READSPAN: ANTICIPATING WHERE DECODING READS FOR KV CACHE EVICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Key-value (KV) cache eviction reduces the memory of long-context inference by keeping only the input positions that an observation window at the end of the input attends to. This snapshot misses a recurring pattern of grounded responses: decoding jumps from the cue that the window attends to onto a nearby entry point and then reads forward through a span that the window barely sees and eviction removes first. We propose ReadSpan, a training-free method that anticipates this span from the first output token, which prefill predicts before any eviction. On top of a head-matched base selection, ReadSpan routes the window scores to the positions holding the first output token, extends them forward, and shifts each head's budget toward the span only to the extent that the base selection misses it, without any extra forward pass. On LongBench and RULER with Llama-3.1-8B-Instruct and Qwen3-4B, ReadSpan outperforms six training-free baselines on both benchmarks in all four model–budget settings; at 5% KV cache, it raises the RULER average over the strongest baseline by 3.4 and 7.2 points.
Code and data: anonymous.4open.science/r/ReadSpan-5443.

1 INTRODUCTION

Large language models (LLMs) increasingly process long inputs, including document collections in retrieval-augmented generation (Lewis et al., 2020), code repositories (Liu et al., 2023a), lengthy reports (Huang et al., 2021), and multi-step agent trajectories. Their responses are often *grounded* in the input, quoting a passage, calling a function defined elsewhere, or restating a finding. Serving such inputs is bottlenecked by the key-value (KV) cache, which stores the keys and values of every input token at every layer and grows linearly with the input length; for Llama-3.1-8B (Llama Team, 2024), a single 128K-token input occupies 16 GiB in half precision, more than the model weights. Reducing this cache without retraining is thus central to efficient long-context inference.

KV cache eviction keeps a subset of positions after the model has processed the input (*prefill*). The dominant paradigm relies on an *observation window*, the last few input tokens, which usually carry the question or instruction (Li et al., 2024); the positions that receive the most attention from these tokens, their *window score*, are kept within a fixed budget. Later work refines the scores and the allocation across layers and heads (Feng et al., 2025b; 2026; An et al., 2026; Cai et al., 2024; Feng et al., 2025a), but all of it shares one assumption: the window's attention at the end of prefill, a single snapshot, stands in for the queries of every later decoding step of the response.

This assumption breaks for grounded responses, because decoding *reads* the input rather than revisiting a fixed set of positions (Figure 1). In document QA, the question makes the window attend to *"harbor bridge"*, the *cue*, but the answer begins a few tokens later at *"Admiral"*, the *entry point*, and continues with *"Grace Hollis"*. Likewise, a code completion re-reads the parameter list of the function that its call site attends to, and a summary rewrites the sentence that follows the heading its instruction attends to. Decoding thus advances from the entry point through a contiguous *reading span*, typically attending to the position right after the one just read, a behavior attributed to induction and retrieval heads (Olsson et al., 2022; Wu et al., 2025). The window has little reason to attend beyond the cue, so the span receives low scores and is among the first positions to be evicted. On RULER needle retrieval, a strong window-based selection keeps only 55.8% of the reading-span tokens at a 5% budget, even though every answer is copied from such a span (§3.3).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

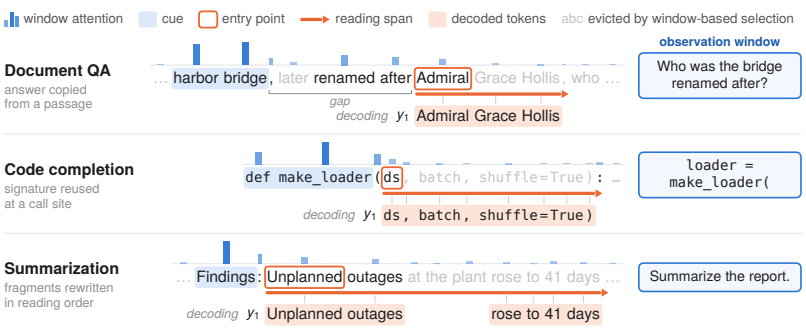

Figure 1: **The observation window sees where the model looks, not where it reads.** Schematic examples from three task families, in which the response copies or rewrites a span that starts shortly after the attended cue; gray tokens are evicted by window-based selection under a tight budget.

Anticipating a reading span seems to require the queries of future decoding steps, which prior work approximates by drafting a response or appending pseudo queries, at the cost of extra forward computation (Wang et al., 2025; Galim et al., 2025; Liu et al., 2025a; Tian et al., 2026). Yet prefill already predicts the first output token y_1, which is fixed before any eviction takes place. When y_1 occurs shortly after a cue in the input, it marks a likely entry point, as *"Admiral"* does in Figure 1. Building on this, we propose ReadSpan, a training-free eviction method with two stages. A head-matched base selection (HM) scores every position by its contribution to the window's attention output under grouped-query attention and allocates the budget across layers and heads. An entry-anchored refinement then uses y_1 to jump from the cue to the entry point, extends the selection forward along the anticipated span, and corrects the base selection only to the extent that the base misses the span. The refinement preserves every head's budget, needs no additional forward pass, and falls back exactly to the base selection whenever y_1 does not occur in the input. Our contributions are as follows:

- We identify the *jump-and-read* pattern of grounded decoding and show that window-based scores miss much of the span that decoding reads, even after smoothing scores over neighbors.
- We propose ReadSpan, which anchors the span on the first output token, extends it forward from the entry point, and applies a coverage-aware correction that keeps every head's budget and provably recedes as the base selection covers the span, without a budget-dependent switch.
- On LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024) with two models, ReadSpan attains the best averages on both benchmarks in all four model–budget settings against six training-free methods, and at 5% it raises the RULER average over the strongest baseline by 3.42 and 7.15 points, at a one-time cost of 9–40% of the prefill time, a share that shrinks with input length.

2 Related Work

KV cache compression. Besides eviction, the KV cache can be shrunk by sharing KV heads (Shazeer, 2019; Ainslie et al., 2023) or quantizing keys and values (Liu et al., 2024; Hooper et al., 2024), and query-aware page selection (Tang et al., 2024b) or an external context memory (Xiao et al., 2024a) reduces the cache accessed at each step. Eviction removes positions altogether, reducing memory and the data read per step, and is complementary to these techniques.

Window-based eviction. Early methods evict during decoding by accumulated attention (Zhang et al., 2023) or persistent token importance (Liu et al., 2023b), or keep only the initial and most recent tokens (Xiao et al., 2024b). SnapKV (Li et al., 2024) introduced the observation window, and later methods refine its score with value vectors and output projections (Feng et al., 2025b), worst-case-aware aggregation (Feng et al., 2026), or layer-wise output reconstruction (An et al., 2026), and its budget with uneven allocation across layers and heads (Cai et al., 2024; Feng et al., 2025a). Others favor contiguity by pooling scores over neighboring positions (Li et al., 2024) or keeping whole semantic chunks (Liu et al., 2025b), which protects the neighbors of high-scoring positions, whereas a reading span begins at an entry point that may lie tens of tokens away. ReadSpan keeps these scores and allocators and refines the selection by where decoding will read.

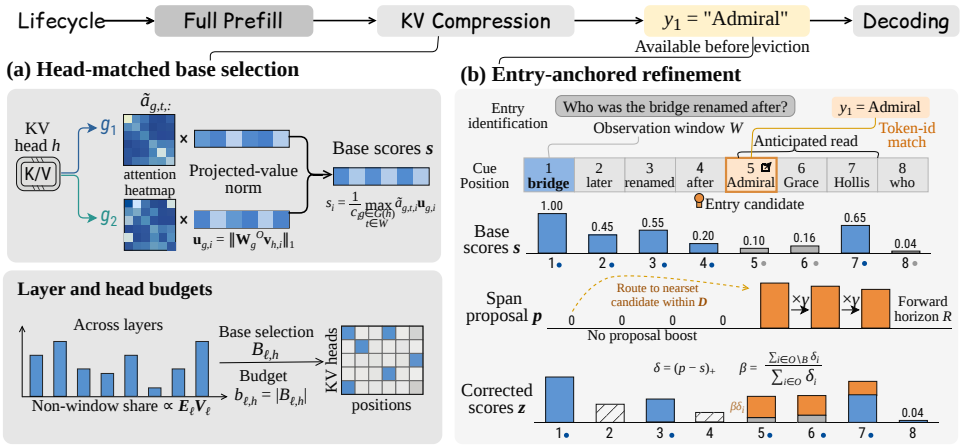

Figure 2: **Overview of ReadSpan.** (a) Head-matched base selection scores each position by its contribution to the window output and allocates the budget across layers and heads. (b) Entry-anchored refinement routes the window scores to the positions holding the first output token y_1, extends them forward, and corrects the base scores under the same per-head KV budget as the base.

Anticipating decoding-time queries. Lookahead Q-Cache (Wang et al., 2025) uses the queries of a short pseudo response, SpecKV (Galim et al., 2025) and Speculative Prefill (Liu et al., 2025a) rely on a small draft model, DapQ (Tian et al., 2026) appends position-aware pseudo queries to the input, and KVzip (Kim et al., 2025) scores positions by how much they help reconstruct the input; all of them spend extra forward computation. Expected Attention (Devoto et al., 2025) instead models the distribution of future queries. Lookahead Q-Cache also finds that even one or two generated tokens markedly improve the recall of the selection, in line with our observation; ReadSpan uses only the identity of the first output token, which prefill already provides, and never computes its query.

How transformers read their context. Induction heads attend to the token that followed an earlier occurrence of the current token (Olsson et al., 2022), and a sparse set of retrieval heads copies information from long contexts (Wu et al., 2025); RazorAttention (Tang et al., 2024a) and DuoAttention (Xiao et al., 2025) keep the full cache of such heads. ReadSpan builds on the same behavior at a different granularity: instead of protecting the heads that read, it protects the positions they will read next, within the same per-head budget as the base selection that it refines.

3 Head-Matched Selection with Entry-Anchored Refinement

ReadSpan compresses the KV cache once, between prefill and decoding, in two stages (Figure 2). The first stage estimates how much each position matters to the observation window: a head-matched scorer (HM) measures the contribution of every position to the attention output of the window, and an allocator distributes the budget across layers and heads. The second stage turns to what the window does not see, namely where decoding will start reading, and moves part of each head's budget toward the anticipated reading span without changing its size. We fix the setting (§3.1), present HM (§3.2), analyze what it misses (§3.3), and then develop the refinement (§3.4–§3.5).

3.1 Window-Based Eviction

Setting and notation. Consider an input $X = (x_0, \ldots, x_{N-1})$ and a decoder-only model with L layers, each with H KV heads; under grouped-query attention (GQA) (Ainslie et al., 2023), each KV head h is shared by a group $\mathcal{G}(h)$ of query heads. Prefill computes the keys and values of all N positions at every layer and KV head, and ends with the distribution of the first output token. Window-based eviction takes the last W positions, $\mathcal{W} = \{N - W, \ldots, N - 1\}$, as the *observation window*; these usually hold the question or instruction and are always kept, while eviction acts only on the older positions $\mathcal{O} = \{0, \ldots, N - W - 1\}$, which make up the remaining part of the input.

Scores, budgets, and selection. A window-based method has two components: a *scorer*, which assigns every older position i a score $s_{\ell,h,i} \geq 0$ at layer ℓ and KV head h from the attention that i receives from the window queries, and an *allocator*, which gives each head a budget of $b_{\ell,h}$ older positions. Each head then keeps

$$\mathcal{B}_{\ell,h} = \mathrm{Top}_{b_{\ell,h}}\big(s_{\ell,h,:}\big), \qquad \mathcal{S}_{\ell,h} = \mathcal{B}_{\ell,h} \cup \mathcal{W}, \tag{1}$$

where Top_b returns the indices of the b largest entries, and discards all other positions; kept entries retain their original positions. We call $\mathcal{B}_{\ell,h}$ the *base selection*. Ideally, $\mathcal{S}_{\ell,h}$ would contain every position that later decoding steps attend to; since their queries are unknown, the window queries stand in for them. Both stages of ReadSpan fit this template: HM specifies the scorer and the allocator, and the refinement changes *which* $b_{\ell,h}$ older positions each head keeps, but not how many.

3.2 Head-Matched Base Selection

Output contribution under GQA. A kept entry matters to the extent that it shapes the attention output. The key and value of position i at KV head h are read by every query head $g \in \mathcal{G}(h)$, and each query head passes its output through its own block $W^O_{\ell,g}$ of the output projection. For a window query at position t, position i thus contributes $\alpha_{\ell,g,t,i}\, W^O_{\ell,g}\, v_{\ell,h,i}$ to the layer output through g, an attention weight times a value projected by the *same* query head. Value-aware scorers typically aggregate the two factors separately over a group, which can pair the attention of one query head with the projected value of another; HM instead keeps the two factors matched for each query head.

Head-matched score. HM computes $\alpha_{\ell,g,t,i}$ for $t \in \mathcal{W}$ at the original rotary positions (Su et al., 2021), without renormalizing over $\mathcal{O}$, and, as in SnapKV (Li et al., 2024), smooths each row by average pooling of width 5; the pooled row, rescaled to keep its mass on $\mathcal{O}$, is $\tilde{\alpha}_{\ell,g,t,i}$. HM then scores

$$u_{\ell,g,i} = \big\| W^O_{\ell,g}\, v_{\ell,h(g),i} \big\|_1, \qquad s_{\ell,h,i} = \frac{1}{c_{\ell,i}} \max_{g \in \mathcal{G}(h),\ t \in \mathcal{W}} \tilde{\alpha}_{\ell,g,t,i}\, u_{\ell,g,i}, \tag{2}$$

where $h(g)$ is the KV head read by g and $c_{\ell,i} = \max\big(\sum_{h'} \frac{1}{|\mathcal{G}(h')|} \sum_{g \in \mathcal{G}(h')} u_{\ell,g,i},\ \epsilon\big)$ is the total projected-value norm of position i over the KV heads of its layer. The maximum keeps a position whenever one window query, through one query head, draws a large contribution from it; the denominator removes the magnitude that a position's values share across all heads and puts all heads of a layer on a common scale for the head-level budget competition that we describe below.

Layer and head budgets. With a retention ratio ρ, every KV head keeps $\lfloor \rho N \rfloor$ entries on average, window included. Each layer first receives room for its window, and the remaining entries are split across layers in proportion to $w_\ell = E_\ell V_\ell$, where E_ℓ is the entropy of the output-reconstruction scores of ReST-KV (An et al., 2026), high when importance is spread over many positions, and V_ℓ is the variance of the window attention across window queries, high when a single snapshot is less reliable (Appendix A.1). Within a layer, the KV heads compete for its entries through a single top-k over their concatenated scores, with window positions scored $+\infty$, so that a head whose scores stay high over many positions keeps more entries. The older positions that head h wins form its base selection $\mathcal{B}_{\ell,h}$, and their number is its per-head budget $b_{\ell,h}$ (Algorithm 1, Appendix A.3).

3.3 Reading Spans beyond the Observation Window

Missed reading spans. HM estimates how much a position matters to the window, but a grounded response also needs positions that the window barely attends to. We make the notions of Figure 1 precise. A *reading span* is a maximal run of consecutive input positions $(a, a{+}1, \ldots, a{+}m{-}1)$ whose tokens the response reproduces in order; its first position a is the *entry point*, and the *cue* is a high-scoring position before a through which the window locates the span. To gauge how well a window-based selection keeps such spans, we examined 96 RULER needle-retrieval inputs with Llama-3.1-8B-Instruct: every input contains a reading span of at least four tokens, yet at a 5% budget the HM base selection keeps on average only 55.8% of their tokens, although its scores are already smoothed over neighboring positions. Because decoding reads a span one position after another, every evicted position inside the span is a point where the copied answer can break.

Jump and read. Reaching a span from the window involves two moves: a *jump* from the cue to the entry point across a gap whose tokens decoding never reads, such as *", later renamed after"* in Figure 1, and a *read* that starts at the entry point and advances one position at a time. The window score reflects neither move: it peaks at the cue and carries no information about where the jump lands or how long the read lasts. Contiguity-based selection, by pooling or chunking, widens the kept region around high-scoring positions by a fixed amount in both directions and so treats the two moves alike: a region wide enough to cover the span also covers the gap and the text before the cue. Under a fixed budget, this suggests three principles: (i) the jump should land on an explicit entry candidate, without decaying across the gap or rewarding the gap itself; (ii) the read should extend forward from the entry point rather than from the cue; and (iii) since every added position displaces one that the base kept, the refinement should act only to the extent that the base selection misses the span. Sections 3.4 and 3.5 realize principles (i)–(ii) and principle (iii), respectively.

3.4 Entry-Anchored Span Proposal

The refinement treats every layer and KV head independently, so we drop the indices ℓ, h and write s, b, and $\mathcal{B}$ for the scores, the budget, and the base selection of a single head.

Entry candidates from the first output token. Prefill ends with the distribution of the first output token, and decoding emits y_1 from it (the argmax under greedy decoding) before any position is evicted, so y_1 is available at no cost and unaffected by eviction. When the response opens by reading a span, y_1 is the token at its entry point. We thus take the older positions that hold this token, matched by token id, as *entry candidates*:

$$\mathcal{A} = \{\, i \in \mathcal{O} : x_i = y_1 \,\}. \tag{3}$$

They are candidates rather than answers, since y_1 may occur in many places, so the next step lets the window scores decide which candidates matter; if $\mathcal{A}$ is empty, ReadSpan keeps the base selection.

Score routing to entry candidates. A candidate matters when the window can reach it, that is, when it follows a cue across a short gap. Each older position j therefore passes its score forward to the nearest candidate at or after it within a distance D:

$$\pi(j) = \min\{\, a \in \mathcal{A} : j \le a \le j + D \,\}, \qquad e_a = \max_{j:\, \pi(j)=a} s_j, \tag{4}$$

where a position with no candidate within D passes nothing, and $e_a = 0$ wherever nothing arrives. The *entry score* e_a is thus the highest window score that reaches a, so a match surrounded by low-scoring text stays low; as principle (i) requires, the score arrives undiminished by the length of the gap, and the gap itself gains nothing. Because each source reaches only its nearest candidate, a cue followed by several matches does not spread its score over all of them. The map π depends only on the input tokens and is shared by all heads, while each head routes its own scores.

Forward extension from entry points. Following principle (ii), the read continues from each entry point over the positions after it, which we express by extending the entry scores forward with a geometric discount:

$$p_i = \max_{\substack{0 \le r \le R \\ i-r \in \mathcal{O}}} \gamma^{\,r}\, e_{i-r}. \tag{5}$$

An entry point and up to R positions after it thus receive a score that decays with the offset r; R bounds the span anticipated from one entry, and $\gamma < 1$ reflects that positions deeper into a span are less certain to be read, without predicting the length of the response. We call p the *span proposal* and combine it with s as $f_i = \max(s_i, p_i)$, which raises the anticipated span and never lowers s.

3.5 Coverage-Aware, Budget-Preserving Selection

Uncovered share of the proposal. Ranking positions by f instead of s may seem harmless, since $f \ge s$ everywhere, but under a fixed budget every position that f lifts into the top b pushes out one that the base selection kept. When the base selection already contains most of the proposed span, as becomes likely at larger budgets, these exchanges give up kept positions for positions that add

little. We therefore measure the proposal by its increment over the base scores, $\delta_i = f_i - s_i = \max(0,\, p_i - s_i)$, and compute the share of this increment that falls outside the base selection:

$$\beta = \frac{\sum_{i \in \mathcal{O} \setminus \mathcal{B}} \delta_i}{\sum_{i \in \mathcal{O}} \delta_i}, \qquad z_i = s_i + \beta\, \delta_i, \tag{6}$$

with $\beta = 0$ when the proposal adds nothing ($\sum_i \delta_i = 0$). The corrected score z interpolates between the base scores ($\beta = 0$) and the full proposal ($\beta = 1$): when most of the increment lies outside $\mathcal{B}$, the base selection misses the span and z stays close to f; when most of it lies inside, the base already keeps the span and z stays close to s. The share β is computed per layer and head from quantities at hand, with no tunable coefficient. Proposition 1 summarizes its behavior (proof in Appendix A.2).

Proposition 1 (Conservative correction)**.** *For every layer and head, (i) $0 \le \beta \le 1$, and hence $s_i \le z_i \le f_i$ for all $i \in \mathcal{O}$; (ii) for fixed s and p, β is non-increasing in the base selection: $\mathcal{B}_1 \subseteq \mathcal{B}_2$ implies $\beta(\mathcal{B}_2) \le \beta(\mathcal{B}_1)$; (iii) if $\mathcal{B}$ contains every position with $\delta_i > 0$, then $\beta = 0$ and $z = s$, so ReadSpan keeps exactly the base selection, up to ties among positions with equal scores.*

Part (ii) matters as budgets grow: once the base covers more of the proposal, the correction recedes on its own, with no budget-dependent switch. With $\mathcal{A} = \emptyset$ or $\beta = 0$, ReadSpan reduces to HM.

Selection under the base budgets. Finally, each head keeps $\mathcal{S} = \mathrm{Top}_b(z) \cup \mathcal{W}$ with the same $b = |\mathcal{B}|$ as in Eq. (1), and gathers the keys and values of $\mathcal{S}$ at their original positions. The number of entries per head, and hence the cache memory and the data read at each decoding step, are identical to those of HM, so any change in quality comes from which positions are kept rather than how many. Decoding then starts from y_1 and continues on the refined cache (Algorithm 2, Appendix A.3). The refinement uses only the input tokens, y_1, and the HM scores, with no forward computation of the model or of a draft model; the map π is a reverse cumulative minimum shared by all heads, and per head, routing and the uncovered share cost $O(N)$ and the extension $O(RN)$ (§4.6).

4 Experiments

We compare ReadSpan with six training-free eviction methods on LongBench and RULER (§4.1–§4.3), separate the contributions of its two stages (§4.4), and examine budgets (§4.5) and cost (§4.6).

4.1 Experimental Setup

Models and benchmarks. We use Llama-3.1-8B-Instruct (Llama Team, 2024) and Qwen3-4B (Qwen Team, 2025) in BF16, with the thinking mode of Qwen3 disabled and a fixed prompt template per model. From LongBench (Bai et al., 2024), we use the 16 English tasks (single- and multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion) with 100 fixed inputs per task; from RULER (Hsieh et al., 2024), we use all 13 tasks (eight needle-in-a-haystack (NIAH) retrieval variants, variable tracking, two aggregation tasks, and two QA tasks) at the nominal 16K configuration with 50 inputs per task. Each model is thus evaluated on 2,250 inputs per budget, with prompts of 0.8K–52.6K tokens (Appendix B.2). Each LongBench task uses its own metric and RULER uses string matching, all on a 0–100 scale; tasks are weighted equally within each benchmark, and the overall score weights the two benchmark averages equally (Appendix B.3).

Baselines. Four baselines score positions with the window: SnapKV (Li et al., 2024); CriticalKV (Feng et al., 2025b), which adds value vectors and output projections; Layer-DefensiveKV (Feng et al., 2026), which aggregates window attention in a worst-case-aware manner and allocates budgets across layers; and ReST-KV (An et al., 2026), which scores positions by layer-wise output reconstruction. The other two estimate future decoding queries: Expected Attention (Devoto et al., 2025) models their distribution, and DapQ (Tian et al., 2026) appends position-aware pseudo queries to the input. Each baseline keeps its own scoring and allocation, with AdaKV's allocation (Feng et al., 2025a) for CriticalKV and Expected Attention (Appendix B.1).

Budgets and decoding. We retain 5% and 10% of the KV cache of each tokenized prompt, distributed across layers and heads by each method's own allocator. Eviction is applied once, between prefill and decoding, and generated tokens are appended without further eviction. All methods

Table 1: **Results on LongBench** at 5% and 10% KV cache; Avg. weights the 16 tasks equally. Within each block, the best score per column is in **bold** and the second best underlined (none when several methods tie for the best); Full KV is an uncompressed reference and is not ranked.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		
Method	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	MuSiQue	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	LCC	RB-P	Avg.
Llama-3.1-8B-Instruct																	
Full KV	24.53	43.69	57.06	54.07	42.78	33.06	33.81	25.56	27.11	76.00	90.89	43.15	8.20	100.00	60.54	55.75	48.51
5% KV cache																	
SnapKV	23.51	29.16	55.32	53.07	40.65	**32.21**	23.05	24.43	18.32	59.00	89.32	41.68	8.14	**100.00**	55.51	54.54	44.25
CriticalKV	23.45	28.19	52.90	53.15	40.21	30.47	24.08	23.65	16.54	60.00	88.50	41.75	8.17	**100.00**	54.40	**56.61**	43.88
Layer-DefensiveKV	23.64	34.68	55.91	53.11	**43.41**	32.03	25.40	**24.44**	12.21	67.00	90.05	41.99	8.00	**100.00**	48.09	54.94	44.68
ReST-KV	**25.69**	31.63	56.07	**53.71**	41.57	31.53	24.93	24.40	19.05	70.00	**90.82**	43.69	8.00	**100.00**	55.94	54.36	45.71
Expected Attention	22.14	25.74	36.39	42.10	30.56	23.88	22.28	21.80	**21.74**	31.00	74.56	36.61	8.00	97.42	35.90	34.19	35.27
DapQ	25.25	24.02	53.90	51.75	42.48	30.29	24.48	24.43	19.85	56.00	90.53	**43.91**	**8.25**	**100.00**	54.43	52.36	43.87
ReadSpan	24.70	**36.61**	**57.71**	53.43	42.01	31.98	**25.65**	24.12	20.69	**76.00**	89.97	42.92	8.00	**100.00**	**59.48**	54.12	**46.71**
10% KV cache																	
SnapKV	24.98	36.92	56.60	**54.00**	42.17	32.03	26.79	24.51	20.69	66.00	**90.82**	42.81	**8.25**	**100.00**	57.03	54.44	46.13
CriticalKV	24.09	37.71	55.57	53.09	41.90	31.59	26.84	23.81	21.92	69.00	90.42	41.50	**8.25**	**100.00**	58.71	55.11	46.22
Layer-DefensiveKV	**25.19**	41.30	57.04	53.92	41.63	**32.36**	**29.40**	24.48	21.90	75.00	90.39	42.33	8.00	**100.00**	57.50	**55.74**	47.26
ReST-KV	25.02	39.01	56.60	53.60	42.80	31.98	27.01	24.70	21.84	73.00	90.64	43.66	**8.25**	**100.00**	56.63	54.71	46.84
Expected Attention	24.43	32.49	47.49	45.64	38.46	25.68	23.72	22.53	**22.86**	39.00	83.60	39.76	8.00	99.00	44.87	40.64	39.89
DapQ	25.16	31.58	55.55	52.76	**43.03**	32.14	26.35	**24.83**	21.53	61.00	90.10	43.19	8.20	**100.00**	55.25	53.51	45.26
ReadSpan	24.72	**41.68**	**58.44**	53.73	42.25	32.11	27.48	24.71	22.47	**76.00**	90.06	**44.61**	8.00	**100.00**	**59.78**	54.43	**47.53**
Qwen3-4B																	
Full KV	23.44	42.25	55.80	53.70	46.39	34.77	32.32	24.86	25.47	56.00	92.02	41.35	1.50	100.00	2.06	1.14	39.57
5% KV cache																	
SnapKV	24.42	33.20	53.13	53.83	44.54	35.23	22.37	24.24	15.79	43.50	91.36	39.72	0.00	**100.00**	5.38	1.95	36.79
CriticalKV	23.13	32.60	49.59	53.49	43.83	35.52	21.96	23.93	14.16	49.50	**91.76**	40.05	0.50	**100.00**	**6.11**	2.62	36.80
Layer-DefensiveKV	23.62	36.13	53.02	53.37	**45.40**	**36.24**	23.36	24.11	17.31	50.00	91.36	40.89	0.50	**100.00**	4.56	1.53	37.59
ReST-KV	23.11	32.66	50.47	53.08	44.42	35.07	23.45	22.59	17.10	**56.50**	89.76	40.96	1.00	**100.00**	2.90	0.95	37.13
Expected Attention	16.78	29.84	29.76	37.28	30.73	22.43	22.56	21.67	**20.38**	14.00	75.02	31.10	0.00	95.00	2.67	**4.36**	28.35
DapQ	24.19	32.05	49.99	51.90	42.48	34.07	22.88	24.47	17.51	40.00	**91.76**	**42.03**	0.50	**100.00**	2.86	1.76	36.15
ReadSpan	**24.57**	**37.29**	**53.53**	**54.31**	45.13	34.97	**24.68**	**24.57**	17.67	55.00	91.36	39.81	**1.50**	**100.00**	4.32	2.85	**38.22**
10% KV cache																	
SnapKV	23.80	36.43	53.54	**54.12**	44.81	35.37	26.08	25.03	18.77	52.00	91.36	40.21	**0.50**	**100.00**	3.14	0.85	37.88
CriticalKV	**24.72**	35.52	53.04	53.33	45.03	34.82	26.06	24.32	19.16	52.00	**92.02**	40.20	**0.50**	**100.00**	**3.74**	2.34	37.93
Layer-DefensiveKV	23.91	38.90	55.41	53.47	46.73	**35.98**	26.08	24.95	20.02	55.00	**92.02**	40.84	**0.50**	**100.00**	3.30	1.96	38.69
ReST-KV	23.36	35.72	53.45	53.98	45.86	34.49	26.65	24.18	20.07	**56.50**	91.86	41.12	**0.50**	**100.00**	2.49	0.93	38.20
Expected Attention	19.93	35.97	39.46	44.77	41.15	24.27	25.94	23.26	**22.23**	31.00	87.49	35.69	0.00	98.00	1.60	**2.45**	33.32
DapQ	23.65	36.45	52.76	53.68	45.89	35.45	25.24	**25.64**	19.46	46.00	91.59	**41.67**	**0.50**	**100.00**	3.31	2.10	37.71
ReadSpan	23.77	**39.15**	**55.70**	53.69	**46.98**	35.46	**27.41**	24.89	20.53	56.00	91.76	41.50	**0.50**	**100.00**	3.26	1.78	**38.90**

decode greedily: the first output token is taken from the uncompressed prefill, and generation continues on the compressed cache. ReadSpan uses $W = 32$ and pooling width 5 for HM and $D = 64$, $R = 32$, and $\gamma = 0.95$ for the refinement, shared by both models, both budgets, and all tasks.

4.2 Results on LongBench

Average scores. ReadSpan attains the highest LongBench average in all four settings (Table 1). At 5%, it leads the strongest baseline by 1.00 points on Llama-3.1-8B-Instruct and by 0.63 points on Qwen3-4B; at 10%, it leads by 0.27 and 0.21 points, respectively. The strongest baseline varies (ReST-KV or Layer-DefensiveKV), whereas ReadSpan ranks first in all four settings.

Sources of the gains. The gains concentrate on tasks whose answers are taken from a specific passage of the input, as the reading-span view predicts. ReadSpan obtains the best score on both extractive single-document QA tasks, Qasper and MultiFieldQA-en, in all four settings, by up to 1.93 and 1.64 points. On Llama-3.1-8B-Instruct, it also leads on TREC and LCC at both budgets, most clearly at 5% (76.00 vs. 70.00 and 59.48 vs. 55.94), as TREC answers are labels recurring in the demonstrations and LCC code often reuses earlier code; on Qwen3-4B, it ranks second on TREC, and all methods score below 7 on both code tasks. The answers of multi-document QA, MultiNews, and RepoBench-P instead draw on several documents or files, so the first output token is less likely to mark a single entry point; there, ReadSpan stays within 2.71 points of the best baseline.

Table 2: **RULER averages and overall scores.** RULER weights its 13 tasks equally, and Overall weights the LongBench and RULER averages equally. Among compressed methods, the best is in **bold** and the second best underlined; Full KV is an uncompressed reference.

	RULER average				Overall (LongBench + RULER)			
	Llama-3.1-8B		Qwen3-4B		Llama-3.1-8B		Qwen3-4B	
Method	5%	10%	5%	10%	5%	10%	5%	10%
Full KV	92.24		90.75		70.38		65.16	
SnapKV	73.98	82.56	74.00	83.99	59.11	64.34	55.39	60.94
CriticalKV	70.80	77.77	73.71	81.77	57.34	62.00	55.25	59.85
Layer-DefensiveKV	77.58	88.97	76.60	88.28	61.13	68.12	57.09	63.49
ReST-KV	85.32	87.76	80.58	85.95	65.51	67.30	58.85	62.07
Expected Attention	20.03	38.49	36.08	61.21	27.65	39.19	32.22	47.27
DapQ	71.83	80.45	60.59	74.70	57.85	62.86	48.37	56.21
ReadSpan	**88.74**	**90.29**	**87.73**	**89.47**	**67.73**	**68.91**	**62.98**	**64.18**

Table 3: **Component analysis on RULER** (13-task average). Upper block: refinement on top of HM; lower block: a single change to the full ReadSpan.

	Llama-3.1-8B		Qwen3-4B		
Variant	5%	10%	5%	10%	Mean
HM (base selection only)	85.96	90.21	84.83	**89.76**	87.69
Extension from all positions	88.42	90.19	87.41	89.28	88.83
Entry-anchored, β=1	88.73	90.24	87.36	89.07	88.85
ReadSpan	**88.74**	90.29	**87.73**	89.47	**89.06**
ReadSpan without routing	88.37	90.28	87.46	89.41	88.87
ReadSpan, uniform layer budgets	87.70	**90.56**	87.38	89.52	88.79
ReadSpan, uniform head budgets	87.82	89.79	86.37	89.32	88.32

Table 4: **Time to a compressed cache** (s): full prefill vs. prefill followed by the complete ReadSpan selection, and the relative overhead Δ. *Beyond the 32K native context length of Qwen3-4B.

	Llama-3.1-8B			Qwen3-4B		
Len.	Prefill	+Ours	Δ	Prefill	+Ours	Δ
8K	1.14	1.51	32%	0.80	1.12	40%
16K	2.65	3.18	20%	2.03	2.63	30%
32K	6.81	7.75	14%	5.70	6.76	19%
64K	19.84	21.68	9%	18.49*	20.27*	10%

4.3 Results on RULER

Average and overall scores. ReadSpan attains the highest RULER average and the highest overall score in all four settings (Table 2; all 13 tasks in Appendix B.5). At 5%, it leads the strongest baseline, ReST-KV for both models, by 3.42 points on Llama-3.1-8B-Instruct (88.74 vs. 85.32) and by 7.15 points on Qwen3-4B (87.73 vs. 80.58), and it leads SnapKV by 14.76 and 13.73 points. At 10%, its lead over the strongest baseline, now Layer-DefensiveKV, narrows to 1.32 and 1.19 points. On the overall score, it leads by 2.22 and 4.13 points at 5% and by 0.79 and 0.69 points at 10%. The narrower lead at the larger budget is consistent with the coverage-aware correction: as the base selection covers more of the anticipated spans, β shrinks and ReadSpan departs less from its base.

Long copied answers. The largest gains occur on S3 and K3, whose needle values are UUIDs rather than seven-digit numbers, so that answering means reading a long span after the key; ReadSpan scores at least 96 on both in all settings. At 5%, the best baseline on K3 reaches only 70 on Llama-3.1-8B-Instruct and 50 on Qwen3-4B, against 100 and 96 for ReadSpan; on S3, only ReST-KV comes close, while SnapKV scores 22 and 48. On the other retrieval tasks, variable tracking, and the two QA tasks, ReadSpan stays within 2 points of the best baseline; the aggregation tasks, which count words rather than copy a span, favor different methods. The query-estimation baselines do not help: DapQ trails SnapKV in all settings, and Expected Attention scores lowest.

4.4 Component Analysis

Head-matched base selection. HM alone is already strong (Table 3), and its allocation matters at both levels. Uniform head budgets lower all four scores and the mean from 89.06 to 88.32, while uniform layer budgets lower the 5% scores by 1.04 and 0.35 points and change the 10% ones by under 0.3 points; head-wise allocation thus helps consistently, and layer-wise mainly at 5%.

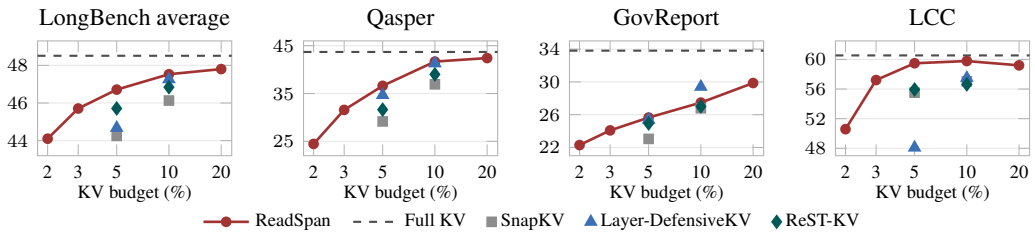

Figure 3: **ReadSpan across KV budgets** (LongBench, Llama-3.1-8B-Instruct). Dashed lines mark the full cache, and markers the three strongest window-based baselines at 5% and 10%.

Entry-anchored refinement. Adding the refinement to HM improves the RULER average by 2.78 and 2.90 points at 5% and changes it by $+0.08$ and -0.29 points at 10%, where the base selection already keeps most spans; the mean rises from 87.69 to 89.06. Fixing $\beta = 1$ lowers the four scores by 0.01–0.40 points, most on Qwen3-4B, where the base selection is already strong. Routing has little effect without the correction (88.85 vs. 88.83 mean) but helps consistently with it: removing it lowers the score in every setting, by 0.37, 0.01, 0.27, and 0.06 points, respectively. Routing thus places the proposal at the entry point, and the correction controls how strongly it changes the base.

4.5 Robustness across KV Budgets

Figure 3 extends the LongBench evaluation of Llama-3.1-8B-Instruct to budgets from 2% to 20% (per-task scores in Appendix B.4). The average of ReadSpan rises monotonically with the budget and retains 90.9%, 96.3%, and 98.5% of the full-cache average at 2%, 5%, and 20%. With 5% of the cache, ReadSpan already scores above SnapKV and CriticalKV at 10% (46.71 vs. 46.13 and 46.22), and with 3% it matches ReST-KV at 5% (45.71), reaching the same quality with roughly half the cache. Short-answer tasks such as HotpotQA, TriviaQA, and TREC change by at most 2.3 points over the range, and LCC saturates at 5%; Qasper improves steeply up to 10%, whereas GovReport and MultiNews keep improving up to 20%, as their summaries draw on the whole input.

4.6 Efficiency

ReadSpan adds only selection work between prefill and decoding (HM scoring, budget allocation, and refinement), implemented with fused GPU kernels; Table 4 compares it with a full prefill from 8K to 64K tokens. The overhead of 0.32–1.85 s shrinks in relative terms as the input grows, from 32.0% to 9.3% on Llama-3.1-8B-Instruct and from 39.7% to 9.6% on Qwen3-4B, as prefill attention is quadratic in the input length whereas the selection is roughly linear. The refinement itself takes only 56–67 ms and at most 6.3 MiB of memory, and the selection runs once, after which each decoding step reads a cache that is 10–20$\times$ smaller at the 5% and 10% budgets.

5 Conclusion

Window-based KV cache eviction misses a recurring pattern of grounded decoding: the response jumps from the cue that the window attends to onto a nearby entry point and then reads forward through a span that the window barely sees. ReadSpan anticipates this span from the first output token, which prefill provides before any eviction, extends the selection forward from the matched entry points, and moves each head's budget toward the span only to the extent that the base selection misses it. Without an extra forward pass or any change to the per-head budgets, it attains the best LongBench, RULER, and combined scores in all four settings, with the largest gains at tight budgets and on long copied answers. Because the refinement only changes which positions each head keeps, it can in principle be layered on other window-based scorers and allocators. Our evaluation covers two models and a single eviction step, and since one first output token anchors one entry point, ReadSpan helps less when the answer assembles content from several documents; anchoring on several decoded tokens, or refining the cache during decoding, is a natural extension of this work.

AI USE STATEMENT

In this work, we used generative AI tools solely to assist in searching for relevant baseline methods. We did not use generative AI tools for any other tasks. We manually reviewed the search results and checked them against the original sources. We take responsibility for the final content of this work, including any content informed by AI-assisted searches.

REPRODUCIBILITY STATEMENT

Our code and evaluation data are available at https://anonymous.4open.science/r/ReadSpan-5443. Section 3 specifies every step of ReadSpan, and Appendix A gives the exact scoring, budget allocation, pseudocode, and the proof of Proposition 1. Appendix B lists the configuration of every baseline, the prompt lengths, the generation limits, and the evaluation protocol; all hyperparameters are shared across models, budgets, and tasks, and decoding is greedy.

ETHICS STATEMENT

This work proposes a training-free method to reduce the memory of long-context inference by evicting entries from the KV cache. It involves no human subjects, crowdsourcing, or collection of personal data. All experiments use publicly available benchmarks, LongBench and RULER, and publicly released models, Llama-3.1-8B-Instruct and Qwen3-4B, each used under its original license. ReadSpan does not train or fine-tune any model and does not add new capabilities; it only changes which cached positions are kept, so the outputs inherit the biases and limitations of the underlying models. Because any compression of the cache can alter or degrade responses, users should validate compressed inference before deploying it in high-stakes applications. By lowering the memory and cost of long-context inference, this work may make long-document and retrieval-augmented applications more accessible and energy-efficient. We are not aware of other ethical concerns raised by this work.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.298. URL https://doi.org/10.18653/v1/2023.emnlp-main.298.

Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026. URL https://openreview.net/forum?id=PhEHuo7oMm.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.172. URL https://doi.org/10.18653/v1/2024.acl-long.172.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic KV cache compression based on pyramidal information funneling. *CoRR*, abs/2406.02069, 2024. doi: 10.48550/ARXIV.2406.02069. URL https://doi.org/10.48550/arXiv.2406.02069.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova,

Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 4599–4610. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.365. URL https://doi.org/10.18653/v1/2021.naacl-main.365.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution. *CoRR*, abs/2510.00636, 2025. doi: 10.48550/ARXIV.2510.00636. URL https://doi.org/10.48550/arXiv.2510.00636.

Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 1074–1084. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1102. URL https://doi.org/10.18653/v1/p19-1102.

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. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025a. URL http://papers.nips.cc/paper_files/paper/2025/hash/a40ff56daab9f4808b1e18350c8a11ce-Abstract-Conference.html.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Identify critical KV cache in LLM inference from an output perturbation perspective. *CoRR*, abs/2502.03805, 2025b. doi: 10.48550/ARXIV.2502.03805. URL https://doi.org/10.48550/arXiv.2502.03805.

Yuan Feng, Haoyu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. Taming the fragility of KV cache eviction in LLM inference. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026. URL https://arxiv.org/abs/2510.13334.

Kevin Galim, Ethan Ewer, Wonjun Kang, Minjae Lee, Hyung Il Koo, and Kangwook Lee. Draft-based approximate inference for llms. *CoRR*, abs/2506.08373, 2025. doi: 10.48550/ARXIV.2506.08373. URL https://doi.org/10.48550/arXiv.2506.08373.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *CoRR*, abs/1911.12237, 2019. URL http://arxiv.org/abs/1911.12237.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing A multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Núria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pp. 6609–6625. International Committee on Computational Linguistics, 2020. doi: 10.18653/V1/2020.COLING-MAIN.580. URL https://doi.org/10.18653/v1/2020.coling-main.580.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

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

Luyang Huang, Shuyang Cao, Nikolaus Nova Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 1419–1436. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.112. URL https://doi.org/10.18653/v1/2021.naacl-main.112.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 1601–1611. Association for Computational Linguistics, 2017. doi: 10.18653/V1/P17-1147. URL https://doi.org/10.18653/v1/P17-1147.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic KV cache compression with context reconstruction. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/f4eaa4b8f2d08edb3f0af990d56134ea-Abstract-Conference.html.

Tomás Kociský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *CoRR*, abs/1712.07040, 2017. URL http://arxiv.org/abs/1712.07040.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: LLM knows what you are looking for before generation. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Jingyu Liu, Beidi Chen, and Ce Zhang. Speculative prefill: Turbocharging TTFT with lightweight and training-free token importance estimation. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/liu25g.html.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *CoRR*, abs/2306.03091, 2023a. doi: 10.48550/ARXIV.2306.03091. URL https://doi.org/10.48550/arXiv.2306.03091.

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. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/2987f911151b39cd3a1761e212319e8e-Abstract-Conference.html.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *CoRR*, abs/2209.11895, 2022. doi: 10.48550/ARXIV.2209.11895. URL https://doi.org/10.48550/arXiv.2209.11895.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient KV cache compression through retrieval heads. *CoRR*, abs/2407.15891, 2024a. doi: 10.48550/ARXIV.2407.15891. URL https://doi.org/10.48550/arXiv.2407.15891.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR / OpenReview.net, 2024b. URL https://proceedings.mlr.press/v235/tang24l.html.

Zhenxu Tian, Yi Su, Juntao Li, and Min Zhang. Where matters more than what: Decoding-aligned KV cache compression via position-aware pseudo queries. *CoRR*, abs/2603.11564, 2026. doi: 10.48550/ARXIV.2603.11564. URL https://doi.org/10.48550/arXiv.2603.11564.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multi-hop questions via single-hop question composition. *CoRR*, abs/2108.00573, 2021. URL https://arxiv.org/abs/2108.00573.

Yixuan Wang, Shiyu Ji, Yijun Liu, Yuzhuang Xu, Yang Xu, Qingfu Zhu, and Wanxiang Che. Lookahead q-cache: Achieving more consistent KV cache eviction via pseudo query. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 34158–34174. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1732. URL https://doi.org/10.18653/v1/2025.emnlp-main.1732.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025. URL https://arxiv.org/abs/2404.15574.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025. URL https://arxiv.org/abs/2410.10819.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir R. Radev. Qmsum: A new benchmark for query-based multi-domain meeting summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021, pp. 5905–5921. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.472. URL https://doi.org/10.18653/v1/2021.naacl-main.472.

A Method Details

A.1 HM Scoring and Budget Allocation

Smoothed window attention. For query head g at layer ℓ, window position $t \in \mathcal{W}$, and older position $i \in \mathcal{O}$, let $\alpha_{\ell,g,t,i}$ be the causal attention weight computed with the original rotary positions; the softmax runs over all positions visible to t, and the weights on $\mathcal{O}$ are not renormalized. Each row is smoothed by average pooling of width 5 with stride 1 and zero padding over the older positions, and rescaled to keep its mass on $\mathcal{O}$:

$$\bar{\alpha}_{\ell,g,t,:} = \mathrm{AvgPool}_5\big(\alpha_{\ell,g,t,\mathcal{O}}\big), \qquad \tilde{\alpha}_{\ell,g,t,i} = \bar{\alpha}_{\ell,g,t,i}\,\frac{\sum_{j\in\mathcal{O}} \alpha_{\ell,g,t,j}}{\max\big(\sum_{j\in\mathcal{O}} \bar{\alpha}_{\ell,g,t,j},\, \epsilon\big)}. \tag{7}$$

Head-matched score. With $u_{\ell,g,i} = \|W^O_{\ell,g} v_{\ell,h(g),i}\|_1$, the mean projected-value norm of KV head h is $\bar{u}_{\ell,h,i} = \frac{1}{|\mathcal{G}(h)|}\sum_{g\in\mathcal{G}(h)} u_{\ell,g,i}$, the normalizer is $c_{\ell,i} = \max\big(\sum_h \bar{u}_{\ell,h,i},\, \epsilon\big)$, and the score is Eq. (2). The pairing inside the maximum is the essential part: the attention of query head g is multiplied by the value projected through the block $W^O_{\ell,g}$ of the same head, never by a norm averaged over the group.

Layer budgets. With retention ratio ρ, the average number of entries per KV head is $\lfloor \rho N \rfloor$; this count includes the window and the bytes reserved for per-head metadata, so all methods are compared at the same nominal ratio. Let $\psi_{\ell,h,i}$ be the layer-wise output-reconstruction score of ReST-KV (An et al., 2026) with its spatial smoothing, and $\bar{\psi}_{\ell,h,i}$ its normalization to unit sum over the older positions and KV heads of layer ℓ. The two factors of the layer preference are

$$E_\ell = -\frac{1}{H\,|\mathcal{O}|}\sum_h \sum_{i\in\mathcal{O}} \bar{\psi}_{\ell,h,i} \log \bar{\psi}_{\ell,h,i}, \qquad V_\ell = \frac{1}{H_q}\sum_g \sum_{i\in\mathcal{O}} \mathrm{Var}_{t\in\mathcal{W}}\left(\alpha_{\ell,g,t,i}\right), \tag{8}$$

where H_q is the number of query heads per layer and the variance is the population variance over the W window queries. Each layer first receives W entries per head for its window; the remaining entries are split across layers in proportion to $w_\ell = E_\ell V_\ell$, capped by the number of positions available in the layer, with zero weights and integer remainders resolved so that the layers together hold exactly $LH\lfloor \rho N \rfloor$ entries. Because the preference of every layer must be known before the split, the allocation runs once, after the full prefill has finished for all layers.

Head budgets. Within layer ℓ, the scores of all KV heads are concatenated, window positions are scored $+\infty$, and a single top-k with k equal to the layer's number of entries selects the kept pairs (h, i). Every head thus keeps its window, and the number of older positions $b_{\ell,h}$ that it wins varies across heads. The refinement of §3.4–§3.5 keeps every per-head budget $b_{\ell,h}$ fixed.

A.2 Proof of Proposition 1

Fix a layer and a head, and recall that $\delta_i = \max(0, p_i - s_i) \geq 0$ depends on s and p but not on $\mathcal{B}$.

(i) If $\sum_{i\in\mathcal{O}} \delta_i = 0$, then $\beta = 0$ by definition. Otherwise the numerator is a sub-sum of the non-negative terms in the denominator, so $0 \leq \beta \leq 1$, and $z_i = s_i + \beta\delta_i$ lies between s_i and $s_i + \delta_i = f_i$.

(ii) If $\mathcal{B}_1 \subseteq \mathcal{B}_2$, then $\mathcal{O} \setminus \mathcal{B}_2 \subseteq \mathcal{O} \setminus \mathcal{B}_1$. Since all $\delta_i \geq 0$, the numerator for $\mathcal{B}_2$ is at most that for $\mathcal{B}_1$, while the denominator is the same; if the denominator is zero, both shares are zero.

(iii) If $\delta_i = 0$ for all $i \notin \mathcal{B}$, the numerator vanishes and $\beta = 0$, so $z = s$ and $\mathrm{Top}_b(z) = \mathrm{Top}_b(s) = \mathcal{B}$ up to the tie-breaking rule. □

A.3 ALGORITHMS

Algorithm 1 summarizes the head-matched base selection of Section 3.2, and Algorithm 2 the entry-anchored refinement of Sections 3.4–3.5, which runs once between prefill and decoding.

Algorithm 1 Head-matched base selection (HM), run once after prefill

Require: window attention α; values v; output-projection blocks W^O; reconstruction scores ψ; ratio ρ

for each layer ℓ **do** ▷ scoring
  $\tilde{\alpha}_\ell \leftarrow$ width-5 average pooling of $\alpha_{\ell,:,:,\mathcal{O}}$, rescaled to its mass on $\mathcal{O}$
  $u_{\ell,g,i} \leftarrow \|W^O_{\ell,g}\, v_{\ell,h(g),i}\|_1$ for every query head g ▷ head-matched value norm
  $s_{\ell,h,i} \leftarrow \max_{g\in\mathcal{G}(h),\, t\in\mathcal{W}} \tilde{\alpha}_{\ell,g,t,i}\, u_{\ell,g,i} \,/\, c_{\ell,i}$ ▷ Eq. (2)
  $w_\ell \leftarrow E_\ell \cdot V_\ell$ ▷ Eq. (8)
end for
$C_\ell \leftarrow HW +$ share of the other $LH(\lfloor \rho N \rfloor - W)$ entries, $\propto w_\ell$ ▷ layer budgets
for each layer ℓ **do** ▷ head budgets
  $\mathcal{T}_\ell \leftarrow \mathrm{Top}_{C_\ell}$ of $s_{\ell,h,i}$ over all pairs (h,i), with $s_{\ell,h,i} = +\infty$ for $i \in \mathcal{W}$
  $\mathcal{B}_{\ell,h} \leftarrow \{i \in \mathcal{O} : (h,i) \in \mathcal{T}_\ell\}$; $b_{\ell,h} \leftarrow |\mathcal{B}_{\ell,h}|$
end for

Algorithm 2 Entry-anchored refinement of ReadSpan, applied once between prefill and decoding

Require: input tokens $x_{0:N-1}$; first output token y_1; from HM (Algorithm 1), for every layer ℓ and KV head h: scores $s_{\ell,h,:}$, base selection $\mathcal{B}_{\ell,h}$, budget $b_{\ell,h} = |\mathcal{B}_{\ell,h}|$; window $\mathcal{W}$; D, R, γ

1: $\mathcal{A} \leftarrow \{i \in \mathcal{O} : x_i = y_1\}$ ▷ entry candidates, Eq. (3)
2: **if** $\mathcal{A} = \emptyset$ **then return** $\{\mathcal{B}_{\ell,h} \cup \mathcal{W}\}_{\ell,h}$ ▷ identical to HM
3: **end if**
4: compute $\pi(j)$ for all $j \in \mathcal{O}$ ▷ reverse cumulative minimum; shared by all heads
5: **for** each layer ℓ and KV head h **do**
6: $e_a \leftarrow \max_{j:\pi(j)=a} s_{\ell,h,j}$ ▷ score routing, Eq. (4)
7: $p_i \leftarrow \max_{0\le r\le R,\ i-r\in\mathcal{O}} \gamma^r e_{i-r}$ ▷ forward extension, Eq. (5)
8: $\delta_i \leftarrow \max(0, p_i - s_{\ell,h,i})$; $\beta \leftarrow \sum_{i\notin\mathcal{B}_{\ell,h}} \delta_i \,/\, \sum_i \delta_i$ (0 if $\sum_i \delta_i = 0$)
9: $z_i \leftarrow s_{\ell,h,i} + \beta\, \delta_i$ ▷ coverage-aware correction, Eq. (6)
10: $\mathcal{S}_{\ell,h} \leftarrow \mathrm{Top}_{b_{\ell,h}}(z) \cup \mathcal{W}$ ▷ same budget, §3.5
11: **end for**
12: gather the keys and values of every $\mathcal{S}_{\ell,h}$ at their original positions; release all scores
13: **return** $\{\mathcal{S}_{\ell,h}\}_{\ell,h}$ ▷ decoding continues from y_1

The extension of line 7 reads the entry scores e at every offset rather than the partially updated p, so no position more than R after an entry point receives a nonzero proposal score from that entry.

B EXPERIMENTAL DETAILS

B.1 BASELINE CONFIGURATIONS

Table 5 lists the configuration of every method as we ran it. Each baseline keeps its own settings rather than a common window or pooling width, so window and kernel sizes differ.

B.2 INPUTS, PROMPT LENGTHS, AND GENERATION LIMITS

The LongBench (Bai et al., 2024) question-answering tasks include NarrativeQA (Kociský et al., 2017), Qasper (Dasigi et al., 2021), HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2021). Its summarization tasks use GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), and Multi-News (Fabbri et al., 2019); the few-shot tasks include TriviaQA (Joshi et al., 2017) and SAMSum (Gliwa et al., 2019), and repository-level code completion uses RepoBench (Liu et al., 2023a). For LongBench, we use a fixed set of 100 inputs per task. RULER (Hsieh et al., 2024) inputs are generated with the nominal 16K configuration; the actual prompt length depends on the tokenizer. Table 6 reports the observed prompt lengths, includ-

Table 5: **Configurations of the compared methods.** Class names follow our implementations.

Method	Configuration
SnapKV	Observation window of 32 positions; average pooling with kernel size 5.
CriticalKV	`CriticalAdaKVPress` with a SnapKV scorer: observation window of 64 positions, kernel size 5, first-stage ratio 0.5; head-wise budget allocation of AdaKV.
Layer-DefensiveKV	Observation window of 32 positions; kernel size 5; critical-mass parameter 0.9.
ReST-KV	Pooling kernel size 21 on RULER and 5 on LongBench.
Expected Attention	`AdaKVPress(ExpectedAttentionPress(`$\epsilon = 10^{-2}$`))`, i.e., combined with the head-wise budget allocation of AdaKV.
DapQ	Pseudo window of 32 tokens, formed by the first 4 and the last 28 prompt tokens and assigned future positions; query-head scores are averaged within each GQA group; every layer and KV head keeps the same number of entries.
ReadSpan	HM with an observation window of 32 positions, pooling width 5, and the allocation of Appendix A.1; $D = 64$, $R = 32$, $\gamma = 0.95$; coverage-aware correction computed per layer and KV head.

ing the prompt template, and Table 7 the maximum number of generated tokens per task, which is the same for both models and all methods; generation may stop earlier at the end-of-sequence token.

Table 6: **Observed prompt lengths** (tokens, including the chat template).

Model	LongBench	RULER
Llama-3.1-8B	826–51,461	11,899–16,372
Qwen3-4B	816–52,558	10,782–22,466

Table 7: **Maximum generated tokens** per task.

Limit	LongBench	RULER
30	–	VT
32	HotpotQA, 2WikiMQA, MuSiQue, TriviaQA, P. Count, P. Re	QA1, QA2
50	–	FWE
64	MultiFieldQA-en, TREC, LCC, RB-P	–
120	–	CWE
128	NarrativeQA, Qasper, SAMSum	S1–S3, K1–K3, MV, MQ
512	GovReport, QMSum, MultiNews	–

B.3 Evaluation Protocol

Metrics. Following LongBench, we use F1 for NarrativeQA, Qasper, MultiFieldQA-en, HotpotQA, 2WikiMQA, MuSiQue, and TriviaQA; ROUGE-L for GovReport, QMSum, MultiNews, and SAMSum; accuracy for TREC, PassageCount, and PassageRetrieval-en; and edit similarity for LCC and RepoBench-P, each with the benchmark's output post-processing. For RULER, a QA output receives full credit if it contains any of the valid answers, and on the other tasks an output receives the fraction of the required answers that it contains, again by string matching.

Aggregation. Scores lie on a 0–100 scale. A task score is the mean over its inputs, a benchmark average weights its tasks equally, and the overall score weights the LongBench and RULER averages equally; it is therefore not the mean over all 2,250 inputs. The component analysis of Section 4.4 uses the same inputs, budgets, and aggregation as the main comparison in Section 4.3.

Scope of the comparison. All results are point estimates from a single greedy run on fixed inputs; apart from the shared first output token, every method generates its own output on its own compressed cache. Both benchmarks were also used during the development of ReadSpan, so we do not present them as held-out confirmation sets. Budgets are nominal retention ratios of the prompt cache; each implementation distributes them with its own allocator, and we do not claim that all implementations occupy exactly identical physical GPU memory at every budget.

B.4 Per-Task Scores across KV Budgets

Table 8 lists the per-task LongBench scores of ReadSpan with Llama-3.1-8B-Instruct from 2% to 20% of the KV cache, whose averages and selected tasks appear in Figure 3; the 5% and 10% columns coincide with the corresponding ReadSpan rows of Table 1 in the main text.

Table 8: **Per-task LongBench scores of ReadSpan across KV budgets** with Llama-3.1-8B-Instruct. Full KV is the uncompressed reference from Table 1, whose abbreviations we reuse.

Task	2%	3%	5%	10%	20%	Full	Task	2%	3%	5%	10%	20%	Full
NrtvQA	24.53	25.89	24.70	24.72	24.59	24.53	MultiNews	15.61	17.49	20.69	22.47	24.28	27.11
Qasper	24.46	31.57	36.61	41.68	42.41	43.69	TREC	75.00	76.00	76.00	76.00	76.00	76.00
MF-en	53.62	55.29	57.71	58.44	56.95	57.06	TriviaQA	88.31	88.57	89.97	90.06	90.56	90.89
HotpotQA	51.94	53.34	53.43	53.73	53.47	54.07	SAMSum	42.30	42.33	42.92	44.61	43.22	43.15
2WikiMQA	39.03	42.01	42.01	42.25	43.16	42.78	P. Count	8.00	8.00	8.00	8.00	8.20	8.20
MuSiQue	32.17	32.02	31.98	32.11	32.82	33.06	P. Re	100.00	100.00	100.00	100.00	100.00	100.00
GovReport	22.30	24.09	25.65	27.48	29.86	33.81	LCC	50.58	57.21	59.48	59.78	59.20	60.54
QMSum	23.77	24.63	24.12	24.71	25.30	25.56	RB-P	54.02	52.87	54.12	54.43	54.83	55.75
							Average	44.10	45.71	46.71	47.53	47.80	48.51

B.5 Per-Task RULER Scores

Table 9 lists the scores of all 13 RULER tasks, whose averages appear in Table 2.

Table 9: **Per-task RULER scores** for all 13 tasks, averaged in Table 2. Task scores are shown with one decimal and the average with two. Within each block, the best score in each column is in **bold** and the second best is underlined (no second best is marked when several methods tie for the best).

	Retrieval (NIAH)								Tracing	Aggregation		QA		
Method	S1	S2	S3	K1	K2	K3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg.
Llama-3.1-8B-Instruct, 5% KV cache														
SnapKV	**100.0**	**100.0**	22.0	**100.0**	**100.0**	8.0	96.0	89.0	98.4	32.4	88.0	76.0	**52.0**	73.98
CriticalKV	**100.0**	98.0	2.0	**100.0**	**100.0**	14.0	91.5	92.5	98.8	17.6	78.0	76.0	**52.0**	70.80
Layer-DefensiveKV	**100.0**	98.0	46.0	**100.0**	**100.0**	42.0	**99.5**	**100.0**	**99.2**	27.2	66.7	**78.0**	**52.0**	77.58
ReST-KV	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	70.0	**99.5**	**100.0**	98.4	17.2	**94.0**	**78.0**	**52.0**	85.32
Expected Attention	92.0	2.0	0.0	4.0	26.0	0.0	4.5	1.5	44.4	2.6	3.3	52.0	28.0	20.03
DapQ	**100.0**	98.0	18.0	98.0	98.0	10.0	84.0	88.5	94.8	31.8	82.7	**78.0**	**52.0**	71.83
ReadSpan	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	97.5	98.5	98.8	**42.2**	86.7	**78.0**	**52.0**	**88.74**
Llama-3.1-8B-Instruct, 10% KV cache														
SnapKV	**100.0**	98.0	72.0	**100.0**	**100.0**	32.0	98.5	97.0	98.8	55.6	**93.3**	76.0	**52.0**	82.56
CriticalKV	**100.0**	**100.0**	12.0	**100.0**	**100.0**	54.0	96.5	97.0	**99.2**	41.0	83.3	76.0	**52.0**	77.77
Layer-DefensiveKV	**100.0**	**100.0**	98.0	**100.0**	**100.0**	98.0	**99.5**	**99.5**	**99.2**	60.4	72.0	**78.0**	**52.0**	88.97
ReST-KV	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	84.0	**99.5**	**99.5**	**99.2**	35.4	**93.3**	**78.0**	**52.0**	87.76
Expected Attention	98.0	20.0	0.0	20.0	94.0	0.0	16.5	12.0	82.8	13.8	39.3	64.0	40.0	38.49
DapQ	**100.0**	**100.0**	66.0	**100.0**	**100.0**	26.0	93.5	95.0	98.8	50.6	86.0	**78.0**	**52.0**	80.45
ReadSpan	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.0	99.0	98.8	**61.6**	87.3	76.0	**52.0**	**90.29**
Qwen3-4B, 5% KV cache														
SnapKV	**100.0**	**100.0**	48.0	**100.0**	70.0	16.0	92.0	99.5	**99.2**	28.6	98.7	60.0	50.0	74.00
CriticalKV	**100.0**	**100.0**	48.0	**100.0**	66.0	14.0	98.5	**100.0**	97.6	16.8	99.3	64.0	**54.0**	73.71
Layer-DefensiveKV	**100.0**	**100.0**	26.0	**100.0**	88.0	50.0	90.0	98.5	93.6	**42.4**	97.3	60.0	50.0	76.60
ReST-KV	**100.0**	**100.0**	98.0	**100.0**	**100.0**	22.0	**99.5**	**100.0**	94.8	17.2	**100.0**	64.0	52.0	80.58
Expected Attention	98.0	48.0	0.0	40.0	0.0	0.0	42.5	45.0	87.6	6.0	46.0	36.0	20.0	36.08
DapQ	**100.0**	**100.0**	12.0	**100.0**	32.0	8.0	53.0	93.5	95.6	12.2	81.3	58.0	42.0	60.59
ReadSpan	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**96.0**	97.5	**100.0**	97.6	31.4	**100.0**	**66.0**	52.0	**87.73**
Qwen3-4B, 10% KV cache														
SnapKV	**100.0**	**100.0**	96.0	**100.0**	96.0	38.0	98.0	**100.0**	**99.2**	49.4	99.3	64.0	52.0	83.99
CriticalKV	**100.0**	**100.0**	86.0	**100.0**	88.0	38.0	**99.5**	**100.0**	98.4	36.4	98.7	64.0	**54.0**	81.77
Layer-DefensiveKV	**100.0**	**100.0**	90.0	**100.0**	**100.0**	92.0	**99.5**	99.5	98.4	**55.6**	98.7	64.0	50.0	88.28
ReST-KV	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	60.0	**99.5**	**100.0**	98.8	39.0	**100.0**	**66.0**	**54.0**	85.95
Expected Attention	**100.0**	86.0	2.0	92.0	46.0	0.0	87.5	91.5	93.2	30.2	89.3	48.0	30.0	61.21
DapQ	**100.0**	98.0	56.0	**100.0**	74.0	18.0	89.5	**100.0**	98.4	25.2	98.0	**66.0**	48.0	74.70
ReadSpan	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**98.0**	98.5	**100.0**	**99.2**	51.4	**100.0**	64.0	52.0	**89.47**