of such adaptive schemes vanishes and eventually turns negative. We also observe that queries at adjacent steps select largely overlapping tokens, and that retrieving only every other step yields the same top-1 token as per-step retrieval at 99.4% of decoding steps. Motivated by these findings, we introduce StrideKV, which retrieves relevant KV entries only once every few decoding steps, with all requests in the batch following the same schedule. When retrieval happens, the entries are still selected based on the current query; in the steps between retrievals, the model simply reuses the entries already on the GPU, avoiding both selection and CPU-to-GPU transfer. Across multiple models and context lengths, StrideKV improves decoding throughput on an A100 GPU by up to 66% over ShadowKV, a strong KV offloading method that retrieves at every step, while keeping long-context accuracy close to that of ShadowKV. These results show that StrideKV is a simple and effective way to accelerate batched long-context decoding with KV cache offloading.

1 INTRODUCTION

Long-context language models enable applications such as document question answering, multi-document summarization, and extended conversations. Serving them efficiently, however, is limited by the key–value (KV) cache, which grows linearly with both context length and the number of concurrent requests. Two complementary techniques address this problem: sparse attention lets each decoding step attend to only a small subset of historical tokens, and KV offloading stores the full KV cache in CPU memory, which is far larger than GPU memory. Combined, they allow the GPU to hold only a small *working set* of KV entries relevant to the current step, rather than the entire KV cache, enabling longer contexts and larger batches (Tang et al., 2024; Lee et al., 2024; Sun et al., 2025).

Maintaining this working set also incurs a cost. We use *retrieval* to denote two operations: *selection* identifies historical KV entries relevant to the current query, and *loading* transfers the required CPU-resident entries to the GPU. Even when sparse attention limits the number of tokens used in computation, repeating selection and loading at every step can introduce substantial overhead. This raises a scheduling question for retrieval-based KV management: *does the historical working set need to be retrieved at every decoding step?*

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.

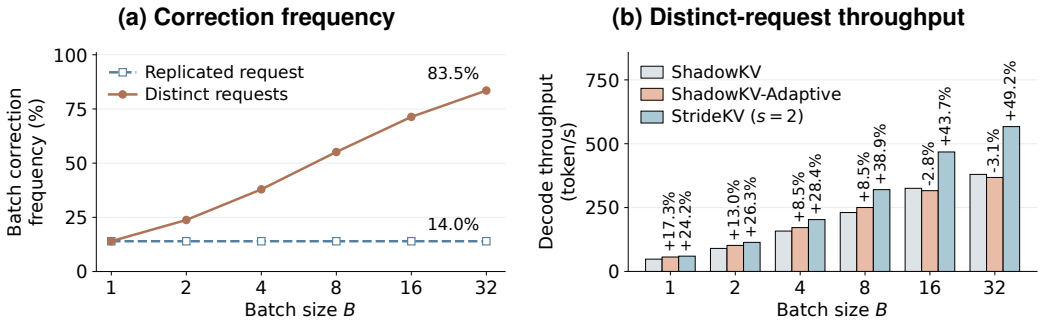

Figure 1: **Adaptive correction under batched decoding.** ShadowKV-Adaptive is a FreeKV-inspired adaptive variant of ShadowKV that performs a correction, an immediate retrieval for the current query, when query similarity falls below $\gamma = 0.8$. **(a) Batch-level correction frequency.** A batch-level correction occurs when at least one request in the batch requires correction at a given layer and decoding step. **Replicated request** batches contain copies of one example, an idealized setting; **Distinct requests** batches contain different examples, as in practical serving. **(b) Decoding throughput with distinct requests.** StrideKV (ours, $s = 2$) retrieves historical KV entries once every two decoding steps on a schedule shared across the batch. Labels give percentage changes relative to ShadowKV at the same batch size; values appear in Table 6 (Appendix A.5). All measurements use Llama-3-8B-1M on an A100, with a context length of 15,360 tokens and a retrieval budget of 2,048.

A natural answer is to retrieve adaptively: reuse the previous working set and perform a *correction*, an immediate retrieval for the current query, only when the query changes substantially. FreeKV (Liu et al., 2026) follows this idea, combining speculative retrieval with query-dependent corrections, optimized data layouts, and streamed transfers. We find, however, that adaptive schemes interact poorly with batched decoding. Requests in a batch execute in lockstep, so whenever any single request needs a correction at a given layer and step, the whole batch must wait for it. Since different requests need corrections at different layers and steps, the batch as a whole performs corrections far more often than any individual request: in our measurements, a single request needs a correction at 14% of layers and decoding steps, whereas a batch of 32 requests needs one at 84%, and the throughput gain of adaptive retrieval disappears (Figure 1).

At the same time, we find that the retrieved entries change slowly across decoding steps. When generating an answer, consecutive tokens often rely on the same parts of the context, so queries at adjacent steps select largely overlapping tokens. Moreover, when retrieval is performed only every other step, the model predicts the same top-1 token as with per-step retrieval at 99.4% of decoding steps. A recently retrieved working set therefore remains useful for the next few steps, even as the query changes.

Based on these observations, we introduce StrideKV, which retrieves KV entries only once every s decoding steps, with all requests in the batch following the same schedule. The schedule fixes *when* retrieval occurs, while the current query still determines *which* entries are selected at each retrieval step. In the steps between retrievals, the model simply reuses the working set already on the GPU, skipping both selection and CPU-to-GPU transfer, while all other computation, including attention and the FFN, still runs at every step. Because the schedule is shared, no request in the batch performs retrieval on reuse steps, which avoids the synchronization problem of adaptive schemes; unlike ShadowKV-Adaptive, its gain over ShadowKV grows with batch size (Figure 1(b)).

We implement StrideKV on top of ShadowKV (Sun et al., 2025) and evaluate its throughput and long-context accuracy. With the default $s = 2$, StrideKV improves throughput over ShadowKV by 34–66% on Llama-3-8B-1M and GLM-4-9B-1M with context lengths from 60K to 244K on an A100 GPU, while keeping accuracy close to that of ShadowKV.

Our contributions are:

- We characterize the cost of KV retrieval in batched decoding with KV offloading, showing that retrieval dominates decoding time and that adaptive retrieval loses its benefit as batches of distinct requests grow, because correction decisions of individual requests do not align in lockstep execution.

- We propose StrideKV, a simple retrieval schedule that is shared across all requests in a batch and reuses the GPU-resident working set between retrievals, reducing the frequency of both selection and CPU-to-GPU transfer.
- We evaluate StrideKV on throughput, long-context accuracy, prediction fidelity, and the choice of stride, showing substantial speedups over a strong KV offloading baseline with accuracy close to per-step retrieval.

2 Related Work

Long-context inference and KV cache memory. The KV cache of long-context models (Grattafiori et al., 2024; Team GLM et al., 2024) grows with both context length and batch size, and several lines of work reduce its footprint or access cost. PagedAttention manages KV memory in fixed-size blocks to serve larger batches (Kwon et al., 2023); IO-aware kernels reduce the memory traffic of exact attention (Dao et al., 2022; Dao, 2024); grouped-query attention shares KV heads across query heads (Ainslie et al., 2023); and quantization stores KV states at low precision (Liu et al., 2024; Hooper et al., 2024). FlexGen offloads weights and KV states to CPU memory and disk to raise batched throughput on a single GPU (Sheng et al., 2023). Benchmarks such as LongBench, InfiniteBench, and RULER measure whether such approximations preserve long-context ability (Bai et al., 2024; Zhang et al., 2024; Hsieh et al., 2024). These techniques are largely complementary to how often a retrieval-based method retrieves its working set.

Sparse attention and KV eviction. Early sparse Transformers restrict each token to fixed local, strided, or global patterns (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). KV eviction methods retain selected historical information and discard the rest: heavy hitters and recent tokens in H_2O, attention sinks in StreamingLLM, pivotal tokens whose importance persists over time in Scissorhands, head-specific policies in FastGen, observation-window voting in SnapKV, and layer-dependent budgets in PyramidKV (Zhang et al., 2023; Xiao et al., 2024b; Liu et al., 2023; Ge et al., 2024; Li et al., 2024; Cai et al., 2024). DuoAttention keeps the full cache only for retrieval heads and a streaming cache for the remaining heads (Xiao et al., 2025), while NSA trains hardware-aligned sparse attention natively (Yuan et al., 2025). Because evicted states cannot be recovered, later queries that need them lose access; retrieval methods instead keep the full history accessible.

Query-aware KV retrieval and offloading. Retrieval methods preserve access to a larger history and select a working set for each computation. Quest scores pages using the current query and per-page key summaries (Tang et al., 2024); InfLLM and ArkVale organize the history into blocks or pages with compact representatives, and ArkVale recalls evicted pages from CPU memory (Xiao et al., 2024a; Chen et al., 2024); Loki and SparQ approximate attention scores using low-rank keys or a subset of query components (Singhania et al., 2024; Ribar et al., 2024). RetrievalAttention, PQ-Cache, and MagicPIG keep KV states in CPU memory and locate relevant entries through approximate nearest-neighbor search, product quantization, or locality-sensitive hashing (Liu et al., 2025; Zhang et al., 2025; Chen et al., 2025). InfiniGen prefetches important KV states, and ShadowKV reconstructs selected keys while fetching offloaded values (Lee et al., 2024; Sun et al., 2025). FreeKV combines speculative recall, query-dependent correction, hybrid layouts, and streamed transfers (Liu et al., 2026); DualDecoder predicts future retrieval needs within a ShadowKV-based prefetching system (Liang et al., 2026). These methods mainly improve *how* each retrieval is performed. We inherit ShadowKV's representation and cache-hit reuse, and study how periodic reuse of the retrieved KV working set reduces repeated retrieval and its associated maintenance.

Reusing selection across time or layers. Scissorhands observes that important tokens tend to remain important across generation steps (Liu et al., 2023), and several methods exploit such stability to amortize selection. LServe reuses query-aware page selection over a predefined chunk of decoding steps (Yang et al., 2025b). It keeps the entire quantized KV cache in GPU memory, so this reuse saves page-selector computation in a GPU-resident setting, whereas StrideKV targets KV offloading, where each retrieval round also transfers selected entries from CPU memory. RefreshKV updates a compact cache using intermittent full-context attention during long-form generation (Xu et al., 2025). TokenSelect's selection cache uses query similarity to reuse indices and can skip the selector itself (Wu et al., 2025). Across layers, TidalDecode reuses selected positions from a few

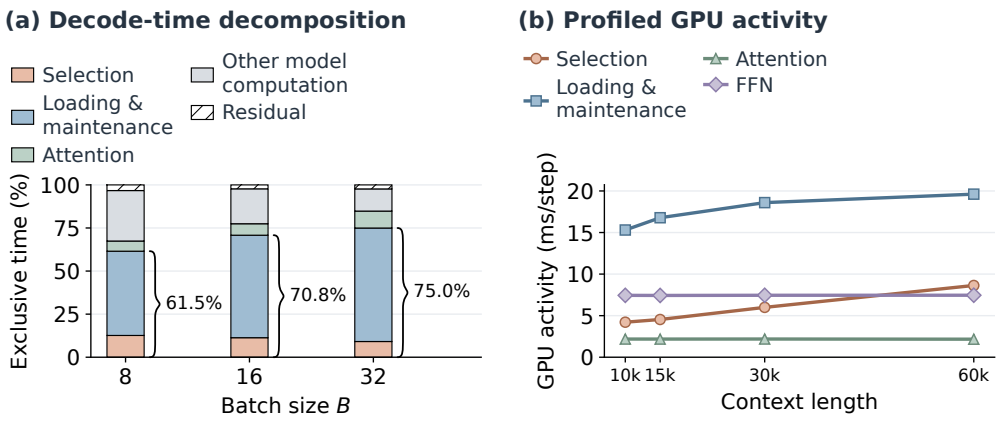

Figure 2: **Retrieval cost of ShadowKV across batch sizes and context lengths.** Both panels profile ShadowKV, which retrieves at every decoding step, with Llama-3-8B-1M on an A100, using a retrieval budget of 2,048 tokens and averaging over 50 decoding steps. **(a)** Percentage of decoding time spent in each component, at a context length of 15,360 tokens. **(b)** Per-step GPU time spent in each component as the context length grows, with $B = 8$.

token-selection layers (Yang et al., 2025a), and OmniKV reuses the context selected by a small set of filter layers for subsequent layers (Hao et al., 2025). These works distinguish selecting *which* states to attend to from choosing *when* to update that selection. StrideKV fixes when to perform a new round of retrieval, retains query-dependent selection on those retrieval steps, and reuses the retrieved KV working set between them, so that in the offloaded setting each reuse step skips both selection and CPU-to-GPU loading. We compare StrideKV with ShadowKV-Adaptive on the same ShadowKV backend, so that differences reflect the retrieval schedule rather than KV representations or kernels. Because the schedule only determines when retrieval occurs, it is complementary to techniques that change how each retrieval is performed.

3 Understanding Retrieval in Batched Decoding

3.1 Retrieval dominates decoding time

ShadowKV reduces GPU memory use through CPU offloading (Sun et al., 2025). At each decoding step, it selects historical KV entries relevant to the current query and makes the selected states available on the GPU, followed by attention and subsequent model computation. We profile ShadowKV to measure the cost of retrieval. Figure 2(a) decomposes 50 decoding steps into disjoint intervals on the dependency-critical path. Selection and loading with associated cache maintenance account for 61.5%, 70.8%, and 75.0% of decode time at $B = 8, 16, 32$, respectively; attention accounts for only 5.9%, 6.7%, and 9.8%.

A fixed retrieval budget bounds the historical set used by attention, while the FFN processes the current token. As context grows, selection examines more historical candidates without a proportional increase in attention or FFN work. At fixed $B = 8$ and budget 2,048, selection activity rises from 4.21 to 8.63 ms per step as context grows from 10K to 60K; attention remains within 2.17–2.19 ms and FFN within 7.43–7.46 ms (Figure 2(b)).

3.2 Distinct requests amplify batch-level correction frequency

To reduce this cost, FreeKV exploits similarity between consecutive queries to overlap retrieval with model computation (Liu et al., 2026). For KV heads that do not require correction, the current query attends to entries prepared at the previous step, while retrieval based on the current query prepares entries for the next step. When query similarity falls below a threshold, the affected heads retrieve the current selection before attention.

To examine this schedule in batched decoding, we incorporate FreeKV-inspired speculative retrieval and query-dependent correction into the ShadowKV backend. We refer to this implementation as **ShadowKV-Adaptive**.

For experimental evaluation, FreeKV forms batches by replicating a single input. This batch construction represents a highly idealized experimental setting. Because the replicas have identical inputs, their corrections tend to occur at the same layers and decoding steps. Such batches of identical requests are rare in practical inference services, where batches typically contain distinct requests with different correction timings. In synchronous batched execution, attention at the current layer must wait for the required corrections to finish. Requests that need no correction therefore cannot independently advance past this dependency.

Let p denote the probability that a single request requires correction at a given decoding step and layer, and let q_B denote the probability that at least one request in a batch of size B requires correction at that step and layer. Assuming independent correction events across requests, each with probability p,

$$q_B = 1 - (1 - p)^B. \tag{1}$$

For $p = 0.1$, $q_{32} = 96.6\%$. Replicas with identical events instead have $q_B = p$.

We measure ShadowKV-Adaptive on 32 QA2 requests at $B \in \{1, 2, 4, 8, 16, 32\}$. We use $\gamma = 0.8$, a context length of 15,360 tokens, and a retrieval budget of 2,048. Figure 1(a) shows that replicated-request correction frequency remains near 14.0%, whereas distinct-request correction frequency increases from 14.0% at $B = 1$ to 83.5% at $B = 32$. Thus, as distinct-request batches grow, even a low per-request correction rate can lead to frequent batch-level corrections.

3.3 Adaptive throughput gains diminish with batch size

To determine whether the increased correction frequency translates into a performance cost, we compare the decoding throughput of ShadowKV-Adaptive and ShadowKV on distinct-request batches. Figure 1(b) reports throughput across batch sizes, with percentage changes relative to ShadowKV. ShadowKV-Adaptive improves throughput over ShadowKV by 17.31% at $B = 1$, demonstrating a benefit from speculative retrieval and overlap in the single-request setting. However, this advantage diminishes as distinct-request batches grow: the gain decreases to 13.02% at $B = 2$ and approximately 8.5% at $B = 4$ and $B = 8$, then becomes negative at $B = 16$ (-2.83%) and $B = 32$ (-3.14%). Together with Figure 1(a), these results show that sparse request-level corrections do not translate into a sustained throughput advantage for ShadowKV-Adaptive in larger synchronous batches.

3.4 Retrieved working sets change slowly

Neighboring queries select overlapping token sets. Consecutive generated tokens can depend on the same portions of a long context. Let $\mathcal{A}_t$ be the complete set of token positions selected for attention by one KV head at decoding step t. We measure retention at lag Δ as

$$\rho_{t,\Delta} = \frac{|\mathcal{A}_t \cap \mathcal{A}_{t+\Delta}|}{|\mathcal{A}_t|}. \tag{2}$$

This is the fraction of positions selected at step t that remain selected at step $t + \Delta$. Across 32 requests, 32 layers, 8 KV heads, and 50 anchor steps, we average within each request and then weight requests equally. Adjacent queries retain 79.11% of selected positions on average; retention remains above 71% at a four-step lag (Table 1). This persistence suggests an opportunity to reuse a previously retrieved working set across nearby steps.

Short reuse intervals introduce small prediction discrepancies. High selected-token overlap alone does not establish prediction fidelity. We compare per-step retrieval, which serves as the reference, with retrieving only once every s decoding steps and reusing the retrieved working set in between, for $s = 2$ and $s = 3$, on 32 QA2 requests. All settings consume the same reference-generated token prefix while their hidden and KV states evolve independently, so reuse errors can accumulate (Appendix A.4). Table 2 reports top-1 agreement, KL divergence, and total variation (TV) relative to the reference.

Table 1: **Selected-token retention across decoding steps.** Mean retention of the complete selected token set, averaged within each request and then equally across the 32 requests.

Lag Δ (steps)	1	2	3	4
Retention (%)	79.11	75.09	72.94	71.27

Table 2: **Fixed-reuse fidelity on QA2 under a common token prefix.** The 32 requests contain 1,856 valid prediction positions. KL and TV are request-balanced means.

Policy	Top-1 (%)	KL	TV
Fixed $s = 2$	99.41	0.00147	0.00831
Fixed $s = 3$	99.25	0.00292	0.01137

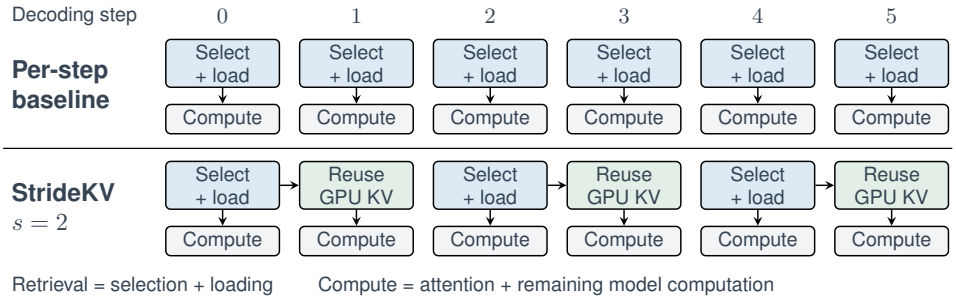

Figure 3: **StrideKV retrieves periodically and reuses the resulting KV states on the GPU.** Each column is a decoding step. The per-step baseline (ShadowKV) retrieves at every step. With $s = 2$, StrideKV retrieves at steps $0, 2, 4, \ldots$ and reuses the selected positions and their GPU-resident KV states at the intervening steps, skipping both selection and loading. All requests in a batch follow the same schedule, so no request retrieves on reuse steps.

The small mean discrepancies support short reuse intervals in this cohort, while their increase at $s = 3$ identifies the cost of longer reuse. Section 5.1 evaluates task accuracy under free generation.

Together, these observations point to a batch-friendly alternative to request-specific correction. Retrieval dominates decoding time, request-specific corrections do not align across distinct requests in a synchronous batch, and a recently retrieved working set remains useful for the next few steps. Section 4 builds StrideKV on these observations.

4 STRIDEKV

4.1 PERIODIC RETRIEVAL AND GPU-RESIDENT REUSE

The per-step baseline retrieves a working set for every generated token, executing both selection and loading. StrideKV instead retrieves once every s steps and retains the resulting working set on the GPU between retrievals (Figure 3). Our implementation builds on ShadowKV (Sun et al., 2025).

Working-set update. Let t be a zero-based decoding step and $\mathcal{W}_t$ the retrieved historical working set at one attention layer, including both selected token positions and their key and value states stored in GPU memory. For stride $s \geq 1$,

$$\mathcal{W}_t = \begin{cases} \mathrm{Retrieve}(q_t), & t \bmod s = 0, \\ \mathcal{W}_{t-1}, & \text{otherwise}, \end{cases} \tag{3}$$

where q_t is the current query. The operator $\mathrm{Retrieve}$ performs selection and loading, returning historical KV states on the GPU; selected entries already available there can be retained. Step $t = 0$ initializes $\mathcal{W}_0$. On reuse steps, $\mathcal{W}_t = \mathcal{W}_{t-1}$ retains both positions and states, so *neither historical selection nor historical loading is performed.*

A shared retrieval clock. Every request follows the same stride: $s = 2$ alternates retrieval and reuse, while $s = 3$ follows each retrieval with two reuse steps. This alignment creates common reuse steps at which no request performs historical retrieval. Figure 7(b) in Appendix B illustrates this alignment across requests.

Per-token computation. We divide the KV cache used for attention into two sets: the *retrieved working set* $\mathcal{W}_t$, which stores KV entries selected from the historical context and loaded onto the GPU, and the *recent set* $\mathcal{R}_t$, which accumulates KV entries appended during the current generation. At each decoding step, the model first computes the current query and appends the current token's KV entries to the recent set. Following the shared retrieval schedule, it then either updates the retrieved working set by selecting and loading historical KV entries or reuses the existing working set. The model performs attention over both sets, followed by the remaining model computation, including the FFN. Figure 7(a) in Appendix B illustrates this execution flow.

4.2 Retrieval cost and the stride trade-off

Over T decoding steps, each layer performs $\lceil T/s \rceil$ retrieval rounds instead of T. Every skipped round eliminates both historical selection and loading, while mandatory per-token operations still run T times.

Let f be the fraction of baseline decoding time attributable to historical selection, loading, and associated cache maintenance skipped on reuse steps. All mandatory operations, including new-token KV appends, belong to the remaining fraction $1-f$. With unchanged per-retrieval and per-step computation costs in a serial-cost model, the asymptotic ideal speedup is

$$S_{\text{ideal}}(s) = \left[(1-f) + f/s\right]^{-1}. \tag{4}$$

This model links the historical retrieval shares measured in Section 3.1 to the opportunity for periodic reuse. Actual gains also depend on execution overhead, cache hits, and overlap. Increasing s makes the historical selection at most $s-1$ steps old, trading fewer retrievals for potentially greater prediction error. Setting $s = 1$ recovers per-step retrieval. We use $s = 2$ by default and compare $s = 1, 2, 3$ in the accuracy and throughput ablations.

5 Experiments

We evaluate StrideKV in terms of task accuracy and decoding throughput, followed by an ablation of the retrieval stride. Detailed settings, including model checkpoints, context lengths, retrieval budgets, and the FreeKV configuration, appear in Appendix A.6.

5.1 Accuracy evaluation

We evaluate accuracy on ten tasks from RULER (Hsieh et al., 2024) at 128K context using Llama-3-8B-1M, GLM-4-9B-1M, and Llama-3.1-8B. Table 3 reports task scores and their averages, comparing full attention, FreeKV, ShadowKV, and StrideKV with the default stride $s = 2$. Full-attention results are taken from the ShadowKV paper (Sun et al., 2025). The remaining results use actual CPU offloading: ShadowKV, FreeKV, and StrideKV store offloaded KV entries in CPU memory and transfer required entries to the GPU on demand, rather than emulating CPU storage in GPU memory.

As shown in Table 3, StrideKV preserves accuracy close to that of ShadowKV while halving the frequency of historical KV retrieval. With the default stride $s = 2$, the average score changes by $+0.29$, -0.80, and -1.44 percentage points across the three models, corresponding to a mean decrease of only 0.65 points. On Llama-3-8B-1M, StrideKV reaches 86.05, slightly above ShadowKV's 85.76 and close to full attention's 86.68. These results show that selecting and loading historical KV entries at every decoding step is not required to maintain high accuracy on these benchmarks.

Under the evaluated configurations with actual CPU offloading, StrideKV also achieves higher average scores than FreeKV on all three models, by 7.68, 6.15, and 6.99 percentage points, respectively. Simple periodic reuse thus maintains competitive task performance in a CPU-offload setting, providing a quality foundation for the throughput gains from less frequent retrieval.

5.2 Throughput evaluation

We evaluate decoding throughput, measured in tokens per second, on a single A100 GPU using Llama-3-8B-1M and GLM-4-9B-1M at context lengths of 60K, 122K, and 244K. Each method uses

Table 3: **RULER accuracy at 128K context.** Scores are reported as percentages, and Avg. denotes the average across the ten tasks. StrideKV uses $s = 2$.

Method	S1	S2	MK1	MK2	MQ	MV	QA-1	QA-2	VT	FWE	Avg.
Llama-3-8B-1M											
Full attention	100.00	100.00	98.96	98.96	98.96	95.57	75.00	48.96	78.54	71.85	86.68
FreeKV	100.00	97.92	93.75	62.50	87.50	75.26	69.79	44.79	80.00	72.22	78.37
ShadowKV	100.00	100.00	100.00	97.92	98.70	93.23	69.79	51.04	76.04	70.83	85.76
StrideKV ($s = 2$)	100.00	100.00	100.00	96.88	97.66	92.19	70.83	52.08	75.83	75.00	86.05
GLM-4-9B-1M											
Full attention	100.00	100.00	94.79	87.50	99.74	93.75	67.71	55.21	97.29	72.22	86.82
FreeKV	100.00	94.79	86.46	58.33	96.61	72.14	58.33	53.13	96.46	76.04	79.23
ShadowKV	100.00	100.00	94.79	85.42	99.74	89.84	66.67	56.25	97.92	71.18	86.18
StrideKV ($s = 2$)	100.00	100.00	94.79	84.38	99.74	88.54	65.63	55.21	97.08	68.40	85.38
Llama-3.1-8B											
Full attention	100.00	100.00	98.96	91.67	98.96	95.31	82.29	47.92	68.96	71.18	85.53
FreeKV	100.00	98.96	94.79	38.54	91.15	58.85	75.00	44.79	66.67	66.32	73.51
ShadowKV	100.00	98.96	96.88	73.96	96.35	93.49	81.25	51.04	63.96	63.54	81.94
StrideKV ($s = 2$)	100.00	98.96	96.88	66.67	96.88	90.89	81.25	48.96	62.71	61.81	80.50

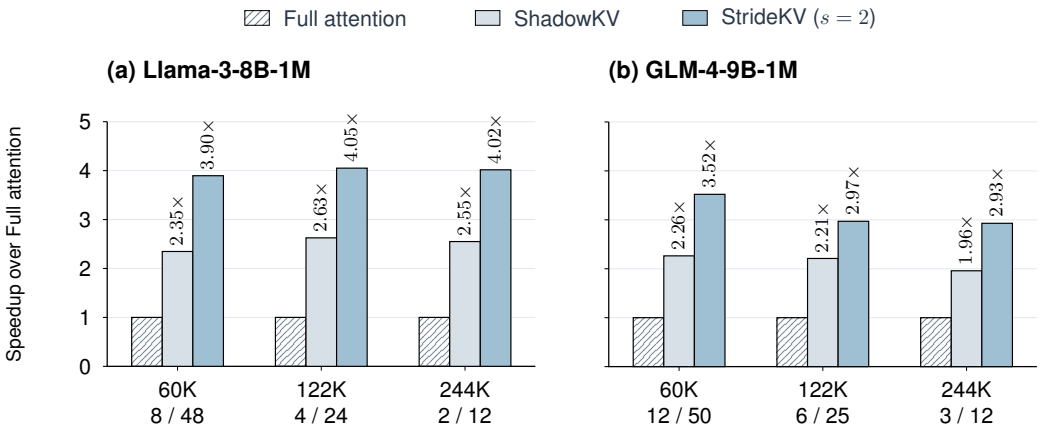

Figure 4: **Decoding speedup over full attention on an A100 GPU.** Each method uses the largest batch size that fits in GPU memory; the numbers below each context length give the batch size for full attention / ShadowKV and StrideKV, which share the same batch size. Bar heights and labels give speedups over full attention. At the same batch size, StrideKV ($s = 2$) improves throughput over ShadowKV by 34.4–66.0%.

the largest batch size that fits in GPU memory; because CPU offloading bounds the GPU-resident KV working set, ShadowKV and StrideKV support larger batches than full attention (Figure 4).

As shown in Figure 4, StrideKV achieves the highest throughput among the three methods in all six configurations, reaching 2.93–4.05× the throughput of full attention. At the three context lengths, StrideKV achieves 627.88, 326.74, and 162.12 tokens/s on Llama-3-8B-1M, and 848.21, 361.04, and 178.16 tokens/s on GLM-4-9B-1M, respectively. These gains reflect the combined effects of sparse computation, larger feasible batches, and reduced historical KV retrieval costs through periodic reuse.

Compared with ShadowKV at the same batch sizes, StrideKV further improves throughput by 34.4–66.0% across both models and all three context lengths, showing that reducing retrieval frequency improves decoding throughput on top of an existing KV-offload method.

5.3 Ablation: effect of stride

We compare $s = 1, 2, 3$ to examine how the historical KV reuse interval affects accuracy and throughput. The reference $s = 1$ corresponds to ShadowKV, which retrieves at every step; $s = 2$

Table 4: **Effect of stride on 128K RULER accuracy.** $s = 1$ is the ShadowKV reference. Changes are percentage points relative to $s = 1$.

Model	$s = 1$	$s = 2$	Δ_2	$s = 3$	Δ_3
Llama-3-8B-1M	85.76	86.05	+0.29	85.20	-0.56
GLM-4-9B-1M	86.18	85.38	-0.80	84.73	-1.45
Llama-3.1-8B	81.94	80.50	-1.44	79.73	-2.21

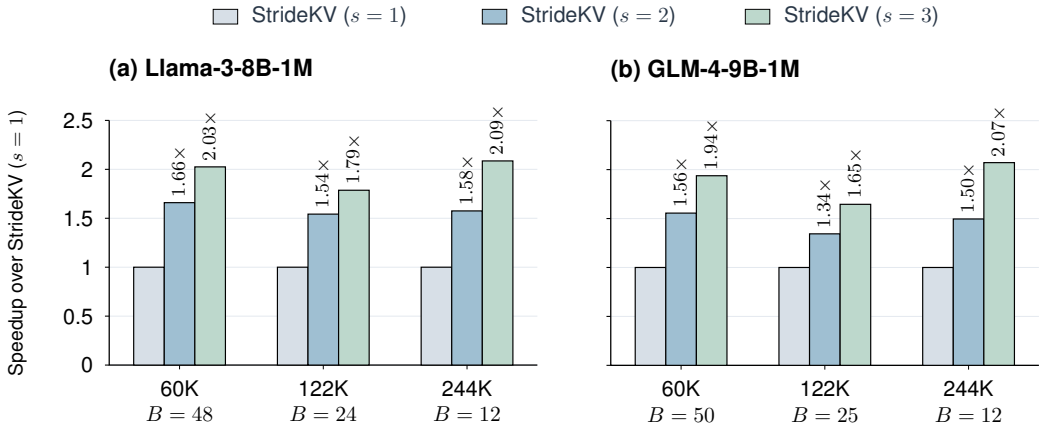

Figure 5: **Effect of stride on throughput.** Bar heights and labels show throughput relative to StrideKV ($s = 1$), within each model–context configuration. The horizontal axis gives context length; B beneath each group is the batch size shared by all three strides.

and $s = 3$ perform selection and loading every two and three decoding steps, respectively, reusing the existing working set in between. Table 4 reports 128K RULER accuracy for three models, while Figure 5 reports throughput for two models at three context lengths. Within each model–context configuration, batch size is held fixed and throughput is normalized to its own $s = 1$ reference.

Taken together, Table 4 and Figure 5 show that $s = 2$ already provides substantial throughput gains with a small change in accuracy. Across the six throughput configurations, $s = 2$ achieves 1.34–1.66× the throughput of $s = 1$, while the mean score across the three models decreases by only 0.65 percentage points in the 128K accuracy evaluation. Short-term reuse therefore reduces historical KV selection and loading costs while retaining task performance close to that of per-step retrieval. Increasing the stride to $s = 3$ further raises throughput in all six configurations, reaching 1.65–2.09× the $s = 1$ reference and improving by another 15.8–38.5% over $s = 2$. The mean accuracy decrease across the three models is 1.41 percentage points relative to $s = 1$, with each model scoring 0.65–0.85 points below its $s = 2$ result. A longer reuse interval thus offers further throughput gains, and the stride provides a simple, direct control over the accuracy–throughput tradeoff. We use $s = 2$ by default to obtain substantial speedups while retaining high accuracy; $s = 3$ offers an alternative for applications that place greater emphasis on throughput.

6 Conclusion

We study the cost of historical KV retrieval in batched long-context decoding. Experiments show that batches of distinct requests amplify batch-level correction frequency, limiting the throughput gains of ShadowKV-Adaptive. Meanwhile, neighboring queries select substantially overlapping KV sets, motivating reuse across decoding steps. We therefore introduce StrideKV, which selects and loads historical KV entries at fixed intervals on a schedule shared across the batch and reuses the loaded KV states between retrieval steps. On an A100 GPU, StrideKV improves decoding throughput by 34.4–66.0% over ShadowKV across two models and context lengths of 60K–244K. It also maintains accuracy close to ShadowKV on the evaluated long-context tasks. These results demonstrate that simple periodic reuse can effectively accelerate batched long-context decoding by reducing the frequency of historical KV selection and loading.

AI USE STATEMENT

In this work, we used generative AI tools to help implement methods and assist with translation. We have not used generative AI tools for the other tasks with required disclosure, namely developing theoretical models or conceptual frameworks, formulating or proving mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, or interpreting results; generating synthetic data sets, cleaning or reformatting data sets, and qualitative data analysis are not applicable to this work. We have reviewed all AI-assisted work: the authors reviewed and tested all AI-assisted code for correctness, and checked all translated text against its intended meaning. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This work studies the efficiency of LLM inference and does not involve human subjects, the collection of new data, or the release of new data sets. All experiments use publicly available models and benchmarks under their respective licenses, and we do not collect or process personal data. StrideKV reduces the cost of long-context decoding without introducing new model capabilities, so it does not add risks of misuse beyond those of the underlying models. Because periodic reuse approximates per-step retrieval, outputs can differ slightly from those of the original model, and deployments in high-stakes settings should validate accuracy on their target tasks. We are not aware of any conflicts of interest related to this work.

REPRODUCIBILITY STATEMENT

We will publicly release the source code of StrideKV upon publication, including our modifications to the ShadowKV implementation, the adaptive baseline, and the scripts used to run all experiments. All models and the RULER benchmark used in this paper are publicly available. Section 4 specifies the retrieval schedule, and Section 5 describes the models, context lengths, batch sizes, and evaluation settings. Appendix A.1 defines the adaptive correction predicate and its execution, and Appendix A.2 details the controlled throughput protocol, including hardware, cache parameters, request construction, warmup, and measurement windows.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024. URL https://arxiv.org/abs/2406.02069.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. ArkVale: Efficient generative LLM inference with

recallable key-value eviction. In *Advances in Neural Information Processing Systems*, volume 37, pp. 113134–113155, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/cd4b49379efac6e84186a3ffce108c37-Abstract-Conference.html.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/6d50d824ae819d5a961c1d8edc15e833-Abstract-Conference.html.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. URL https://arxiv.org/abs/1904.10509.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

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://proceedings.iclr.cc/paper_files/paper/2025/hash/da1131a86ac3c70e0b7cae89c3d4df22-Abstract-Conference.html.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024. URL https://arxiv.org/abs/2404.06654.

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*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172.

USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lee.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Zuning Liang, Zhiyi Yao, Qi Chen, Yuedong Xu, Hao Dai, Zhiqiang Ding, Tongkai Yang, Jinlong Hou, and Yuan Cheng. DualDecoder: Accelerate long context LLM inference by predictive prefetch. *arXiv preprint arXiv:2607.26475*, 2026. doi: 10.48550/arXiv.2607.26475. URL https://arxiv.org/abs/2607.26475.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. In *Advances in Neural Information Processing Systems*, volume 38, pp. 60539–60566, 2025. doi: 10.52202/085713-1816. URL https://arxiv.org/abs/2409.10516.

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.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, volume 36, pp. 52342–52364, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42558–42583. PMLR, 2024. URL https://proceedings.mlr.press/v235/ribar24a.html.

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. URL https://proceedings.mlr.press/v202/sheng23a.html.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, pp. 16692–16723, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/1e027da6bec9ceb2ec37951ceeccae93-Abstract-Conference.html.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57355–57373. PMLR, 2025. URL https://proceedings.mlr.press/v267/sun25b.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, et al. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024. URL https://arxiv.org/abs/2406.12793.

Wei Wu, Zhuoshi Pan, Kun Fu, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Zheng Wang, and Hui Xiong. TokenSelect: Efficient long-context inference and length extrapolation for LLMs via dynamic token-level KV cache selection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21264–21281. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1079. URL https://aclanthology.org/2025.emnlp-main.1079/.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119638–119661, 2024a. URL https://papers.nips.cc/paper_files/paper/2024/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=NG7sS51zVF.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5c1ddd2e59df46fd2aa85c833b1b36ed-Abstract-Conference.html.

Fangyuan Xu, Tanya Goyal, and Eunsol Choi. RefreshKV: Updating small KV cache during long-form generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24878–24893. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1211. URL https://aclanthology.org/2025.acl-long.1211/.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. TidalDecode: Fast and accurate LLM decoding with position persistent sparse attention. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/11440c427f0f76f191ac06b50d7a2517-Abstract-Conference.html.

Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. LServe: Efficient long-sequence LLM serving with unified sparse attention. In *Proceedings of Machine Learning and Systems*, volume 7, 2025b. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/cc8c6b9d89f7a898a29f58869b238e46-Abstract-Conference.html.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird:

Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297, 2020. URL https://papers.nips.cc/paper_files/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product quantization-based KVCache for long context LLM inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025. doi: 10.1145/3725338. URL https://doi.org/10.1145/3725338.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.814. URL https://aclanthology.org/2024.acl-long.814/.

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. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

A Additional Protocol and Evidence

A.1 Adaptive predicate and execution

For each layer and request, let $q_{t,g}$ be the current post-rotation query of query head g. The model has 32 query heads grouped into 8 KV heads, with four query heads $\mathcal{G}(h)$ sharing each KV head. The adaptive correction predicate is

$$M_{i,t,\ell,h} = \mathbb{1}\left\{ \frac{1}{|\mathcal{G}(h)|} \sum_{g \in \mathcal{G}(h)} \cos(q_{i,t,\ell,g}, q_{i,t-1,\ell,g}) < \gamma \right\}. \tag{5}$$

The first step always initializes the working set. Similarities are evaluated in float32 and the comparison is strict. Query-head threshold counts are auxiliary; the actual correction mask uses the average over the query-head group. A larger γ makes this predicate more likely to request a correction for the same queries.

The optimized device path selects history every layer-step. Before attention, masked correction updates the affected heads from the current selection. After attention, complementary speculation updates the remaining heads on a separate stream. The next use of that layer's cache waits on the recorded dependency event. GPU-resident masks avoid a per-step host `any().item()` decision. CUDA graphs and persistent buffers reduce launches and allocation overhead, but do not imply that inactive graph submissions cost zero. Cache-hit matching can also avoid missing-entry reconstruction and CPU-to-GPU loading.

These semantics differ from reusing the same selection indefinitely until a threshold is crossed. In particular, the previous query is updated every step because complementary speculation prepares the current selection for the next step. A true-skip selection cache, such as TokenSelect, requires its own comparison and update semantics; replacing ShadowKV-Adaptive with a naive host gate would conflate policy and implementation costs. ShadowKV-Adaptive retains ShadowKV's representation and is not the complete FreeKV system.

A.2 Matched batch-composition protocol

The controlled throughput measurements use Llama-3-8B-Instruct-Gradient-1048k in BF16 on an NVIDIA A100, with context 15,360, retrieval budget 2,048, rank 160, chunk size 8, and seed 42. We compare unchanged per-step ShadowKV, ShadowKV-Adaptive with $\gamma = 0.8$, and synchronized fixed $s = 2$. The predicate and correction/speculation semantics are unchanged across batch sizes (Appendix A.1).

Each $B \in \{1, 2, 4, 8, 16, 32\}$ is measured in a separate, sequentially launched process with a fresh model load and prefill. Within that process, one $B = 32$ prefill supplies a bank of request-dependent states that is restored into independent, physically B-sized cache slots. The three methods start from the same restored state and consume the same immutable reference token prefixes. Initial-state matching is within each process; fresh prefills need not be bitwise identical across processes. Distinct batches partition the same 32 QA2 explanatory requests into $32/B$ groups. Replicated batches contain B copies of one request and rotate over all 32 requests. The two compositions are interleaved, and every base request has equal coverage within each composition. At $B = 1$, the compositions share the same measured records. There are 223 physical groups in total.

Each policy has one uninstrumented timing window per group, following eight warmup steps and covering 50 decoding steps with a final asynchronous drain. Method order rotates across groups. Model loading, prefill, restoration, graph construction, and warmup are excluded. Throughput is total physical generated tokens divided by total measured wall time; each gain uses the baseline at the same batch size and composition. The audit checks raw-window aggregation, population coverage, source and prefix identities, cache restoration, independent physical storage, and graph replay counts. All 669 windows are retained. There is one independent process per batch size, rather than repeated processes at each batch size; we do not report confidence intervals.

Correction frequencies in Figure 1(a) and Table 5 come from a separate experiment with the same requests and reference prefixes. That experiment restores one shared prefill bank across batch sizes

Table 5: **Correction events in the batch-composition sweep.** Rep. and Dist. denote replicated and distinct requests. Request and KV-head counts include each base request once. Corresponding masks match across compositions at every fixed B. Initialization and warmup are excluded.

	Batch-any (%)		Request corrections		KV-head corrections	
B	Rep.	Dist.	Count	Rate (%)	Count	Rate (%)
1	13.98	13.98	7160	13.98	13071	3.19
2	13.98	23.79	7157	13.98	13051	3.19
4	13.98	37.91	7156	13.98	13043	3.18
8	13.98	55.16	7158	13.98	13060	3.19
16	13.98	71.34	7158	13.98	13060	3.19
32	13.98	83.50	7156	13.98	13051	3.19

and records masks in dedicated replays. Its cross-composition mask-agreement checks apply to those correction records; the independent-process throughput runs do not record correction masks.

A.3 Correction counts and denominators

Each unique request contributes 50 post-warmup steps, 32 layers, and 8 KV heads. The unique-request denominator is $32 \cdot 50 \cdot 32 = 51{,}200$, and the corresponding KV-head denominator is 409,600. Replicated runs repeat these requests in B slots; the reported request/head frequencies weight the base requests equally. Distinct batches have $(32/B) \cdot 50 \cdot 32$ layer-step sites; replicated batches have $32 \cdot 50 \cdot 32$ sites. Table 5 reports the measured rates and unique-request counts.

At each fixed B, comparing each replicated request with its counterpart in a distinct batch gives zero differences across 51,200 request-level and 409,600 KV-head-level decisions. Replica slots also agree within every replicated group. The $B = 1$ comparison uses identical artifacts. Small count changes across physical batch sizes are preserved; the audit establishes cross-composition agreement at each B, not bitwise equality across different B.

A.4 Prediction-fidelity protocol

The fidelity comparison in Section 3.4 uses 32 QA2 explanatory requests at physical batch size $B = 4$, a context length of 15,360 tokens, a retrieval budget of 2,048, and 58 decoding steps. All settings start from the same prefill snapshot and consume the same reference-generated token prefix; their hidden states and newly generated KV states continue to evolve along that prefix. We exclude positions after the end-of-sequence token and average within each request before averaging across requests. KL denotes $\mathrm{KL}(p_{\mathrm{ref}} \| p_{\mathrm{policy}})$ in nats.

A.5 Paired throughput values

Table 6 gives the values underlying Figures 1(b) and 6. Gains use unrounded timings and may differ slightly from ratios of the rounded token/s values below.

Measured benefit across batch sizes. We evaluate StrideKV with $s = 2$ against per-step ShadowKV at each batch size and composition, using the same experimental setup as Section 3.3.

Figure 6 shows increasing gains across all six tested batch sizes for both compositions. With distinct requests, the improvement is 24.22% at $B = 1$ and 49.25% at $B = 32$; replicated requests reach 49.39% at $B = 32$. Fixed reuse also outperforms ShadowKV-Adaptive at every tested B and composition. Its closely matched gains across compositions show that periodic reuse remains effective when batches contain different requests. These results support reducing retrieval frequency on a shared schedule; they do not require identical correction decisions across requests. Full throughput values appear in Table 6.

A.6 Main experimental setup

We evaluate accuracy on Llama-3-8B-1M, GLM-4-9B-1M, and Llama-3.1-8B, and throughput on the first two models. ShadowKV and StrideKV follow ShadowKV's evaluation configuration

Table 6: **Paired throughput by batch composition.** All throughput values are token/s. Each batch size uses an independent process; gains are relative to ShadowKV in the same row. At $B = 1$, both labels refer to the same measurements.

B	Composition	ShadowKV	ShadowKV-Adaptive	Gain (%)	Fixed $s = 2$	Gain (%)
1	Both	48.11	56.44	+17.31	59.77	+24.22
2	Replicated	92.12	106.48	+15.59	115.85	+25.76
2	Distinct	90.10	101.83	+13.02	113.75	+26.26
4	Replicated	165.48	187.34	+13.21	209.76	+26.76
4	Distinct	158.05	171.54	+8.54	202.95	+28.41
8	Replicated	235.66	276.99	+17.54	324.80	+37.83
8	Distinct	230.54	250.18	+8.52	320.26	+38.92
16	Replicated	327.86	336.34	+2.59	470.32	+43.45
16	Distinct	325.38	316.16	-2.83	467.73	+43.75
32	Replicated	377.23	380.95	+0.98	563.56	+49.39
32	Distinct	379.91	367.97	-3.14	566.99	+49.25

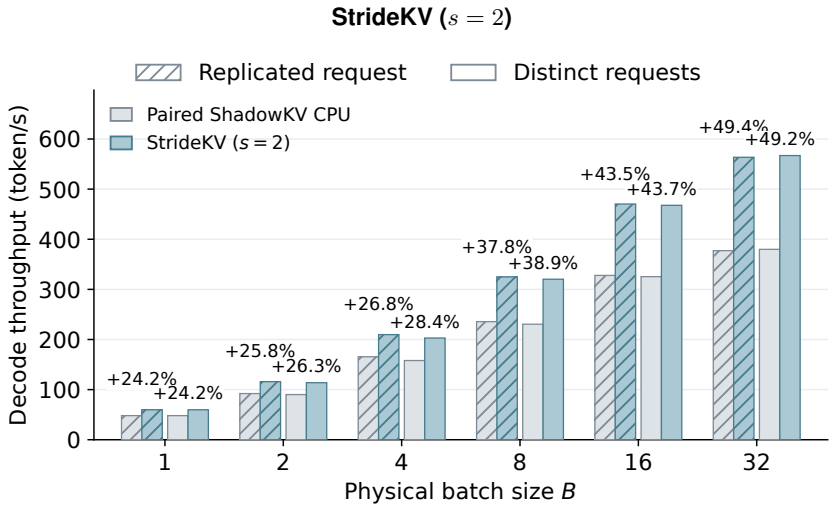

Figure 6: **Fixed-stride throughput with replicated and distinct requests.** StrideKV ($s = 2$) and the paired ShadowKV baseline on an A100, with context 15,360 and retrieval budget 2,048. Hatched bars denote replicated requests; solid bars denote distinct requests. Labels give percentage gains over the baseline at the same batch size and composition.

Table 7: **Main evaluation settings for ShadowKV and StrideKV.** Throughput context lengths and retrieval budgets correspond in the listed order.

Setting	Accuracy	Throughput
Input data	Ten RULER tasks; 96 examples per task	Distinct QA2 examples from the 256K dataset
Context length	131,072 tokens (128K)	61,440 / 124,928 / 249,856 tokens (60K / 122K / 244K)
Retrieval budget	2,048 tokens	1,024 / 2,048 / 4,096 tokens

(Sun et al., 2025): BF16 precision, key-compression rank 160, and 8-token chunks. Table 7 summarizes their main settings. StrideKV uses $s = 2$ by default; the ablation varies only the retrieval stride over $s = 1, 2, 3$, with ShadowKV serving as the $s = 1$ reference.

FreeKV configuration. The FreeKV accuracy evaluation uses the same 128K context length and 96 examples per task, with greedy decoding and seed 42. Its dynamic budget is 2,048 tokens, supplemented by 128 sink and 128 recent tokens, for a total attended capacity of 2,304. It uses 32-token pages, Quest page summaries, the `avgSM` policy, two speculative retrieval steps, threshold $\gamma = 0.8$, and `avg` correction grouping; the initial layer is excluded from sparse retrieval.

B Execution and Scheduling Illustrations

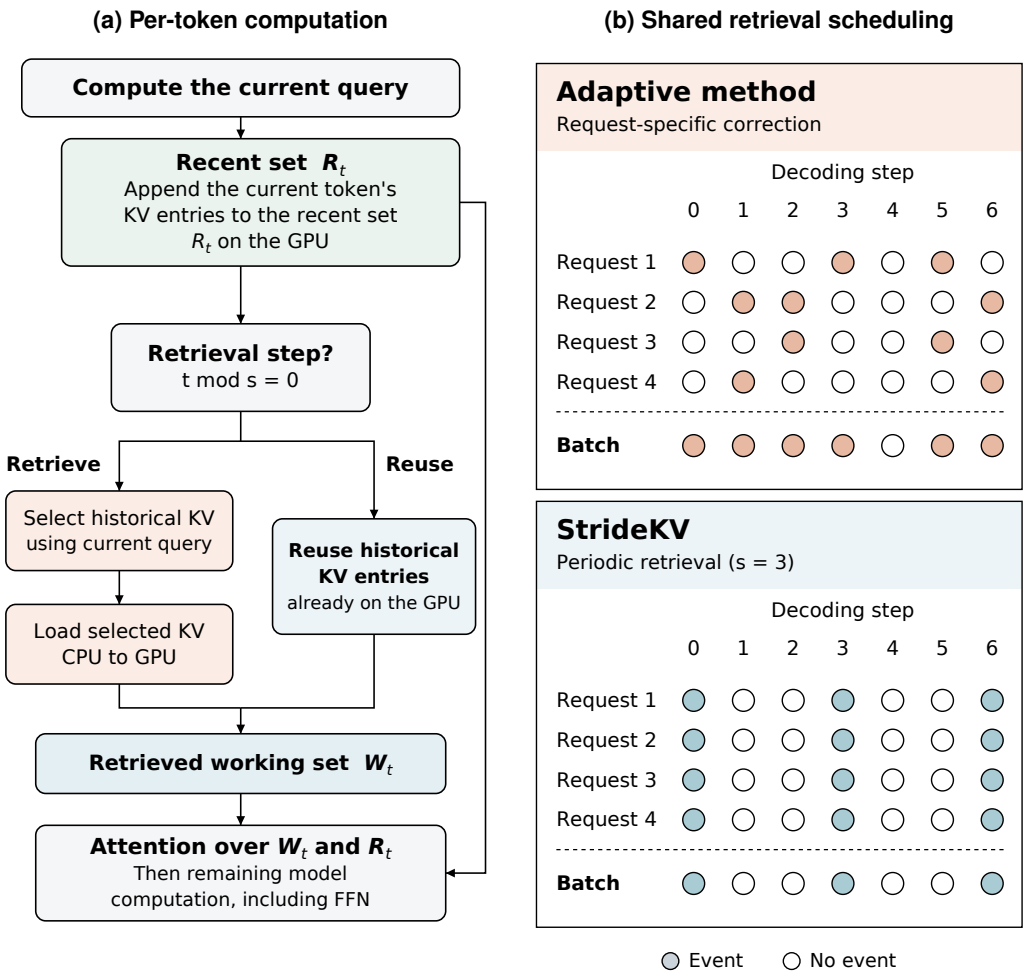

Figure 7: **Per-token execution and shared retrieval scheduling.** (a) Each step computes the current query and appends the current token's KV entries to the recent set $\mathcal{R}_t$. The shared schedule determines whether to update the retrieved working set $\mathcal{W}_t$ through selection and CPU-to-GPU loading or reuse its existing entries. Attention uses both sets, followed by the remaining model computation, including the FFN. (b) Illustrative schedules at one attention layer. The adaptive method has request-specific correction events, whereas StrideKV synchronizes historical retrieval every three steps. Filled circles indicate these events; open circles indicate their absence. The Batch row marks steps with at least one request event.