AsdaKV: Attention-Overlap Driven Semantic KV Retrieval for Long-Context LLMs

Tianming Yan[1,2,3,4] **Kun Yang**[1,2,3,4*] **Tingwang You**[1,3,4] **Kui Ren**[1,2,3,4]
[1]The State Key Laboratory of Blockchain and Data Security, Zhejiang University
[2]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security
[3]School of Cyber Science and Technology, Zhejiang University
[4]College of Computer Science and Technology, Zhejiang University
{ytm,kuny,twyou,kuiren}@zju.edu.cn

Abstract

Long-context inference with large language models is bottlenecked by the linear growth of the KV cache. Existing eviction-based methods reduce memory overhead by irreversibly discarding historical states, while retrieval-based methods preserve the full context but rely on retrieval units defined by tokens, fixed pages, or proxy signals including query-key similarity, clustering, and linguistic rules. Such externally defined units often misalign with the model's actual contextual dependencies during generation. We propose AsdaKV, an adaptive semantic drift-aware KV cache retrieval framework that defines semantic continuity directly from the model's inherent attention behavior. AsdaKV detects semantic drift by measuring the overlap between high-attention historical regions across decoding steps: stable overlap indicates that the active cache still covers the relevant contextual evidence, whereas a sharp overlap drop triggers cache refresh. Guided by this drift signal, AsdaKV organizes historical KV states into variable-length semantic windows, offloads full window KV states to CPU memory, and maintains only lightweight window anchors plus a fixed-budget active KV cache on GPU. During decoding, the current query retrieves relevant windows via anchor matching, while adaptive-stride drift detection and deferred page recall reduce redundant detection, retrieval, and KV reloading overhead. Experiments across diverse tasks and model families show that AsdaKV achieves the best average accuracy among compressed methods on LongBench and RULER under a 1,024-token budget, matches the strongest retrieval baselines on reasoning tasks, and keeps decoding overhead low through adaptive detection and deferred page recall.

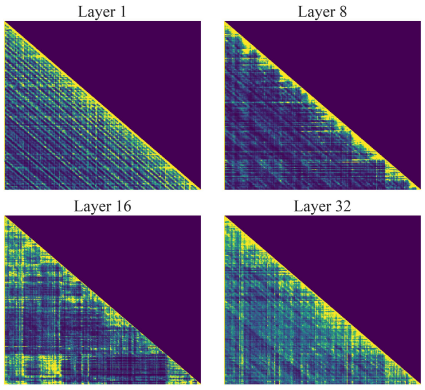
1 Introduction

Long-context large language models (LLMs) [35] are now used for document reasoning, multi-turn interaction and extended generation [1, 3, 12, 15, 24, 37]. Yet as context windows scale to millions of tokens [2, 27, 34], the KV cache becomes a major source of GPU memory use, memory traffic, and attention cost [16, 28, 31]. Efficient KV-cache management is therefore essential for long-context LLM inference.

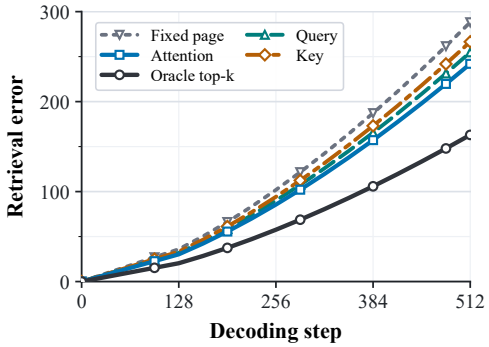
Most KV-compression methods keep selected cache entries under a fixed GPU budget and evict the rest [4, 6, 19, 25, 26, 39, 40]. This is efficient and often strong on long-input tasks, but discarded evidence cannot be recovered when later tokens need it. Retrieval-based KV management avoids this irreversibility by preserving historical states, or recoverable representations, and activating only a relevant subset during decoding. Its key challenge is not only what to retrieve, but when to refresh the active cache. Existing retrieval units, such as tokens, fixed chunks, punctuation-delimited segments,

*Corresponding author: Kun Yang (kuny@zju.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Attention patterns vary across layers.



(b) Attention-overlap refresh gives lower error.

Figure 1: Motivation for attention-overlap retrieval. (a) Attention distributions from different layers; brighter colors indicate larger attention weights. (b) Retrieval-error curves computed from decoding traces. For each decoding step, we evaluate the active cache selected by fixed-page, query similarity, key similarity, attention-overlap refresh, and an oracle top-k attention-support reference, then plot the resulting retrieval error over time.

or embedding-similarity units [18, 21, 30, 33, 38, 41], can be misaligned with the model's evolving dependencies: generated tokens may share overlapping historical evidence, while real semantic transitions can be invisible to fixed segmentation or query/key similarity alone.

As illustrated in Figure 1, attention maps exhibit strong heterogeneity across layers, while attention-support overlap yields a smaller retrieval-error gap than fixed refresh or query/key-similarity proxies. These observations motivate an intrinsically model-driven question: can we define and refresh KV retrieval units according to the attention behavior of the LLM itself?

To address this challenge, we propose AsdaKV, an attention-overlap driven KV cache management framework for efficient long-context inference. AsdaKV directly uses the model's own attention behavior to detect semantic drift during generation: consecutive decoding tokens with highly overlapping attention regions over historical context are grouped into the same semantic window, while a sharp drop in overlap triggers a new window. This allows AsdaKV to identify semantic boundaries from intrinsic attention dynamics rather than fixed chunks, punctuation rules, or embedding-similarity heuristics.

Within each semantic window, contextual demand is relatively stable, so AsdaKV represents a window by a lightweight anchor computed as the mean of its key states and selects the top-k relevant windows through query-anchor matching. Full historical KV states remain in CPU memory, while GPU memory holds only anchors and a small active cache. To reduce latency overhead, AsdaKV further introduces adaptive stride and deferred recall: the former skips unnecessary drift checks during stable periods, while the latter overlaps CPU–GPU KV transfer with attention and FFN computation. The KV cache management is organized into a head-window-page hierarchy, and custom CUDA kernels optimize active-cache updates, page filtering, and metadata operations.

The main contributions of this paper are summarized as follows:

- We introduce attention-overlap drift as a retrieval trigger for semantic changes. The signal is defined from the model's own high-attention historical regions, making refresh decisions depend on realized contextual demand rather than fixed chunks, punctuation rules, or representation-similarity proxies.
- We design adaptive stride and deferred recall mechanisms for overlap-based KV retrieval: adaptive stride skips redundant drift checks during stable semantic intervals, while deferred recall overlaps page loading with decoding so cache refresh does not block the current attention step.
- We implement a CPU-resident KV store, GPU-resident anchors, page-level recall, deferred updates, and fused metadata operations. Experiments on LongBench [3], Needle-in-a-Haystack [13], RULER [10], and Math500 show that AsdaKV is competitive with SOTA KV retrieval baselines and has lower decode latency overhead.

2 Related Work

KV Cache Eviction. KV cache eviction methods aim to reduce the GPU memory usage and attention cost. Early approaches estimate token importance from position priors or historical attention statistics. StreamingLLM [39] preserves attention sink tokens together with a recent sliding window, while H2o [40] retains tokens with high accumulated attention scores. Later methods improve either the estimation of important KV positions or the allocation of compression budgets. SnapKV [19] leverages the attention distribution within an observation window to identify important KV positions. PyramidKV [4] allocates cache budgets across layers according to their different roles, and AdaKV [6] further adapts the budget at the attention-head level. Recent studies also attempt to alleviate the semantic fragmentation caused by token-level eviction. ChunkKV [25] preserves contextual semantics by using semantic chunks as compression units, while SABlock [5] introduces semantic-aware segmentation and adaptive block sizes to constrain KV eviction boundaries. These eviction-based methods are simple and efficient, but their use-and-discard paradigm means that once a KV entry is removed, it cannot be recovered in later decoding steps, risking irreversible information loss.

Retrieval-based KV Cache Management. To address this issue, retrieval-based methods retain the full KV cache or its recoverable representation and dynamically activate only query-relevant KV entries during decoding. InfLLM [38] organizes long contexts into memory units and retrieves relevant units. Quest [33] estimates query-KV page relevance using page-level key summaries to select important KV pages. RetrievalAttention [21] formulates KV access as attention-aware vector retrieval over historical KV entries, enabling fine-grained cache selection. InfiniGen [17] couples KV selection with offloading-based serving by speculatively identifying and prefetching essential KV tokens through lightweight rehearsal. ShadowKV [32] maintains compact low-rank key representations on GPU while offloading value cache to CPU, and FreeKV [23] improves end-to-end retrieval efficiency through speculative retrieval and hybrid CPU-GPU memory management. Some recent studies explore semantic or structure-aware retrieval units. SentenceKV [41] manages KV cache at the sentence level, ClusterKV [22] and Squeezed Attention [9] perform cluster-level KV selection in semantic or key space, LouisKV [36] triggers retrieval at semantic boundaries based on the similarity of consecutive decoding queries, and LycheeCluster [18] further investigates hierarchical KV indexing for efficient long-context inference. Although retrieval-based methods improve information recoverability over eviction-based compression, their boundary detection and selection decisions still rely on fixed structural units, embedding-similarity heuristics, or coarse proxy signals, which struggle to capture real-time semantic drift intrinsic to the model's own attention dynamics during generation. This can result in misaligned retrieval windows or missing critical context at semantic transition points.

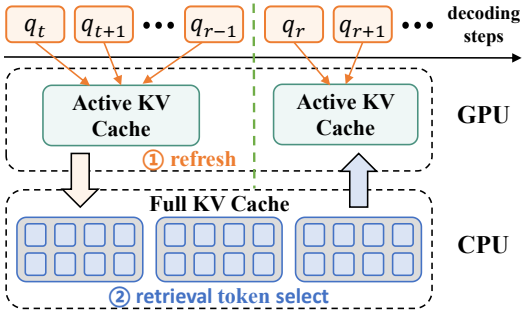


Figure 2: Overview of retrieval-based KV-cache management.

3 Method

3.1 Theoretical Analysis of Attention-Overlap Retrieval

Retrieval paradigm and objective. As illustrated in Figure 2, retrieval-based methods preserve the full historical KV cache in CPU memory and keep only a bounded active subset on GPU. Consider one layer and one attention head at decoding step t, with head dimension d and query, key, and value vectors $q_t, k_j, v_j \in \mathbb{R}^d$. Let $H_t = \{(k_j, v_j)\}_{j=1}^{t-1}$ be the full historical cache and let $S_t \subseteq \{1, \dots, t-1\}$ be the set of historical token positions in the active GPU cache, with $|S_t| \le B$. The full-cache attention distribution is

$$a_t(j) = \frac{\exp(q_t^\top k_j/\sqrt{d})}{\sum_{\ell<t} \exp(q_t^\top k_\ell/\sqrt{d})}, \qquad j < t, \tag{1}$$

and the full output is $o_t = \sum_{j<t} a_t(j) v_j$. A retrieval method computes an approximate output $\widetilde{o}_t(S_t) = \sum_{j\in S_t} a_t(j) v_j / \sum_{j\in S_t} a_t(j)$, renormalizing attention over the nonempty selected set S_t.

Such methods therefore make two decisions: when to refresh the active cache and which historical KV states to load. In this abstraction, B denotes the historical retrieval budget after any fixed sink or recent-token reservations, measured in tokens, and $1 \le k \le B$ is the number of positions used to define the top-k attention-support reference:

$$T_t = \mathrm{TopK}\big(a_t, k\big), \tag{2}$$

where T_t contains the historical token positions with the largest attention weights; TopK returns indices rather than scores. This reference maximizes retained attention mass, but need not minimize output error for fixed value vectors. The objective is to keep the approximation error $\|o_t - \widetilde{o}_t(S_t)\|_2$ small under the GPU budget $|S_t| \le B$. Thus, the key question is when the current active cache no longer matches the model's current contextual demand.

Three refresh signals. Let $r(t)$ be the latest refresh step before t, so the current active cache was built around $T_{r(t)}$, and let $q_{r(t)}, \kappa_{r(t)} \in \mathbb{R}^d$ denote the query and generated-token key stored at that refresh. Existing systems often detect semantic continuity through proxy representation similarity. ClusterKV is representative of key-similarity criteria, which compare the generated-token key states:

$$\Omega_t^k = \cos(\kappa_t, \kappa_{r(t)}), \tag{3}$$

where $\kappa_t \in \mathbb{R}^d$ denotes the key state of the current generated token, distinct from the historical keys k_j, and cos denotes cosine similarity, $\cos(u, v) = u^\top v/(\|u\|_2\|v\|_2)$. FreeKV and LouisKV are representative query-similarity methods, which check whether the current query remains close to the query used at the latest refresh:

$$\Omega_t^q = \cos(q_t, q_{r(t)}). \tag{4}$$

Both signals measure latent-state drift, but neither directly observes which historical positions enter or leave the dominant attention support.

AsdaKV instead measures semantic continuity by comparing the model's realized attention supports:

$$\Omega_t^{\mathrm{attn}} = \frac{|T_t \cap T_{r(t)}|}{|T_t \cup T_{r(t)}|}. \tag{5}$$

In this token-level abstraction, each signal is evaluated at every step and triggers refresh when its continuity score falls below its threshold; otherwise the active cache is reused. A refresh at t sets the reference to t for subsequent steps. The practical window detector and its detection stride are defined in Secs. 3.2–3.3.

Certified error for cache reuse. The retrieval error is governed by the attention mass missed by the active cache. Let

$$\rho_t^k = 1 - \sum_{j \in T_t} a_t(j), \quad \alpha_t^{\max} = \max_{j \in T_t} a_t(j). \tag{6}$$

Here, ρ_t^k is the attention mass outside the current top-k support, $\alpha_t^{\max}$ is its largest attention weight, and $V_{\max}$ below bounds the Euclidean norm of every value vector.

Lemma 3.1 (Support-change error bound under a fixed budget)**.** *For a top-k token-position support with $k \le B$, suppose the active cache satisfies $|S_t| \le B$, contains the previous refreshed support $T_{r(t)}$, and $\|v_j\|_2 \le V_{\max}$ for all j. Then*

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}|T_t \setminus T_{r(t)}|\right). \tag{7}$$

The proof is given in Appendix A.2. The lemma links the error of reusing the active cache to the number of newly required top-k positions. Attention overlap turns this quantity into a threshold certificate.

Lemma 3.2 (Attention-overlap threshold certificate)**.** *Under the assumptions of Lemma 3.1, let $|T_t| = |T_{r(t)}| = k$ and $\tau \in [0, 1]$. If $\Omega_t^{\mathrm{attn}} \ge \tau$, then reusing the active cache satisfies*

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}\frac{k(1-\tau)}{1+\tau}\right). \tag{8}$$

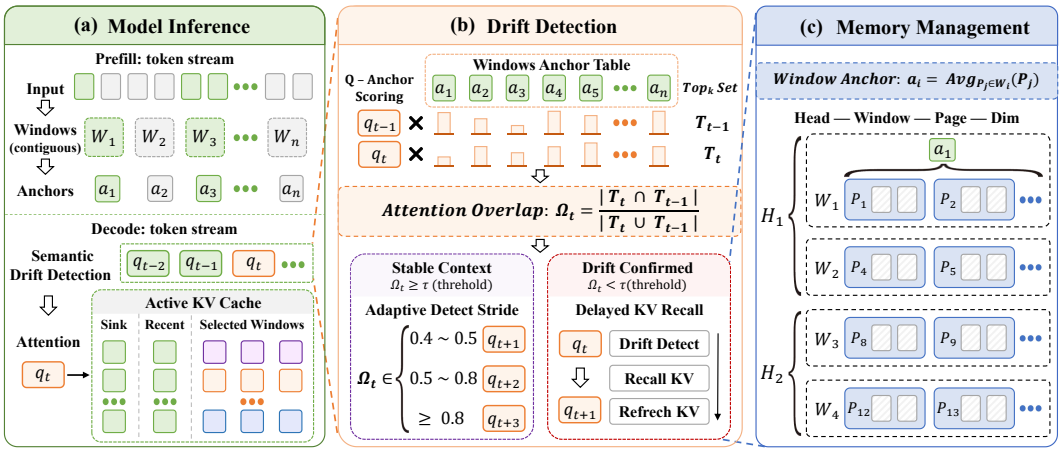


Figure 3: Overview of AsdaKV. AsdaKV first constructs semantic windows and window anchors during prefill, then detects semantic drift during decoding and recalls missing KV pages from CPU memory to the GPU active cache for attention computation.

The certificate follows from the support-change identity

$$|T_t \setminus T_{r(t)}| = \frac{k(1-\Omega_t^{\mathrm{attn}})}{1+\Omega_t^{\mathrm{attn}}} \tag{9}$$

and the fact that $k(1-x)/(1+x)$ decreases on $[0,1]$. Appendices A.3 and A.5 give the derivation. Thus, the token-level rule permits cache reuse only at steps with a certified support-mismatch contribution of at most $2V_{\max}\alpha_t^{\max}k(1-\tau)/(1+\tau)$; falling below τ requests a refresh. Raising τ tightens this contribution, reaching zero at $\tau = 1$. The tail term ρ_t^k accounts for attention mass outside the current top-k support even when the supports agree.

Information needed by similarity thresholds. Query/key cosine thresholds require additional information to certify support retention. For a fixed historical candidate pool, let $s_u(j) = q_u^\top k_j/\sqrt{d}$ for $u \in \{r,t\}$, and let $s_r^{(k)}$ be the old top-k cutoff. A logit perturbation bound Γ_t^a, with $a \in \{q,k\}$, defines $G_a(t) = \{j : |s_r(j) - s_r^{(k)}| \le \Gamma_t^a\}$. The pairing argument in Appendix A.4 yields $|T_t \setminus T_r| \le |G_a(t)|$, and hence

$$\textbf{Query similarity:} \quad \|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}|G_q(t)|\right), \tag{10}$$

$$\textbf{Key similarity:} \quad \|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}|G_k(t)|\right). \tag{11}$$

Evaluating these bounds requires query norms, historical key norms, and the logit margins around the cutoff; key similarity additionally requires a quantitative relation between generated-key drift and query drift. A cosine threshold alone does not provide these quantities. Indeed, query cosine can approach one while the top-k supports are disjoint, and generated-key cosine can equal one in the same construction (Appendix A.5). Attention overlap directly certifies the support-retention condition used to decide whether to reuse the cache. Appendix A.6 connects this token-level certificate to window detection through explicit aggregation errors.

3.2 AsdaKV Algorithm–System Overview

AsdaKV is a hierarchical algorithm-system co-design. As shown in Figure 3(c), AsdaKV organizes the full KV cache as a three-level structure. KV states are stored in CPU memory with HND layout, where KV heads are the leading dimension. This layout matches AsdaKV's per-head retrieval decisions and allows each head to recall the windows without reshaping the full cache. At the logical level, drift detection and KV recall operate on semantic windows. At the physical level, memory management is performed on fixed-size pages. A semantic window can contain a variable number of pages, so AsdaKV keeps the semantic retrieval unit decoupled from the physical transfer unit.

Prefill: building window anchors. AsdaKV first splits the prefill KV cache into fixed-size pages, and then groups adjacent pages into semantic windows according to the attention pattern of the last

L_{obs} prefill queries. These observation queries provide an estimate of the context regions that will be needed at the beginning of decoding. Pages whose estimated attention demand changes smoothly are merged, while sharp changes mark potential window boundaries. Minimum and maximum window sizes are enforced to avoid overly fragmented or overly coarse partitions.

Each semantic window is represented by a compact anchor. For a window W_i that contains pages P_j, where $|P_j|$ is the number of tokens in page P_j, the anchor $\bar{k}_i \in \mathbb{R}^d$ is the average of page-level mean keys:

$$\bar{k}_i = \text{Avg}_{P_j \in W_i} \left(\frac{1}{|P_j|} \sum_{\ell \in P_j} k_\ell \right). \tag{12}$$

Here, Avg averages over the pages in the window. For each KV head, $A_t \in \mathbb{R}^{N_t \times d}$ stacks the N_t available anchors as rows. The anchors stay on GPU for query-window scoring, while full KV pages remain in CPU memory. Appendix E.2 ablates the window-anchor construction, and Appendix F.3 studies the observation length L_{obs}.

From token support to window detection. Appendix A.6 bounds the token–window overlap discrepancy using window-length variation, aggregation error, and anchor-ranking error.

Offline profile calibration. AsdaKV builds a small profile table offline from unlabeled held-out calibration traces. The calibration prompts are grouped into workload profiles that represent different retrieval-pressure regimes, including long-input evidence seeking, aggregation, generation-dominant reasoning, and code generation.

Let $\mathcal{D}_c^{\text{cal}}$ be the calibration prompts assigned to workload profile c. For prompt $x_i \in \mathcal{D}_c^{\text{cal}}$, i is the prompt index and $H_t^{(i)}$ is its historical cache at decode step t. The cutoff t_{warm} excludes warm-up steps before the decode page, anchors, and stride state stabilize. We collect overlap samples only when the historical cache exceeds budget B, so retrieval is needed:

$$\mathcal{T}_i = \{t : \ |H_t^{(i)}| > B, \ t > t_{\text{warm}}\}. \tag{13}$$

This avoids using early short-input reasoning steps where the full historical cache still fits inside the active-cache budget. For long-input tasks, $|H_t^{(i)}| > B$ already holds at the first decoding step. At each valid step, AsdaKV scores the window anchors with the current query and records the selected semantic windows

$$\mathcal{W}_t^{(i)} = \text{TopK}(A_t^{(i)} q_t^{(i)}, K_w). \tag{14}$$

Here, $\mathcal{W}_t^{(i)}$ contains window IDs and K_w is the number of selected windows, distinct from the token-support size k in Sec. 3.1. For GQA scoring, KV head h uses $q_t^{(h)} = |\mathcal{G}_h|^{-1} \sum_{g \in \mathcal{G}_h} q_t^{(g)}$, where $\mathcal{G}_h$ is the group of query heads sharing that KV head. Layer and head indices are otherwise omitted for readability. Let $\mathcal{T}_i^{(512)}$ denote the first at most 512 valid steps in chronological order. One overlap sample is the Jaccard overlap between adjacent selected-window sets:

$$\Omega_t^{(i)} = \frac{|\mathcal{W}_t^{(i)} \cap \mathcal{W}_{t-1}^{(i)}|}{|\mathcal{W}_t^{(i)} \cup \mathcal{W}_{t-1}^{(i)}|}, \qquad \mathcal{O}_c = \{\Omega_t^{(i)} : \ x_i \in \mathcal{D}_c^{\text{cal}}, \ t \in \mathcal{T}_i^{(512)}\}. \tag{15}$$

Thus, each prompt contributes up to 512 contiguous valid overlap samples after the historical cache exceeds the active-cache budget. The profile drift threshold is then calibrated as

$$\tau_c = Q_{\alpha_c}(\mathcal{O}_c), \tag{16}$$

where Q_{α_c} is the empirical α_c-quantile of the collected overlap samples and $\alpha_c \in (0, 1)$ is selected offline for profile c. We also sweep the stride boundaries and store $(\tau_c, b_c^{\text{low}}, b_c^{\text{high}})$ in a profile table, with $0 \le \tau_c \le b_c^{\text{low}} \le b_c^{\text{high}} \le 1$. At runtime, AsdaKV only selects a workload profile and looks up these calibrated values. Appendix F.4 analyzes the offline calibration quantile, and Appendix F.5 discusses the number of valid decode samples used for calibration traces. Appendix E.4 ablates task-specific calibration against a C4-only mismatched profile.

Calibration data and generalization. Appendix E.4 shows that a shared threshold, length-based classification, and online calibration give comparable accuracy, while workload-specific profiles give the best accuracy–latency trade-off.

Decode: drift detection and page recall. During decoding, AsdaKV uses the current query to score window anchors and selects the top-K_w windows for each KV head (the window top-k used in the experiments). A workload tag, or a simple rule based on input length and task template when no tag is provided, chooses a profile c. The selected window set is compared with the previous detected set using attention overlap. If the overlap remains high, AsdaKV reuses the current GPU active cache and may increase the detection stride. If the overlap drops below the profile threshold τ_c, AsdaKV treats the change as a shift in contextual demand and refreshes the active cache. Appendix F.1 reports the sensitivity to top-k.

The refresh is planned at the window level but executed at the page level. The GPU active cache contains fixed sink tokens, recent generated tokens, and recalled historical pages. This design keeps the algorithmic decision semantic, while making the runtime update compatible with fixed-size page allocation and asynchronous CPU–GPU transfer. Appendix E.7 and Appendix F.2 evaluate page-level loading and page-size sensitivity.

3.3 Adaptive Stride for Efficient Drift Detection

We observe that the retrieved semantic windows often stay unchanged for several consecutive decoding steps within a semantically stable interval. This motivates adaptive drift-detection stride. AsdaKV checks drift sparsely when the overlap is stable and returns to fine-grained monitoring near semantic transitions. Let c be the selected workload profile, and retrieve $(\tau_c, b_c^{\text{low}}, b_c^{\text{high}})$ from the offline profile table. Let t_{last} be the previous detection step and δ_t be the current detection stride. At step t, drift detection is performed only when

$$t - t_{\text{last}} \geq \delta_t. \tag{17}$$

Otherwise, AsdaKV skips window detection and reuses the active cache. When detection is performed, AsdaKV compares the current top-K_w window-ID set $\mathcal{W}_t$ with the set from the previous detection step:

$$\bar{\Omega}_t = \frac{|\mathcal{W}_t \cap \mathcal{W}_{t_{\text{last}}}|}{|\mathcal{W}_t \cup \mathcal{W}_{t_{\text{last}}}|}. \tag{18}$$

It then maps the overlap to one of four fixed stride levels:

$$\delta_{t+1} = \begin{cases} 1, & \bar{\Omega}_t < \tau_c, \\ 2, & \tau_c \leq \bar{\Omega}_t < b_c^{\text{low}}, \\ 3, & b_c^{\text{low}} \leq \bar{\Omega}_t < b_c^{\text{high}}, \\ 4, & \bar{\Omega}_t \geq b_c^{\text{high}}. \end{cases} \tag{19}$$

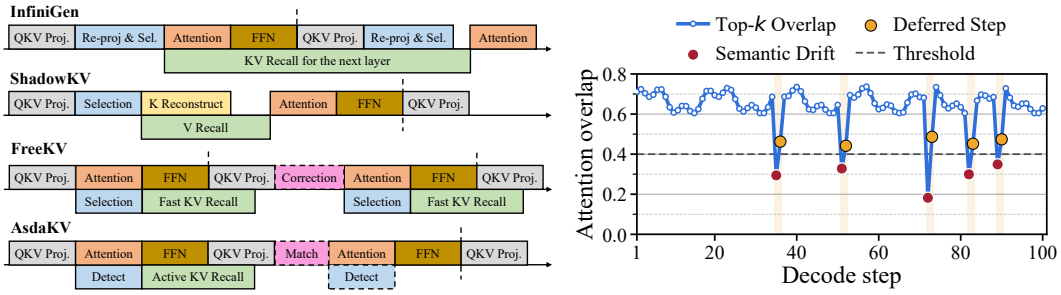
When the overlap falls below τ_c, AsdaKV refreshes the active cache and immediately returns to step-by-step detection. Higher overlap indicates a more stable retrieval state and therefore permits less frequent checks. The two offline-calibrated boundaries b_c^{low} and b_c^{high} separate moderate, high, and very high stability regimes, yielding stride levels 2, 3, and 4 respectively. After each check, AsdaKV stores the current window set and sets $t_{\text{last}} = t$, so each detection compares successive retrieval states and adapts the next checking interval. Appendix E.3 ablates the adaptive-stride schedule.

3.4 Deferred KV Recall for Low-Latency Decoding

AsdaKV further reduces decoding latency with deferred KV recall. In many middle and deep layers, a sudden attention-overlap drop is not followed by an immediate return to high overlap at the next step. The next step often remains near the transition drift state, especially when the drift token corresponds to punctuation or a sentence boundary, and stable overlap appears only after the new semantic unit starts. Figure 4b shows this behavior on NarrativeQA task with a 32K input.

Based on this observation, AsdaKV starts recalling selected semantic-window KV states as soon as drift is detected, but the current attention still uses the active cache already resident on GPU. The recalled pages become visible from the next step, turning recall from a blocking operation into a parallel background update.

Deferred recall reduces average per-layer decode latency from 2.26 to 1.21 ms on Qwen3-8B/A100 (Table 16). As shown in Figure 4a, AsdaKV overlaps drift detection and active KV recall with the decoding pipeline, so CPU–GPU transfer is submitted asynchronously and mostly hidden behind



(a) Inference pipeline.

(b) Attention-overlap trace.

Figure 4: Deferred recall in AsdaKV. (a) The inference pipeline overlaps drift detection and active-cache recall with decoding. The upper path shows decoding, the lower path shows overlap computation. The dashed box indicates that the calculation is not performed at every step. (b) Attention-overlap trace on Llama-3.1-8B-Instruct with a 32K NarrativeQA input, measured at layer 16 and averaged over heads.

subsequent attention and feed-forward computation. Compared with InfiniGen, ShadowKV, and FreeKV, AsdaKV moves less work onto the visible decode path: it keeps only lightweight query-anchor matching on the critical path, and adaptive stride further reduces how often that matching is invoked. Appendix E.5 evaluates the deferred-recall path. We further optimize page-level recall with custom CUDA kernels. Top-k selection and overlap computation are also fused to reduce kernel launches and metadata movement. Appendix E.6 evaluates per-head retrieval.

4 Experiments

4.1 Settings

Backbone LLMs. We conduct the main experiments on Llama-3.1-8B-Instruct and Qwen3-8B. We futher provide complementary experiments on Mistral-7B-Instruct-v0.3 [11] and DeepSeek-R1-Distill-Llama-8B. We also evaluate Llama-3.1-70B-Instruct [7] to examine whether AsdaKV remains effective when scaling to a larger backbone.

Baseline Methods. We compare AsdaKV with representative SOTA KV retrieval methods, including Quest [33], InfiniGen [17], ShadowKV [32], FreeKV [23], ClusterKV [22], SentenceKV [41], and LouisKV [36]. We also include SnapKV [19] as a KV eviction baseline.

Datasets. We evaluate long-input tasks on LongBench [3], Needle-in-a-Haystack (NIAH) [13], and RULER [10] and long-output task on MATH500 [20].

Environment. Unless otherwise stated, experiments use an NVIDIA A100-SXM4-80GB GPU with an Intel Xeon Platinum 8358P CPU, connected via PCIe 4.0 $\times$16, with Ubuntu, Linux kernel 5.15.0, and CUDA 12.8. Additional efficiency experiments use a single NVIDIA H200 NVL. Appendix D reports the A100 and H200 measurements separately, with their respective timing protocols.

4.2 Accuracy Evaluation

Long-input accuracy. We first test whether AsdaKV preserves information from long inputs. We set Top-k to 12 and use the default offline profile calibration for drift detection. On LongBench with a 1,024-token budget, AsdaKV achieves the best average score among the compressed baselines, retaining 99.13% of FullKV performance (Table 1). Additional length-scaling results on RULER are provided in Appendix C.3.

Table 1: LongBench performance with Llama-3.1-8B-Instruct under a 1,024-token budget. Each category reports the mean score over its constituent tasks, and Avg. is averaged over all 16 tasks. The best and second-best compressed methods are marked in **bold** and underlined, respectively.

Method	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synth.	Code	Avg.
FullKV	43.18	43.62	29.38	68.24	53.50	57.89	48.50
SnapKV	42.23	41.98	25.35	66.09	**53.92**	56.29	46.71
Quest	38.42	41.13	25.46	67.79	47.33	53.06	44.95
InfiniGen	41.65	40.79	26.02	**68.42**	46.90	56.18	46.05
ClusterKV	42.13	42.68	28.19	64.21	51.00	**57.68**	46.81
SentenceKV	41.82	42.44	26.62	66.84	53.25	56.86	47.09
ShadowKV	42.58	42.29	28.30	67.22	52.30	55.80	47.34
LouisKV	42.51	41.67	**28.50**	66.77	52.20	57.17	47.32
FreeKV	42.82	43.25	27.65	66.88	53.00	57.26	47.64
AsdaKV	**43.26**	**43.92**	27.83	68.07	52.88	57.16	**48.08**

Reasoning accuracy. We further evaluate whether KV retrieval preserves long-output reasoning ability on Math500 using Qwen3-8B and DeepSeek-R1-Distill-Llama-8B. We set `max_new_tokens` to 12,800 and enable thinking mode for Qwen3-8B; truncated generations without a final answer are scored as incorrect. As shown in Table 2, AsdaKV matches or exceeds the strongest retrieval baseline on both backbones.

4.3 Efficiency Evaluation

We first evaluate raw throughput on Llama-3.1-8B-Instruct under a 1,024-token KV budget. We then analyze end-to-end latency, time-to-first-token (TTFT), and time-per-output-token (TPOT). Since different methods use different attention kernels and runtime backends, latency is compared with the FullKV baseline from each method's own codebase. For throughput and latency measurements, we discard 5 warmup runs and report the average over 5 measured runs. The evaluation covers both 32K-input/128-output long-input decoding and 256-input/12K-output long-output decoding.

Throughput. Figure 5 reports raw throughput across input lengths. AsdaKV scales well with context length, outperforming Quest, InfiniGen and FreeKV from 32K to 128K.

Latency. Figure 6 reports per-implementation normalized latency on A100. Table 9 additionally reports absolute E2E, TTFT, and TPOT in a common HuggingFace runtime: AsdaKV lowers TPOT relative to HF-FullKV and FreeKV, while prefill preparation adds TTFT and E2E cost. Separate H200 experiments in Tables 11–13 show higher AsdaKV throughput during long-output decoding; ShadowKV remains faster in large-batch, long-input settings.

5 Discussion

Limitations. Our empirical conclusions cover the model families, benchmarks, cache budgets, and A100/H200 configurations reported in this paper. The token-level certificate compares each step with the last refresh, while the implementation compares window sets at successive detection steps. Extending the guarantee to sparse window checks and deferred recall remains future work.

Future work. Our implementation focuses on GQA models. Future directions include adapting the retrieval policy to MLA and MQA, reducing the prefill computation of request-local calibration, improving multi-batch throughput through CPU–GPU transfer overlap and batched recall, and integrating with production inference frameworks such as vLLM.

Table 2: Math500 reasoning accuracy (%).

Method	Qwen3-8B	DeepSeek-R1
FullKV	94.4	79.8
InfiniGen	91.0	74.2
ShadowKV	92.8	77.6
LouisKV	**93.2**	77.4
FreeKV	92.4	75.8
AsdaKV	**93.2**	**78.0**

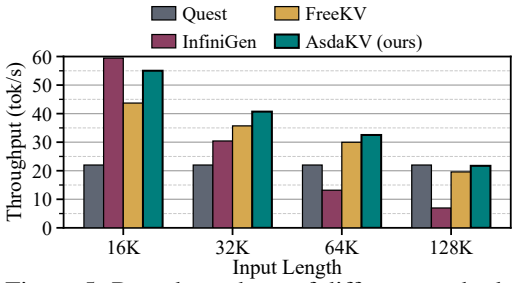


Figure 5: Raw throughput of different methods on A100.

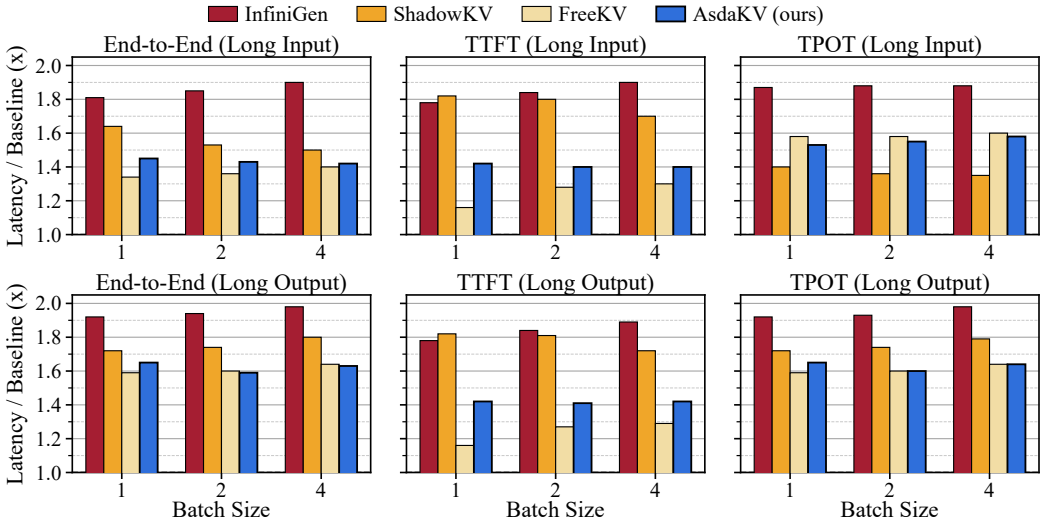


Figure 6: Normalized latency on Llama-3.1-8B-Instruct on A100. Each method is compared against its own FullKV baseline under the same backend. We report end-to-end latency, TTFT, and TPOT under long-input and long-output settings across batch sizes.

6 Conclusion

We presented AsdaKV, an attention-overlap driven KV cache retrieval framework for efficient long-context inference. AsdaKV uses the model's own attention dynamics to identify semantic drift, constructs variable-length semantic windows with lightweight anchors, and refreshes the active cache only when the retrieved historical evidence changes. Combined with offline profile calibration, adaptive drift detection, page-level recall, and deferred CPU–GPU transfer, AsdaKV provides a practical accuracy–efficiency trade-off for retrieval-based KV cache management under tight GPU memory budgets.

Acknowledgments and Disclosure of Funding

We would like to express our sincere gratitude to the area chairs and the anonymous reviewers for their constructive comments and valuable suggestions, which have significantly improved the quality of this paper.

This research was supported in part by the National Key R&D Program of China (Grant No. 2023YFB3105903), the National Natural Science Foundation of China (Grant No. 62372407), and the "Pioneer and Leading Goose" R&D Program of Zhejiang (Grant No. 2025C01089).

References

[1] Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14388–14411, 2024.

[2] Anthropic. Introducing claude 3.5 sonnet. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2024. Accessed: 2026-05-05.

[3] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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)*, pages 3119–3137, 2024.

[4] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2024. *URL https://arxiv. org/abs/2406.02069*, 2024.

[5] Jinhan Chen, Jianchun Liu, Hongli Xu, Xianjun Gao, and Shilong Wang. Sablock: Semantic-aware kv cache eviction with adaptive compression block size. *arXiv preprint arXiv:2510.22556*, 2025.

[6] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024.

[7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[8] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[9] Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32631–32652, 2025.

[10] 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.

[11] Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *ArXiv*, abs/2310.06825, 2023.

[12] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL `https://arxiv.org/abs/2310.06770`.

[13] Greg Kamradt. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023. Accessed: 2026-0-05.

[14] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 TB of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

[15] Wai-Chung Kwan, Xingshan Zeng, Yuxin Jiang, Yufei Wang, Liangyou Li, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Mt-eval: A multi-turn capabilities evaluation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20153–20177, 2024.

[16] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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*, pages 611–626, 2023.

[17] 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)*, 2024.

[18] Dongfang Li, Zixuan Liu, Gang Lin, Baotian Hu, and Min Zhang. Lycheecluster: Efficient long-context inference with structure-aware chunking and hierarchical kv indexing. *arXiv preprint arXiv:2603.08453*, 2026.

[19] 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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[20] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

[21] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024.

[22] Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*, 2024.

[23] 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. *arXiv preprint arXiv:2505.13109*, 2025.

[24] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2024. URL `https://arxiv.org/abs/2306.03091`.

[25] Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *arXiv preprint arXiv:2502.00299*, 2025.

[26] 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. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[27] OpenAI. Introducing gpt-4.1 in the api. `https://openai.com/index/gpt-4-1/`, 2025. Accessed: 2026-05-05.

[28] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

[29] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[30] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

[31] 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 *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[32] 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. *arXiv preprint arXiv:2410.21465*, 2024.

[33] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[34] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[35] Xindi Wang, Mahsa Salmani, Parsa Omidi, Xiangyu Ren, Mehdi Rezagholizadeh, and Armaghan Eshaghi. Beyond the limits: A survey of techniques to extend the context length in large language models. *arXiv preprint arXiv:2402.02244*, 2024.

[36] Wenbo Wu, Qingyi Si, Xiurui Pan, Ye Wang, and Jie Zhang. Louiskv: Efficient kv cache retrieval for long input-output sequences. *arXiv preprint arXiv:2510.11292*, 2025.

[37] Yuhao Wu, Ming Shan Hee, Zhiqing Hu, and Roy Ka-Wei Lee. Longgenbench: Benchmarking long-form generation in long context llms. *arXiv preprint arXiv:2409.02076*, 2024.

[38] 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. *Advances in neural information processing systems*, 37:119638–119661, 2024.

[39] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[40] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[41] Yuxuan Zhu, Ali Falahati, David H Yang, and Mohammad Mohammadi Amiri. Sentencekv: Efficient llm inference via sentence-level semantic kv caching. *arXiv preprint arXiv:2504.00970*, 2025.

A Proofs for the Theoretical Analysis

This appendix proves the two lemmas used in the theoretical part of Section 3.1. Throughout the proof, we focus on one layer, one attention head, and one decoding step t. We write $r = r(t)$ and $T_r = T_{r(t)}$ for the latest refresh step and its top-k support. We assume $\|v_j\|_2 \le V_{\max}$, $|T_t| = |T_r| = k$, and $1 \le k \le B$ for the historical active-cache budget considered in the main text.

A.1 Auxiliary Bound: Missed Attention Mass

We first relate output error to the attention mass absent from the active cache.

Lemma A.1 (Attention-output error under missed attention mass)**.** *For any nonempty retrieved set S_t, define the missed attention mass as*

$$\epsilon_t(S_t) = \sum_{j \notin S_t} a_t(j).$$

If $\|v_j\|_2 \le V_{\max}$ for all j, then

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\epsilon_t(S_t).$$

Proof. Omit the subscript t for readability. Let $Z_S = \sum_{j \in S_t} a(j) = 1 - \epsilon$ and write

$$u_S = \sum_{j \in S_t} a(j)v_j, \qquad u_{\bar{S}} = \sum_{j \notin S_t} a(j)v_j.$$

The full output is $o = u_S + u_{\bar{S}}$, while the retrieved output renormalizes attention over S_t: $\widetilde{o} = u_S/(1-\epsilon)$. Thus

$$o - \widetilde{o} = u_{\bar{S}} - \frac{\epsilon}{1-\epsilon}u_S.$$

Using the triangle inequality and $\|v_j\|_2 \le V_{\max}$,

$$\|u_{\bar{S}}\|_2 \le \epsilon V_{\max}, \qquad \frac{\epsilon}{1-\epsilon}\|u_S\|_2 \le \epsilon V_{\max}.$$

Therefore, $\|o - \widetilde{o}\|_2 \le 2V_{\max}\epsilon$. □

A.2 Proof of Lemma 3.1

The first main-text lemma states that, once the current active cache fits within the historical budget and contains the previous top-k support T_r, the remaining error is controlled by how many current top-k positions are newly required.

Proof. The missed mass can be decomposed into two parts: mass outside the current top-k support, and current top-k mass absent from the active cache:

$$\epsilon_t(S_t) = \sum_{j \notin S_t} a_t(j) \le \sum_{j \notin T_t} a_t(j) + \sum_{j \in T_t \setminus S_t} a_t(j).$$

The first term is exactly ρ_t^k. Because the active cache contains the previous refreshed support T_r, we have $T_t \setminus S_t \subseteq T_t \setminus T_r$. Also, every index in T_t has attention weight at most $\alpha_t^{\max}$. Hence

$$\sum_{j \in T_t \setminus S_t} a_t(j) \le \alpha_t^{\max}|T_t \setminus T_r|.$$

Plugging this upper bound into Lemma A.1 gives

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}|T_t \setminus T_r|\right),$$

which is Eq. (7). □

A.3 Attention-Overlap Support-Change Identity

This part proves why attention overlap directly measures the realized support-change term.

Proof. Since T_t and T_r are both top-k sets under the same budget, $|T_t| = |T_r| = k$. Let $I = |T_t \cap T_r|$. From the definition of attention overlap,

$$\Omega_t^{\mathrm{attn}} = \frac{|T_t \cap T_r|}{|T_t \cup T_r|} = \frac{I}{2k - I}.$$

Solving for I gives

$$I = \frac{2k\Omega_t^{\mathrm{attn}}}{1 + \Omega_t^{\mathrm{attn}}}.$$

Therefore,

$$|T_t \setminus T_r| = k - I = \frac{k(1 - \Omega_t^{\mathrm{attn}})}{1 + \Omega_t^{\mathrm{attn}}},$$

which is Eq. (9). □

A.4 Proxy Perturbation Bounds by Pairing

This subsection bounds support changes from a logit perturbation radius on a fixed historical candidate pool. Both T_r and T_t rank the same positions, with historical keys held fixed and a fixed tie-breaking rule. This is the setting needed to relate old logit margins to rank changes. The support-change identity requires only equal support sizes; the certificate additionally retains the cache-containment and value-norm assumptions of Lemma 3.1.

Query similarity. Let $s_t(j) = q_t^\top k_j/\sqrt{d}$ be the attention logit, and let K stack historical key vectors as rows, so $\|K\|_{2,\infty} = \max_j \|k_j\|_2$. Define the measured query displacement $\eta_t^q = \|q_t - q_r\|_2$. Cauchy–Schwarz gives

$$\max_j |s_t(j) - s_r(j)| \le \frac{\|K\|_{2,\infty}}{\sqrt{d}} \eta_t^q \triangleq \Gamma_t^q.$$

Let $s_r^{(k)}$ be the k-th largest old logit and define the boundary band

$$G_q(t) = \left\{ j : \ |s_r(j) - s_r^{(k)}| \le \Gamma_t^q \right\}.$$

Write $A = T_r \setminus T_t$ and $D = T_t \setminus T_r$. Since both supports have size k, $|A| = |D|$. Choose any bijection between A and D. For a paired leaving position $j \in A$ and entering position $i \in D$, top-k membership at t gives $s_t(i) \ge s_t(j)$, so

$$0 \le s_r(j) - s_r(i) = [s_r(j) - s_t(j)] + [s_t(j) - s_t(i)] + [s_t(i) - s_r(i)] \le 2\Gamma_t^q.$$

At the old step, $s_r(j) \ge s_r^{(k)} \ge s_r(i)$. Thus the two nonnegative distances $s_r(j) - s_r^{(k)}$ and $s_r^{(k)} - s_r(i)$ sum to at most $2\Gamma_t^q$, and at least one is at most Γ_t^q. Each pair therefore contains at least one member of $G_q(t)$. The pairs are disjoint, so these members are distinct and

$$|T_t \setminus T_r| \le |G_q(t)|.$$

The same argument places each changed position within $2\Gamma_t^q$ of the old cutoff; the narrower Γ_t^q band suffices for the counting bound because each pair contributes at least one position to it.

For nonzero queries, the displacement obeys the exact relation

$$(\eta_t^q)^2 = (\|q_t\|_2 - \|q_r\|_2)^2 + 2\|q_t\|_2\|q_r\|_2(1 - \Omega_t^q).$$

Consequently, a cosine threshold needs norm information to bound the logit perturbation, and old logit margins to bound $|G_q(t)|$.

Key similarity. Key similarity uses the generated-token key κ_t as an indirect signal. To obtain a query-side perturbation bound from it, additional quantitative information must relate key drift to query drift. In particular, given $L_{qk} \geq 0$ and a residual bound $\xi_t \geq 0$ established independently of the observed query displacement, suppose

$$\|q_t - q_r\|_2 \leq L_{qk}\|\kappa_t - \kappa_r\|_2 + \xi_t.$$

Here, L_{qk} bounds sensitivity to generated-key drift and ξ_t bounds query drift not explained by that key difference. These quantities give

$$\Gamma_t^k = \frac{\|K\|_{2,\infty}}{\sqrt{d}} \left(L_{qk}\|\kappa_t - \kappa_r\|_2 + \xi_t\right),$$

and the corresponding ambiguity set

$$G_k(t) = \left\{j : \ |s_r(j) - s_r^{(k)}| \leq \Gamma_t^k\right\}.$$

For each disjoint pair $(j, i) \in A \times D$ in a bijection, the same reasoning gives $s_r(j) - s_r(i) \leq 2\Gamma_t^k$. Since $s_r(j) \geq s_r^{(k)} \geq s_r(i)$, at least one endpoint lies in $G_k(t)$. Counting distinct endpoints over the pairs yields

$$|T_t \setminus T_r| \leq |G_k(t)|.$$

A generated-key cosine threshold supplies neither the coupling parameters L_{qk}, ξ_t nor the norms and old logit margins needed to control this band. The proxy error bounds in Eqs. (10) and (11) follow by substitution into Lemma 3.1, with its active-cache containment assumption retained.

A.5 Proof and Interpretation of the Threshold Certificate

Proof of Lemma 3.2. For $x \in [0, 1]$, let $f(x) = k(1 - x)/(1 + x)$. Its derivative is $f'(x) = -2k/(1 + x)^2 < 0$. If $\Omega_t^{\mathrm{attn}} \geq \tau$, the identity in Appendix A.3 gives

$$|T_t \setminus T_r| = f(\Omega_t^{\mathrm{attn}}) \leq f(\tau) = \frac{k(1-\tau)}{1+\tau}.$$

Substituting into Lemma 3.1 proves

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \leq 2V_{\max}\left(\rho_t^k + \alpha_t^{\max}\frac{k(1-\tau)}{1+\tau}\right),$$

which is Eq. (8). □

Refresh decisions and the certificate. The token-level policy checks overlap before deciding to reuse the active cache. Every reuse step with $\Omega_t^{\mathrm{attn}} \geq \tau$ has the above error certificate; a smaller overlap requests a refresh of the support reference. The guarantee concerns the error on those reuse steps and includes the tail mass ρ_t^k and maximum attention weight $\alpha_t^{\max}$. Appendix A.6 quantifies the additional aggregation error when the overlap is estimated through window anchors.

Why a cosine threshold alone does not control support retention. Consider $d = 2$, queries $q_r = (1, \varepsilon)$ and $q_t = (1, -\varepsilon)$ with $\varepsilon > 0$, and a fixed pool of $2k$ historical keys: k copies of $(0, 1)$ and k copies of $(0, -1)$. The first group is exactly T_r and the second is exactly T_t, so $|T_t \setminus T_r| = k$ and $\Omega_t^{\mathrm{attn}} = 0$. Nevertheless,

$$\Omega_t^q = \frac{1 - \varepsilon^2}{1 + \varepsilon^2} \longrightarrow 1 \qquad \text{as } \varepsilon \longrightarrow 0.$$

Thus, every query-cosine threshold strictly below one can permit complete support replacement. Generated-token keys can also satisfy $\kappa_t = \kappa_r \neq 0$, giving $\Omega_t^k = 1$, while these historical logits and support sets remain as above. Bounding support change from such proxy thresholds therefore requires the additional norm, margin, and coupling information described in Appendix A.4.

A.6 From Token-Level Theory to Window-Level Detection

The token-level analysis uses the true top-k attention support, whereas the implementation selects window IDs through mean-key anchors. We analyze this aggregation for one layer and one head. Let $C_1, \ldots, C_m$ partition the historical token positions into disjoint semantic windows. For this comparison, window IDs and memberships are fixed across the two steps being compared, and each window consists of complete, equal-size pages. Under this condition, the page-mean-then-window-mean anchor in Eq. (12) is also the token-mean key. Let $L_{\min} \leq |C_i| \leq L_{\max}$ and $\kappa = L_{\max}/L_{\min}$.

Anchor ranking and true attention mass. Define the token logit $z_t(j) = q_t^\top k_j/\sqrt{d}$ and the scaled anchor score $s_t(i)$. Scaling all anchor scores by $1/\sqrt{d}$ does not change their ranking. Linearity gives

$$\bar{k}_i = \frac{1}{|C_i|}\sum_{j\in C_i} k_j, \qquad s_t(i) = \frac{q_t^\top \bar{k}_i}{\sqrt{d}} = \frac{1}{|C_i|}\sum_{j\in C_i} z_t(j). \tag{20}$$

Let $P_t(i) = \sum_{j\in C_i}\exp(z_t(j))$ be unnormalized window attention mass and $g_t(i) = \log P_t(i)$. Ranking by g_t is equivalent to ranking by true window attention mass because the softmax denominator is shared by all windows. Assume $|z_t(j) - s_t(i)| \le \nu_t$ within each window, where ν_t bounds within-window logit variation. Jensen's inequality and Hoeffding's lemma imply

$$g_t(i) = s_t(i) + \log|C_i| + \xi_t(i), \qquad 0 \le \xi_t(i) \le \frac{\nu_t^2}{2}. \tag{21}$$

Indeed, $\xi_t(i)$ is the log moment-generating function of the zero-mean variable $z_t(j) - s_t(i)$ under a uniform token draw from C_i, whose range has width at most $2\nu_t$. Thus, for any two windows,

$$|(g_t(i) - g_t(j)) - (s_t(i) - s_t(j))| \le \beta_t, \qquad \beta_t = \log\kappa + \frac{\nu_t^2}{2}. \tag{22}$$

If the anchor-score gap between the K_w-th and (K_w+1)-th windows exceeds β_t, anchor top-K_w and true-mass top-K_w select the same windows. Otherwise, ranking reversals can occur only between windows whose anchor scores differ by at most β_t. Window-length constraints control κ, while attention-continuity segmentation is intended to reduce within-window logit variation.

Token support and selected windows. Let $\mathcal{W}_t^\star$ be the true-mass top-K_w window-ID set and $\widehat{\mathcal{W}}_t$ the anchor-selected set used by the implementation. Their expanded token sets are $U_t^\star = \bigcup_{i\in\mathcal{W}_t^\star} C_i$ and $\widehat{U}_t = \bigcup_{i\in\widehat{\mathcal{W}}_t} C_i$. For nonempty sets, define Jaccard distance $d_J(A,B) = 1 - |A\cap B|/|A\cup B|$. Write $\eta_t = d_J(T_t, U_t^\star)$ for token-to-window aggregation error and $h_t = |\mathcal{W}_t^\star \setminus \widehat{\mathcal{W}}_t|$ for the number of misselected windows. Both window sets contain K_w IDs. Their expanded symmetric difference contains at most $2h_tL_{\max}$ tokens, while their union contains at least $K_wL_{\min}$ tokens. The triangle inequality therefore gives

$$d_J(T_t, \widehat{U}_t) \le \epsilon_t, \qquad \epsilon_t = \eta_t + \min\left\{1, \frac{2\kappa h_t}{K_w}\right\}. \tag{23}$$

This separates the aggregation error η_t from anchor-ranking mistakes h_t and window-length variation κ.

Overlap and drift decisions. For the same pair of steps t, u, let $\Omega_{t,u}^{\mathrm{tok}}$, $\Omega_{t,u}^{\mathrm{win}}$, and $\Omega_{t,u}^{\mathrm{exp}}$ denote the Jaccard overlaps of (T_t, T_u), $(\widehat{\mathcal{W}}_t, \widehat{\mathcal{W}}_u)$, and $(\widehat{U}_t, \widehat{U}_u)$, respectively. If $x = \Omega_{t,u}^{\mathrm{win}}$, bounding the token counts in the shared and unshared windows by $L_{\min}$ and $L_{\max}$ yields

$$\frac{x}{\kappa - (\kappa-1)x} \le \Omega_{t,u}^{\mathrm{exp}} \le \frac{\kappa x}{1 + (\kappa-1)x}. \tag{24}$$

Maximizing the deviation from x over $x \in [0,1]$ gives

$$\left|\Omega_{t,u}^{\mathrm{exp}} - \Omega_{t,u}^{\mathrm{win}}\right| \le \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \tag{25}$$

Combining this with the triangle inequality for Jaccard distance and Eq. (23) gives

$$\left|\Omega_{t,u}^{\mathrm{tok}} - \Omega_{t,u}^{\mathrm{win}}\right| \le E_{t,u}, \qquad E_{t,u} = \epsilon_t + \epsilon_u + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \tag{26}$$

For a common threshold τ, the two drift decisions agree whenever $\Omega_{t,u}^{\mathrm{win}} < \tau - E_{t,u}$ or $\Omega_{t,u}^{\mathrm{win}} > \tau + E_{t,u}$. Disagreement is confined to the corresponding threshold band; the bound is informative when the aggregation and ranking errors and the window-length ratio are small. The same-pair requirement matters: $u = r(t)$ gives the refresh-reference comparison in Sec. 3.1, whereas $u = t_{\mathrm{last}}$ gives the previous-detection comparison in Eq. (18).

Under the active-cache containment assumption of Lemma 3.1, set $\underline{\Omega}_{t,r} = \max\{0, \Omega^{\mathrm{win}}_{t,r} - E_{t,r}\}$. Since $k(1-x)/(1+x)$ decreases with x, the support-change identity implies

$$\|o_t - \widetilde{o}_t(S_t)\|_2 \le 2V_{\max}\left(\rho^k_t + \alpha^{\max}_t \frac{k(1-\underline{\Omega}_{t,r})}{1+\underline{\Omega}_{t,r}}\right). \tag{27}$$

Equation (27) expresses the token-overlap term through windows, with an explicit aggregation residual.

Detection cost and empirical comparison. With n historical tokens, exact token detection requires n query-key dot products and $O(nd)$ key reads per event. CPU-resident historical keys would therefore need to be scanned or retained on GPU. With mean window length $\bar{L}$, anchor detection uses about $n/\bar{L}$ dot products and $O(nd/\bar{L})$ GPU anchor storage. An average stride $\bar{\delta}$ reduces the amortized scoring work to $O(nd/(\bar{L}\bar{\delta}))$ per decoding step.

Table 3 compares both detectors on six LongBench tasks with Llama-3.1-8B-Instruct and a 1,024-token KV budget. Window detection matches or improves five of the six scores, with average accuracy 43.89 versus 43.62. Drift-point overlap ranges from 0.58 to 0.78, averaging 0.685. Within-window token-rank swaps can alter token support while leaving the recalled pages unchanged.

Table 3: Token-level and window-anchor detection with Llama-3.1-8B-Instruct under a 1,024-token KV budget. The first two rows report task scores; the last row reports drift-point overlap between the detectors. Avg. is the arithmetic mean across the six tasks.

Detector	**NQA**	**Qasper**	**Musique**	**TriviaQA**	**P.Count**	**LCC**	**Avg.**
Per-token Top-k	29.34	43.88	30.37	88.36	8.75	61.03	43.62
Window anchor	30.05	44.08	31.08	88.65	8.75	60.74	43.89
Drift-point overlap	0.58	0.69	0.66	0.69	0.78	0.71	0.685

B Additional Details for Observations

Figure 7 complements Figure 4b in the main Method section. While Figure 4b visualizes a partial trace from the NarrativeQA setting, this appendix figure provides a broader view across five representative LongBench subsets and three model depths: a shallow layer (L2), a middle layer (L18), and a deep layer (L30). The traces show how the Top-k attention-overlap score evolves during decoding under different task types and layers, making the layer- and task-dependent variation of semantic drift more explicit.

C Expanded Accuracy Results

C.1 Additional Model Backbones

We further evaluate AsdaKV on LongBench with additional model backbones. The category definitions follow Table 1: Single-Doc. QA, Multi-Doc. QA, Summarization, Few-shot Learning, Synthetic Tasks, and Code Completion are averaged over their corresponding LongBench tasks, while Avg. is averaged over all 16 tasks. All compressed methods use the same 1,024-token KV budget.

C.2 Needle-in-a-Haystack Results

We additionally evaluate retrieval methods on the Needle-in-a-Haystack setting with Llama-3.1-8B-Instruct under the same 1,024-token KV budget. This benchmark stresses whether a method can preserve and retrieve sparse evidence as the context length grows. Figure 8 reports accuracy from 4K to 40K tokens. AsdaKV remains close to FullKV across all lengths and is substantially more stable than FreeKV at longer contexts.

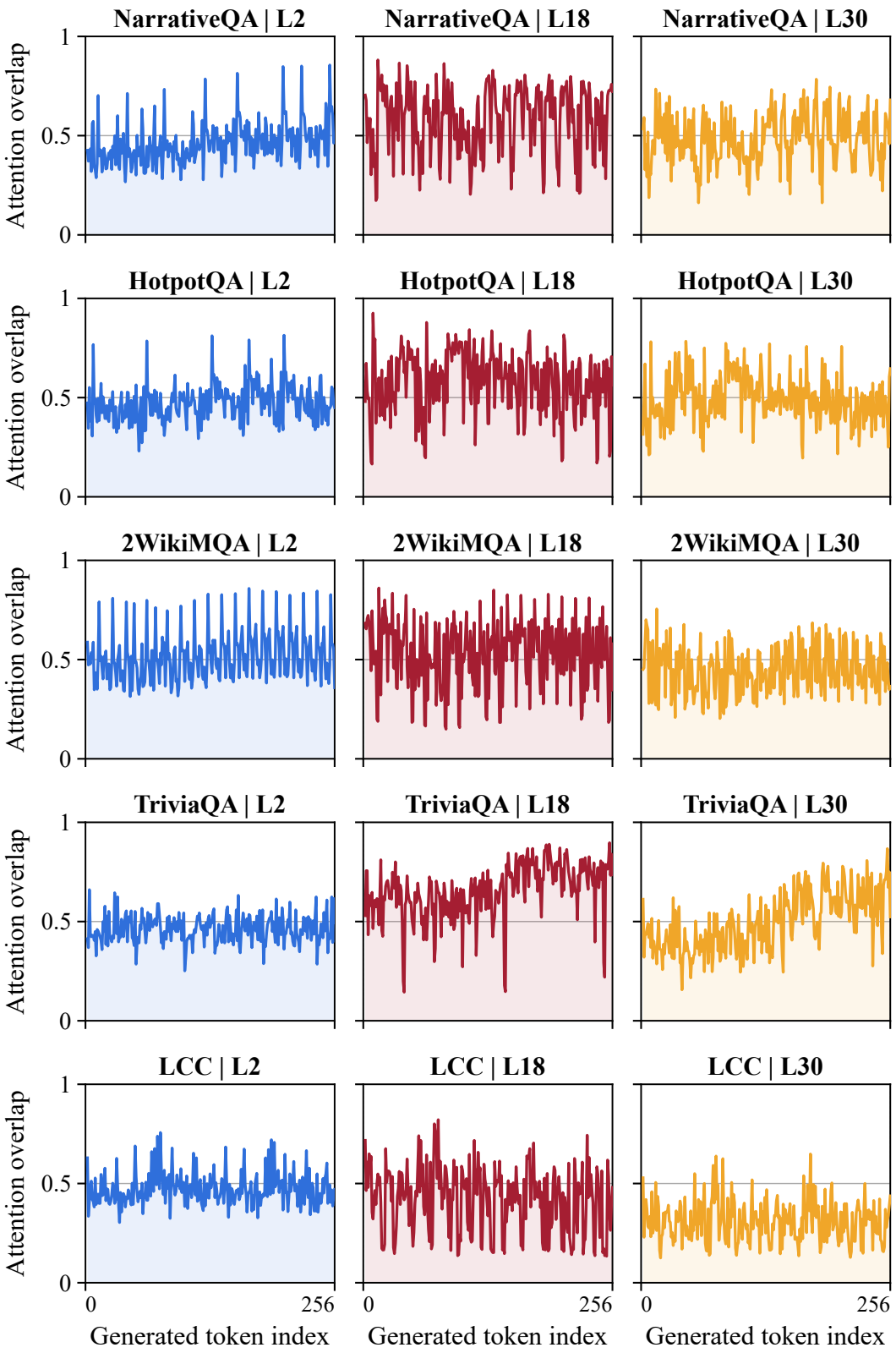


Figure 7: Attention-overlap traces across representative LongBench tasks and model layers. Each row corresponds to a task, and each column corresponds to a layer. The x-axis is the generated-token index, and the y-axis is the Top-k attention-overlap score.

Table 4: LongBench performance with Mistral-7B-Instruct-v0.3 under a 1,024-token budget. Each category reports the mean score over its constituent tasks, and Avg. is averaged over all 16 tasks. The best and second-best compressed methods are marked in **bold** and underlined, respectively.

Method	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synth.	Code	Avg.
FullKV	39.32	37.56	28.88	53.20	51.00	62.24	43.96
SnapKV	38.08	35.74	26.25	54.92	**50.25**	61.45	43.02
ClusterKV	37.28	34.09	26.95	54.63	39.75	61.62	41.35
ShadowKV	37.18	35.71	**28.57**	56.84	48.50	61.13	43.39
FreeKV	37.30	36.28	27.85	56.46	49.00	**61.79**	43.45
AsdaKV	**38.16**	**36.52**	27.78	**56.88**	48.25	61.49	**43.59**

Table 5: LongBench category-level performance with Qwen3-8B under a 1,024-token budget. Each category reports the mean score over its constituent tasks, and Avg. is averaged over all 16 tasks. The best and second-best compressed methods are marked in **bold** and underlined, respectively.

Method	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synth.	Code	Avg.
FullKV	42.79	46.33	27.59	34.34	50.50	66.69	42.97
ClusterKV	40.92	44.86	25.71	31.89	50.00	63.50	41.07
ShadowKV	42.38	46.22	27.00	32.38	**51.00**	63.12	42.01
FreeKV	**42.55**	**46.31**	**27.08**	32.07	50.75	64.17	42.12
AsdaKV	42.31	46.28	26.31	**33.44**	50.50	**66.20**	**42.40**

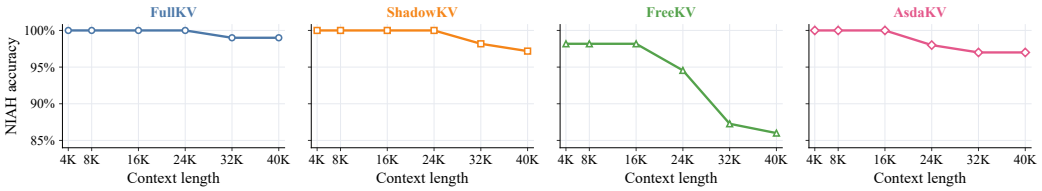


Figure 8: Needle-in-a-Haystack accuracy under a 1,024-token KV budget. Accuracy is reported across context lengths from 4K to 40K tokens.

C.3 RULER Length-Scaling Results

We further evaluate length extrapolation on RULER benchmark with Llama-3.1-8B-Instruct model under the same 1,024-token budget. As shown in Table 6, AsdaKV remains competitive across lengths and gives the highest average score among the compressed methods, with larger gains at longer contexts such as 128K.

Table 6: RULER length-scaling accuracy with Llama-3.1-8B-Instruct under a 1,024-token budget. The best and second-best compressed methods are marked in **bold** and underlined, respectively.

Method	4K	8K	16K	32K	64K	128K	Avg.
FullKV	96.13	95.05	93.21	91.60	84.77	76.96	89.62
SnapKV	91.36	78.94	64.73	60.28	51.87	35.47	63.78
SentenceKV	94.03	**94.47**	88.62	87.06	77.46	68.40	85.01
ShadowKV	94.42	94.11	91.19	84.92	**81.53**	67.83	85.67
FreeKV	**96.08**	90.51	86.16	81.40	77.15	56.26	81.26
AsdaKV	94.97	94.13	**91.88**	**88.67**	81.32	**70.07**	**86.84**

C.4 LongBench Results under Different Token Budgets

We further compare the LongBench average score under different KV token budgets. Table 7 shows that AsdaKV becomes increasingly competitive as the available cache budget grows. FreeKV

performs best under very tight 128- and 256-token budgets, while AsdaKV achieves the highest average from 512 to 2,048 tokens. This trend is consistent with AsdaKV's window-level design: when the budget is too small, the active cache cannot retain enough complete semantic windows after fixed sink and recent-token reservations. As a result, variable-length semantic blocks are more likely to be truncated or fragmented, and the retrieved context may not contain enough key information from each block. FreeKV's finer-grained token/page-level retrieval is less exposed to this failure mode at extremely small budgets. Once the budget is large enough to keep several complete semantic windows, AsdaKV's attention-overlap driven retrieval more effectively selects useful long-context evidence.

Table 7: LongBench average performance under different KV token budgets. The best and second-best compressed methods under each budget are marked in **bold** and underlined, respectively.

Method / budget	128	256	512	1024	2048
ShadowKV	40.66	45.46	46.08	47.34	47.88
SentenceKV	38.21	43.11	45.97	47.09	47.33
LouisKV	37.90	39.64	45.24	47.32	47.65
FreeKV	**43.17**	**46.56**	47.13	47.64	48.03
AsdaKV	41.78	46.35	**47.15**	**48.08**	**48.11**

C.5 Large-Model Evaluation on Llama-3.1-70B-Instruct

We further test whether the advantage of AsdaKV remains on a much larger model. We evaluate six representative LongBench tasks with Llama-3.1-70B-Instruct. As shown in Table 8, AsdaKV gives the best average score among the compressed baselines under the same 1,024-token budget.

Table 8: LongBench task-level performance with Llama-3.1-70B-Instruct under a 1,024-token budget. The best and second-best compressed methods are marked in **bold** and underlined, respectively. Two Lcc samples that exceed GPU memory under FullKV are excluded for all methods.

Method	Qasper	Musique	MultiNews	TriviaQA	PR-en	Lcc	Avg.
FullKV	49.31	47.11	26.55	92.77	100.00	39.64	59.23
SnapKV	48.07	**48.08**	25.97	**92.55**	**99.50**	38.79	58.83
ShadowKV	49.03	47.16	27.42	89.71	**99.50**	39.24	58.68
LouisKV	49.35	43.13	**28.71**	68.53	99.00	38.96	54.61
FreeKV	48.26	45.75	26.50	91.83	**99.50**	38.85	58.45
AsdaKV	**49.56**	47.35	26.56	91.56	**99.50**	**39.83**	**59.06**

D Detailed Efficiency Analysis

We report two experimental platforms separately. The A100 experiments measure absolute latency in a common HuggingFace runtime and provide per-layer profiling results. The H200 experiments evaluate an optimized AsdaKV backend against ShadowKV and FreeKV on the same H200 GPU.

D.1 A100: Absolute End-to-End and Decoding Latency

All retrieval methods in Table 9 use Llama-3.1-8B-Instruct in the same HuggingFace runtime on one NVIDIA A100-80GB GPU with PCIe 4.0, a 1,024-token KV budget, batch size 1, and 128 output tokens. We discard five warm-up runs and report the mean of five measured runs. HF-FullKV is the common-runtime baseline. vLLM-FullKV provides a separate optimized-backend reference.

AsdaKV's TPOT increases from 16.55 to 23.44 ms between 16K and 128K input, compared with 21.49 to 49.30 ms for HF-FullKV and 26.07 to 36.82 ms for FreeKV. This supports a decode-side scaling advantage.

Table 9: Absolute latency on A100 with Llama-3.1-8B-Instruct, batch size 1, 128 output tokens, and a 1,024-token retrieval budget. E2E and TTFT are in seconds; TPOT is in milliseconds per output token. Lower is better. vLLM-FullKV uses a different backend.

	16K input			128K input		
Method	**E2E (s)**	**TTFT (s)**	**TPOT (ms)**	**E2E (s)**	**TTFT (s)**	**TPOT (ms)**
HF-FullKV	3.43	0.70	21.49	24.07	17.81	49.30
ShadowKV	9.59	7.47	16.66	34.06	31.91	16.97
FreeKV	4.43	1.12	26.07	27.46	22.79	36.82
InfiniGen	7.34	2.48	38.30	32.43	27.26	40.67
AsdaKV	5.09	3.00	16.55	33.43	30.99	23.44
vLLM-FullKV	1.26	0.53	5.79	12.64	11.43	9.54

D.2 A100: Per-Layer Runtime Breakdown

Table 10 reports an average single-layer latency breakdown of AsdaKV on the A100 platform across different sequence lengths. The active-cache attention cost stays nearly flat as the input length increases because AsdaKV keeps the active KV budget fixed, while the query-anchor dot product and drift detection grow with the number of historical anchors. The recall cost also increases with sequence length because longer contexts expose more candidate pages for retrieval. The final column reports full attention as a reference baseline.

Table 10: A100 per-layer latency breakdown of AsdaKV under different sequence lengths. All values are measured in milliseconds. QKV includes the projection and RoPE cost. Full attention reports the raw attention-computation baseline used as a reference point.

Seq. Len.	**QKV**	**Active Attn.**	**Anchor Dot**	**Detect (overlap)**	**Offload (overlap)**	**Recall (overlap)**	**Full Attn. (baseline)**
8k	0.46	0.11	0.09	0.45	0.08	1.23	0.10
16k	0.49	0.10	0.11	0.52	0.07	1.24	0.12
32k	0.53	0.11	0.12	0.62	0.11	1.25	0.16
64k	0.60	0.10	0.15	1.98	0.08	1.28	0.26
128k	0.85	0.10	0.21	1.76	0.12	1.29	0.45

D.3 H200: Long-Input Decoding Throughput

The H200 experiments use Llama-3.1-8B-Instruct on a single NVIDIA H200 NVL. AsdaKV, ShadowKV, and FreeKV are evaluated on this shared platform, with an optimized AsdaKV backend. All H200 throughput entries report aggregate output tokens per second across the batch, measured over the final 100 decoding steps.

Table 11: Long-input decode throughput on H200 NVL with Llama-3.1-8B-Instruct. Values are aggregate tokens/s over the final 100 decoding steps. Higher is better; the fastest method in each row is bold.

Input	**Batch**	**AsdaKV**	**ShadowKV**	**FreeKV**
32K	1	**62.886**	59.345	49.320
64K	1	**60.720**	58.932	40.734
128K	1	**61.493**	58.964	31.328
60K	48	350.600	**941.410**	83.420
122K	24	224.950	**463.780**	112.200

At batch size 1, AsdaKV maintains approximately 61–63 tokens/s from 32K to 128K input and exceeds both baselines. ShadowKV is substantially faster in the large-batch settings: 941.41 versus

350.60 tokens/s at 60K input and batch 48, and 463.78 versus 224.95 tokens/s at 122K input and batch 24.

D.4 H200: Long-Output Throughput and GPU Memory

We fix the input length at 4K, generate the output prefix listed in each row, and measure throughput over the final 100 decoding steps. Peak GPU memory is reported for the same runs. Tables 12 and 13 use the H200 platform and model specified above.

Table 12: Long-output decode throughput on H200 NVL with Llama-3.1-8B-Instruct and a 4K input. Output denotes the generated prefix length. Values are aggregate tokens/s over the final 100 decoding steps; higher is better. The fastest method in each row is bold.

Output	Batch	AsdaKV	ShadowKV	FreeKV
16K	1	**64.823**	58.450	48.945
16K	4	**249.604**	224.155	166.774
16K	8	**468.066**	448.201	255.613
8K	16	**865.156**	853.551	438.047
16K	16	**820.817**	730.038	519.819
32K	16	**822.545**	527.063	156.820

Table 13: Peak GPU memory (GiB) for the same H200 long-output runs as Table 12, with Llama-3.1-8B-Instruct and a 4K input. Lower is better; the smallest footprint in each row is bold.

Output	Batch	AsdaKV	ShadowKV	FreeKV
16K	1	16.72	18.26	**16.67**
16K	4	17.48	25.05	**17.41**
16K	8	18.57	34.09	**18.32**
8K	16	20.33	36.18	**19.67**
16K	16	20.58	52.17	**20.20**
32K	16	21.99	84.18	**21.57**

AsdaKV has the highest throughput in all six long-output settings. At 32K output and batch 16, it reaches 822.545 tokens/s versus 527.063 for ShadowKV, a 56.1% increase, while peak GPU memory is 21.99 versus 84.18 GiB, a 73.9% reduction. From 8K to 32K output at batch 16, AsdaKV memory grows from 20.33 to 21.99 GiB, whereas ShadowKV grows from 36.18 to 84.18 GiB. FreeKV uses slightly less memory than AsdaKV in each run, but has lower throughput.

E Ablation Studies

This appendix provides detailed ablations for the algorithmic and system components of AsdaKV. We group the studies by the role each component plays in the pipeline: refresh triggering signal, semantic window anchor construction, drift-detection scheduling, task-profile calibration, recall scheduling, head-wise retrieval and physical KV organization. The measurements in this section use the A100 platform described in the main experimental settings. H200 throughput and memory measurements are reported separately in Appendix D.3.

E.1 Semantic Boundary and Refresh Signal Ablation

We first ablate the drift-detection signal inside the AsdaKV framework. All variants keep the same retrieval pipeline, active-cache budget, page layout, and recall schedule, and only replace how semantic drift is detected. We compare AsdaKV's attention-overlap detector with fixed-page variants using page sizes 32, 64, and 128, as well as proxy detectors based on query similarity and key similarity. The goal is to test whether attention overlap identifies changes in the model's actual retrieval support more reliably than static page boundaries or indirect representation-similarity signals. Table 14 reports the category-level LongBench results on Llama-3.1-8B-Instruct under the 1,024-

token budget. Attention-overlap detection achieves the highest average, with the largest gain over query/key similarity on summarization.

Table 14: Semantic boundary and refresh signal ablation on LongBench with Llama-3.1-8B-Instruct under a 1,024-token budget. Each category reports the mean score over its constituent tasks, and Avg. is averaged over all 16 tasks. The best and second-best ablated variants are marked in **bold** and underlined, respectively.

Method	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synth.	Code	Avg.
FullKV	43.18	43.62	29.38	68.24	53.50	57.89	48.50
Fixed-size 32	42.60	43.33	27.05	66.88	52.74	57.03	47.45
Fixed-size 64	41.80	42.20	26.69	65.44	52.67	56.24	46.64
Fixed-size 128	39.25	39.68	24.37	63.00	50.28	55.10	44.35
Query Similarity	43.12	43.88	26.55	**68.10**	53.03	**57.22**	47.84
Key Similarity	**43.30**	43.58	27.02	67.94	**53.11**	56.36	47.78
Attention Overlap	43.26	**43.92**	**27.83**	68.07	52.88	57.16	**48.08**

E.2 Window Anchor Design Ablation

We next study how the design of window anchors affects retrieval quality. The comparison isolates the contribution of query-aware prefill construction and anchor aggregation choices, measuring whether the anchors remain representative of the full semantic window during later decoding. This ablation tests the central assumption that lightweight anchors can guide recall without keeping all historical KV states on GPU. Table 15 reports the results. The max-attention page mean variant uses the subsequent snap tokens to attend over the pages inside each preceding window, selects the page with the largest accumulated attention mass, and uses the mean key of that page as the window anchor. This choice gives slightly higher accuracy on several categories, but it requires extra attention computation during inference and therefore introduces latency. To balance accuracy and efficiency, AsdaKV uses the page-mean-then-window-mean design.

Table 15: Window anchor ablation on LongBench with Llama-3.1-8B-Instruct under a 1,024-token budget. Each category reports the mean score over its constituent tasks, and Avg. is averaged over all 16 tasks. The best and second-best ablated variants are marked in **bold** and underlined, respectively.

Method	Single-Doc. QA	Multi-Doc. QA	Summ.	Few-shot	Synth.	Code	Avg.
FullKV	43.18	43.62	29.38	68.24	53.50	57.89	48.50
First-page Mean	40.68	39.97	27.05	66.07	51.74	55.27	45.96
Last-page Mean	42.11	41.20	26.89	65.44	52.27	56.05	46.47
Random-page Mean	43.02	42.18	27.07	66.00	52.08	56.10	46.95
Max-attn Page Mean	**43.30**	**44.02**	**27.95**	**68.10**	52.83	**57.22**	**48.14**
Page-to-Window Mean	43.26	43.92	27.83	68.07	**52.88**	57.16	48.08

E.3 Adaptive Drift-Detection Stride Ablation

We ablate the adaptive drift-detection stride, which avoids launching expensive drift checks at every decoding step when the retrieval state changes slowly. The experiment is conducted on Qwen3-8B under the same decoding and evaluation protocol as the other ablations, and uses the same offline profile-level stride calibration as the main AsdaKV configuration. As shown in Table 16, enabling stride reduces both the average per-layer decoding latency and the extra computation latency. This is because many adjacent decoding steps reuse a similar retrieval state, so the drift detector does not need to be relaunched every step. The small accuracy changes on LongBench and MATH500 indicate that the skipped checks have limited impact on accuracy.

Table 16: Qwen3-8B ablations on A100. Decode latency is the average per-layer decoding latency in milliseconds. Each pair isolates one component; parentheses give changes relative to the variant without it. Red and blue mark positive and negative effects.

Component	Method	Decode	Extra Comp.	Comm.	LongBench	MATH500
Adaptive Stride	No Stride	1.34	0.54	0	42.46	93.4
	Stride	**1.21 (-0.13)**	**0.31 (-0.23)**	0 (+0)	**42.40 (-0.06)**	**93.2 (-0.2)**
Task Calibration	C4-only mismatched profile	1.11	0.31	0	40.37	91.4
	Task-specific	**1.21 (+0.10)**	0.31 (+0)	0 (+0)	**42.40 (+2.03)**	**93.2 (+1.8)**
Deferred Recall	Sync Recall	2.26	0.52	0.94	42.37	93.2
	Deferred Recall	**1.21 (-1.05)**	**0.31 (-0.21)**	**0 (-0.94)**	**42.40 (+0.03)**	**93.2 (+0.0)**
Per-Head Retrieval	Head-mean	1.07	0.27	0	42.09	92.2
	Head-wise	**1.21 (+0.14)**	**0.31 (+0.04)**	0 (+0)	**42.40 (+0.31)**	**93.2 (+1.0)**
Physical Layout	Window-level	1.96	0.48	0.68	42.3	93.2
	Page-level	**1.21 (-0.75)**	**0.31 (-0.17)**	**0 (-0.68)**	**42.4 (+0.1)**	**93.2 (+0.0)**

E.4 Task-Specific Calibration Ablation

We ablate the calibration strategy used to set the drift threshold and adaptive stride parameters. The comparison keeps semantic-window construction, anchor design, page layout, and deferred recall fixed, and changes only how the calibrated drift and stride values are assigned to requests. The C4-only mismatched profile calibrates one set of parameters using only C4 and applies the resulting drift threshold and stride boundaries to all tasks. It measures calibration-distribution mismatch. The task-specific variant follows the workload taxonomy described in the method section and separates requests into four types: evidence-seeking, aggregation, generation-dominant reasoning, and code generation. Each task type has its own calibrated parameter set.

This ablation tests whether the retrieval refresh policy should adapt to task specificity. Different workloads create different retrieval-pressure regimes: long-input evidence-seeking and aggregation tasks exceed the active-cache budget from the first decoding step, while generation-dominant reasoning and code generation often start from short inputs and only require historical retrieval after enough output tokens have been generated. As shown in Table 16, task-specific calibration improves LongBench and MATH500 accuracy, with average per-layer decode latency increasing from 1.11 to 1.21 ms. The parameters are calibrated offline, so profile selection introduces no request-local calibration on the TTFT path.

Calibration sources and workload assignment. Each of the four profiles uses 64 unlabeled calibration samples disjoint from the evaluation examples. Table 17 lists the sources. Calibration uses forward-pass attention-overlap traces. As described in Sec. 3.2, samples are collected after the historical cache exceeds the active budget and the runtime state has warmed up, with at most 512 valid overlap samples per prompt.

Table 17: Offline calibration data for the four workload profiles. Each source provides 64 unlabeled samples disjoint from the evaluation examples.

Profile	Calibration source	Samples
Evidence seeking	PG-19 [29]	64
Aggregation	Concatenated Wikipedia documents	64
Generation-dominant reasoning	MATH training split [8]	64
Code generation	The Stack [14]	64

The workload-specific assignment maps NarrativeQA, Qasper, MultiFieldQA-en, and PassageRetrieval-en to evidence seeking. HotpotQA, 2WikiMQA, MuSiQue, GovReport, QMSum, MultiNews, TREC, TriviaQA, SAMSum, and PassageCount use the aggregation profile.

LCC and RepoBench-P use the code profile; MATH500 and GSM8K use the generation-dominant reasoning profile.

Shared-threshold calibration. We evaluate a shared drift threshold τ across workloads, with stride boundaries $b^{\text{low}} = \tau + 0.15$ and $b^{\text{high}} = \min\{b^{\text{low}} + 0.15, 1\}$. Increasing τ causes more frequent refreshes, improving accuracy at greater decoding cost. A shared threshold achieves comparable accuracy at higher latency (Table 18): $\tau = 0.60$ yields 42.43 on LongBench versus 42.40 for workload-specific calibration, with latency 1.38 versus 1.21 ms. Workload-specific profiles mainly improve the accuracy–latency trade-off.

Table 18: Shared-threshold calibration with Qwen3-8B on A100. The table reports LongBench scores and average per-layer decode latency for the threshold sweep and the same workload-specific configuration as in Table 16.

Calibration	LongBench	Decode latency (ms)
Shared $\tau = 0.40$	40.03	1.17
Shared $\tau = 0.45$	40.27	1.22
Shared $\tau = 0.50$	40.37	1.26
Shared $\tau = 0.55$	41.34	1.32
Shared $\tau = 0.60$	42.43	1.38
Workload-specific	42.40	1.21

Automatic classification and online calibration. For requests without workload labels, we evaluate two alternatives. The automatic binary classifier uses a 4K-token input-length threshold to distinguish context-dominant from generation-dominant requests, without task names. Online request-local calibration uses no workload classification: after prefill, it uses the latest $L_{\text{cal}} = \min\{128, L_{\text{prefill}}\}$ queries, where L_{prefill} is the prompt length. For these queries it computes overlap between adjacent top-12 window sets and sets

$$\tau_{\text{init}} = Q_{0.05}\big(\{\Omega_2, \ldots, \Omega_{L_{\text{cal}}}\}\big). \tag{28}$$

Here Ω_ℓ is the window-set Jaccard overlap for consecutive calibration queries, and $Q_{0.05}$ is the empirical fifth percentile.

Table 19 reports the results. Automatic classification achieves 41.26 on LongBench and 93.2 on MATH500 at 55.38 tokens/s. Online calibration reaches 42.35 and 92.8, respectively, at 51.76 tokens/s, compared with 42.40, 93.2, and 55.58 tokens/s for workload-specific calibration. Its throughput is 6.9% lower. These alternatives allow deployment without detailed workload labels, while the offline profiles provide the best combined accuracy and throughput among the three options.

Table 19: Calibration and deployment alternatives with Qwen3-8B on A100. MATH500 entries are accuracy percentages, and throughput is in tokens/s.

Calibration	LongBench	MATH500	Throughput
Automatic binary classifier	41.26	93.2	55.38
Online request-local	42.35	92.8	51.76
Workload-specific	42.40	93.2	55.58

E.5 Deferred KV Recall Ablation

We next evaluate deferred KV recall. The synchronous baseline waits for KV transfer and recall to finish on the critical path, whereas the deferred design overlaps recall with other decoding work and commits pending pages later. In this ablation, both variants use the same retrieval decisions and differ only in whether KV recall is executed synchronously or deferred. As summarized in Table 16, deferred recall reduces average per-layer decoding latency from 2.26 to 1.21 ms and explicit communication latency from 0.94 to 0 ms, because CPU–GPU transfers are removed from the immediate decode path. The nearly unchanged LongBench and MATH500 scores are consistent with both variants eventually recalling the same selected pages.

E.6 Per-Head Retrieval Ablation

We compare two ways of using head-level retrieval signals. In the head-wise variant, each KV head independently detects drift from its own anchor scores and recalls the pages selected for that head. In the head-mean variant, retrieval scores are first averaged across heads, and the resulting shared score is used to make one recall decision for all heads. The experiment keeps the same page organization and recall schedule, and changes only whether drift detection and recall are performed independently per head or after cross-head averaging. Head-wise retrieval introduces a small latency increase because it maintains separate retrieval decisions, but Table 16 shows that it improves LongBench and MATH500 accuracy by preserving head-specific retrieval patterns that are smoothed out by the head-mean baseline.

E.7 Physical KV Layout and Page Organization Ablation

Finally, we ablate the physical organization of KV states. Page-level layout separates semantic retrieval units from physical transfer units: the model can make the same high-level retrieval decision while moving only the pages that are actually needed. Table 16 shows that, compared with window-level transfer, page-level organization reduces decode latency, extra computation latency, and data communication latency without hurting accuracy. Here, window-level means that KV states are not organized into fixed-size pages; instead, each recall directly transfers the selected variable-length semantic windows, whose sizes are not constrained to be multiples of the page size. Page-level layout is faster because it turns these irregular transfers into reusable fixed-size physical pages, reducing redundant movement while preserving the same semantic window selection.

F Hyperparameter Sensitivity Analysis

F.1 Sensitivity to Retrieved Window Number top-k

Figure 9(a) varies the number of retrieved semantic windows. Increasing top-k from 8 to 12 substantially improves accuracy, indicating that too small a retrieval set misses useful historical evidence. Beyond the default choice top-$k = 12$, the accuracy curve is nearly saturated: top-$k = 16$ gives only a marginal gain, while detection latency keeps increasing as more candidate windows are scored and maintained. This suggests that a moderate top-k captures most useful retrieval context without paying unnecessary detection overhead.

F.2 Sensitivity to Physical Page Size

Figure 9(b) studies the physical page size used by the KV transfer path. The default page size 8 preserves the highest accuracy, but incurs the largest transfer latency due to more fragmented page movement. Larger pages reduce transfer cost, but they degrade accuracy because each recalled unit brings in more irrelevant KV states under the same active-cache budget. The curve therefore shows a clear accuracy–latency trade-off.

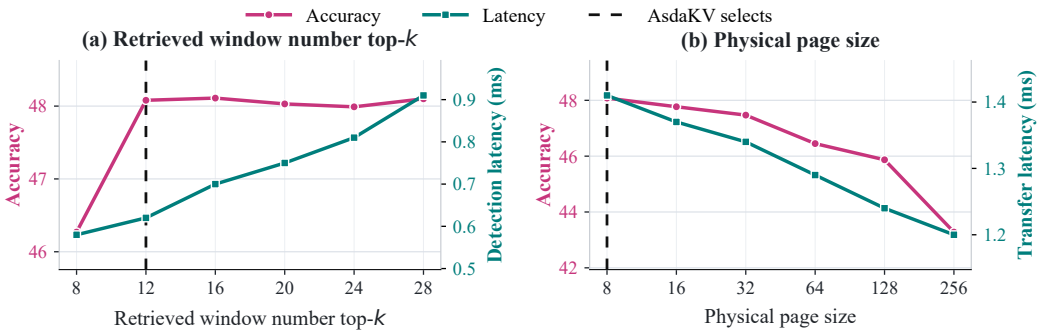


Figure 9: Hyperparameter sensitivity of AsdaKV to (a) retrieved window number top-k and (b) physical page size. Pink curves report accuracy and cyan curves report the corresponding latency metric, showing the trade-off between retrieval quality and system overhead. Black dashed vertical lines mark the configuration selected by AsdaKV.

F.3 Sensitivity to Observation Length L_{obs}

AsdaKV builds semantic windows from the attention pattern of the last L_{obs} prefill queries. A shorter observation window reduces prefill overhead but may miss useful context-demand variation near the prompt boundary, whereas a longer observation window gives a more stable boundary estimate at the cost of higher TTFT. Table 20 varies L_{obs} on LongBench with Llama-3.1-8B-Instruct. Accuracy is already stable at $L_{\text{obs}} = 32$, while larger values mainly increase TTFT. We therefore use $L_{\text{obs}} = 32$ as the default setting.

Table 20: Sensitivity to the prefill observation length L_{obs} on LongBench with Llama-3.1-8B-Instruct. TTFT is measured for a 32K-token input and 32-token output setting. The pink-bold entries mark the default setting.

L_{obs}	**16**	**32**	**64**	**128**	**256**
TTFT (s)	4.69	**4.98**	5.45	6.19	6.80
LongBench accuracy	48.05	**48.08**	48.11	48.12	48.12

F.4 Sensitivity to Offline Calibration Quantile α_c

Table 21 varies the offline calibration percentile $100\alpha_c$ from 2 to 10, corresponding to $\alpha_c = 0.02$–0.10 in the drift threshold $\tau_c = Q_{\alpha_c}(\mathcal{O}_c)$. Accuracy increases from 44.37 at the second percentile to 48.08 at the default fifth percentile ($\alpha_c = 0.05$), while average per-layer decoding latency rises from 0.92 to 1.11 ms. The eighth and tenth percentiles yield similar accuracy but increase latency to 1.27 and 1.34 ms, respectively. We therefore use the fifth percentile to balance accuracy and decoding cost.

Table 21: Sensitivity to the offline calibration percentile $100\alpha_c$. Decode latency is the average per-layer decoding latency in milliseconds. The pink-bold entries mark the default fifth percentile ($\alpha_c = 0.05$).

Percentile	**2**	**2.5**	**3**	**4**	**5**	**8**	**10**
Accuracy	44.37	45.36	47.24	48.01	**48.08**	48.09	48.04
Decode latency	0.92	0.94	0.99	1.08	**1.11**	1.27	1.34

F.5 Calibration Output Length Sensitivity

For short-input reasoning or code-generation prompts, the first decoded tokens may not create real retrieval pressure: if the active budget is $B = 1024$ and the input has only 100–200 tokens, the full historical KV cache can still fit inside the active cache for hundreds of generated tokens. We therefore do not calibrate by simply generating a fixed number of tokens. Instead, for each calibration

prompt x_i, we collect valid steps $\mathcal{T}_i = \{t : |H_t^{(i)}| > B,\ t > t_{\text{warm}}\}$ and keep at most the first 512 valid overlap samples. Varying this cap controls the calibration trace length after retrieval pressure appears. In our default setting, 512 valid samples provide enough overlap observations for generation-dominant and code profiles, while avoiding unnecessarily long offline traces. Long-input profiles are less sensitive because the cache already exceeds the budget at the first decode step.

G Broader Impacts and Responsible Release

AsdaKV is a systems method for reducing the memory traffic and latency of long-context LLM inference. Its main positive impact is practical: by lowering the cost of serving long prompts and long generations, it can make long-context applications more accessible, reduce hardware requirements, and improve energy efficiency for workloads such as document analysis, code assistance, and scientific reasoning. These benefits are indirect, because AsdaKV does not change the training data, model weights, or task objective of the underlying language model.

The same efficiency gains can also have negative downstream effects. Cheaper long-context inference may lower the cost of using existing LLMs for harmful applications, including large-scale spam, disinformation, or privacy-invasive analysis, if the underlying model and deployment environment are not governed responsibly. AsdaKV also moves historical KV states between GPU and CPU memory; in real deployments, these runtime states should be treated as sensitive request data and protected by the same isolation, retention, and access-control policies used for prompts and model outputs.

The planned release consists of code for a KV-cache retrieval system and does not include a new pretrained model, image generator, scraped dataset, or other high-risk asset. The experiments use public benchmarks and existing model families cited in the paper. We therefore do not anticipate the need for additional safeguards beyond the safety policies, monitoring, and usage restrictions of the host model and serving platform. Deployments should clear or isolate CPU-side KV pages after a request ends.

H AsdaKV Algorithm

Algorithm 1 summarizes the online decoding procedure. The drift threshold and stride parameters are read from an offline profile table before decoding starts. No request-local drift calibration is performed on the TTFT path. The resulting stride controls how often AsdaKV launches semantic drift detection, while the physical page size controls when a pending decode-window boundary can be materialized and offloaded. Page recall is deferred: a drift check can build a recall update at the beginning of a step, but the update is submitted only after the current attention has consumed the active cache that was already resident on GPU. Overlap, stride, and recall state are maintained per semantic head; head indices are suppressed in the corresponding update lines below.

Algorithm 1: AsdaKV Decoding with Strided Drift Detection and Deferred Recall

Input: query/key/value (q_t, k_t, v_t) at decode step t; workload profile c; profile table $\mathcal{P}$; per-head anchor matrices $\{A_t^{(h)}\}$; CPU page store $\mathcal{M}_{cpu}$; GPU active cache $\mathcal{S}$; current decode window $\mathcal{W}_{dec}$; semantic-head indices $\mathcal{H}_{sem}$; selected-window count K_w; page size P in tokens; active-cache budget B in tokens

State: drift stride δ; previous detection step t_{last}; previous detected window-ID set $\mathcal{W}_{prev}$; loaded-page mask $\mathcal{L}$; pending window-cut flag c_{cut}; pending recall event e_{recall}

$(\tau_c, b_c^{\text{low}}, b_c^{\text{high}}) \leftarrow \mathcal{P}[c]$

// Make the deferred update from an earlier step visible before this step uses the active cache.

if $e_{recall} \neq \emptyset$ **then**

  Wait(e_{recall}); CommitPendingPages($\mathcal{S}, \mathcal{L}$); $e_{recall} \leftarrow \emptyset$

end if

$\mathcal{R}_{plan} \leftarrow \emptyset$

// Drift detection is launched only on stride-selected decode steps.

if $t - t_{\text{last}} \geq \delta$ **then**

  for $h \in \mathcal{H}_{sem}$ **do**

    $s^{(h)} \leftarrow A_t^{(h)} q_t^{(h)}$

    $\mathcal{W}_t^{(h)} \leftarrow \text{TopK}(s^{(h)}, K_w)$

  end for

  $\bar{\Omega}_t \leftarrow |\mathcal{W}_t \cap \mathcal{W}_{prev}| / |\mathcal{W}_t \cup \mathcal{W}_{prev}|$

  if $\bar{\Omega}_t < \tau_c$ **then**

    $c_{cut} \leftarrow \text{true}$

    $\mathcal{R}_{plan} \leftarrow$ BuildPageRecall($\mathcal{W}_t, \mathcal{M}_{cpu}, \mathcal{L}, B$)

    $\delta \leftarrow 1$

  else if $\bar{\Omega}_t < b_c^{\text{low}}$ **then**

    $\delta \leftarrow 2$

  else if $\bar{\Omega}_t < b_c^{\text{high}}$ **then**

    $\delta \leftarrow 3$

  else

    $\delta \leftarrow 4$

  end if

  $\mathcal{W}_{prev} \leftarrow \mathcal{W}_t$; $t_{\text{last}} \leftarrow t$

else

  // No query-anchor dot product, top-k, or overlap decision is performed.

end if

// Current attention still uses the old active cache, before this step's recall plan is applied.

Append($\mathcal{S}, (k_t, v_t)$); Append($\mathcal{W}_{dec}, (k_t, v_t)$)

$o_t \leftarrow$ Attention($q_t, \mathcal{S}$)

if $\mathcal{R}_{plan} \neq \emptyset$ **then**

  $e_{recall} \leftarrow$ SubmitAsyncRecall($\mathcal{R}_{plan}$)

end if

// A detected semantic cut is materialized only at a physical page boundary.

if c_{cut} **and** $|\mathcal{W}_{dec}| \bmod P = 0$ **then**

  $\mathcal{P}_{new} \leftarrow$ FinalizePages($\mathcal{W}_{dec}$)

  Offload($\mathcal{P}_{new}, \mathcal{M}_{cpu}$); UpdateAnchors($\{A_t^{(h)}\}, \mathcal{P}_{new}$)

  $\mathcal{W}_{dec} \leftarrow \emptyset$; $c_{cut} \leftarrow \text{false}$

end if

return o_t

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the KV-cache bottleneck, distinguish eviction from retrieval, describe AsdaKV's attention-overlap retrieval mechanism, and summarize the main accuracy and efficiency claims.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The discussion section describes the evaluation scope, extending the token-level certificate to sparse window checks and deferred recall, the current focus on GQA models, and future work on MLA/MQA, calibration overhead, and multi-batch serving.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 3.1 states the retrieval abstraction, assumptions, and attention-overlap threshold certificate. Appendix A provides the support-change identity, pairing proofs for the proxy bounds, and the certificate proof. Appendix A.6 states the assumptions and derivation connecting token-level support to window-level detection.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper reports model backbones, baselines, datasets, KV budgets, experimental environments, and algorithm details. Appendix E.4 specifies calibration sources, sample counts, workload assignments, and alternative deployment rules.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The AsdaKV implementation will be made publicly available once code preparation is complete. The experiments use publicly available benchmark datasets and model families cited in the paper, and the appendix documents calibration data sources and evaluation settings.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experiments section specifies backbone models, baselines, datasets, KV budgets, and long-input and long-output settings. The appendix separates A100 and H200 measurements and provides calibration sources, sample counts, shared-threshold and deployment comparisons, ablations, and sensitivity analyses.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: The paper reports benchmark scores and mean throughput and latency, but does not report error bars, confidence intervals, or statistical significance tests. For the repeated timing measurements, five warm-up runs are discarded and the mean of five measured runs is reported; these averages alone do not quantify statistical significance.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The settings section reports the GPU, CPU, CPU–GPU interconnect, operating system, kernel version, and CUDA version used for the experiments; efficiency tables and figures report latency and throughput.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is an algorithmic and systems study for KV-cache retrieval using public benchmarks and released or cited model families, and we are not aware of any deviation from the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The appendix section on broader impacts and responsible release discusses positive impacts from lower long-context serving cost and energy use, as well as possible negative downstream impacts from making existing LLM capabilities cheaper to deploy.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The appendix section on broader impacts and responsible release explains that the planned release consists of system code rather than a new high-risk pretrained model, image generator, or scraped dataset, so no additional responsible-release safeguards for such assets are required.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing benchmarks, model families, and baseline methods are cited in the paper. The work does not redistribute those assets; it evaluates against publicly available or previously published resources.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: The AsdaKV implementation and accompanying documentation will be made publicly available once code preparation is complete. The paper and appendix document the algorithm, system components, calibration data and procedures, and evaluation settings needed to interpret the implementation.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used only as auxiliary tools for writing and editing paper text and for simple implementation assistance, such as plotting scripts and experiment configuration. They were not used as an important, original, or non-standard component of the core method; the authors are responsible for the main ideas, system design, and scientific claims.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.