ARC-KV: AMORTIZING ANCHOR SEARCH FOR RECONSTRUCTION-BASED KV CACHE COMPACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context large language model inference is bottlenecked by KV caches that grow linearly with sequence length. This burden is especially severe for long, reusable context prefixes, whose cache must serve many downstream queries. Reconstruction-based methods such as Attention Matching achieve strong downstream task performance with compact KV caches. However, iterative anchor search dominates the compaction cost of OMP-based Attention Matching. This motivates our selective amortization principle of learning a reusable anchor-selection policy across contexts while retaining context-specific reconstruction. In this work, we propose ARC-KV, a novel reconstruction-based KV cache compaction method that follows this principle. To this end, we first train a value-aware indexer to select real-key anchors in a single scoring pass. ARC-KV then applies convex-hull-constrained key merging and fits an attention-mass bias and compact values against the full cache. At inference time, ARC-KV builds the compact cache once per context using the frozen indexer and reuses it for all subsequent queries. Extensive experiments demonstrate that ARC-KV outperforms reported compaction methods in most settings across QuALITY, RULER, and LongBench on Llama-3.1-8B-Instruct. In particular, at 10% KV retention on QuALITY, ARC-KV improves accuracy from 0.6409 to 0.6474 over Attention Matching while reducing compaction time by a factor of 25.73, from 959.8 s to 37.3 s.

1 INTRODUCTION

The rapid evolution of large language models (LLMs) has unlocked remarkable capabilities across a wide array of tasks, including reasoning, coding and agentic workflows (OpenAI, 2026; Yang et al., 2025; Guo et al., 2025). Expanding the context window beyond 1M tokens has further extended these capabilities. For instance, Gemini Pro (Comanici et al., 2025) supports a 10-million-token context window, enabling deep debugging over entire system logs, stack traces, and source code.

However, these gains come at the cost of intensive computation and memory consumption, primarily stemming from the attention mechanism. Since the Key-Value (KV) cache grows significantly as sequence length increases, the auto-regressive decoding stage must repeatedly write and read this large KV cache to and from memory, making the process memory-bound and posing a major challenge to scaling long-context LLMs. Take LLaMA-65B as an example: a 128K-token sequence requires 320GB of memory for the KV cache alone in FP16 (Hooper et al., 2024). Such a memory footprint of the KV cache degrades not only serving throughput but also latency.

Extensive prior work has leveraged attention sparsity to accelerate long-context inference. These approaches fall into three main categories: *rule-based token eviction*, *learned token eviction*, and *cache reconstruction*. *Rule-based token eviction* method discards KV pairs according to token position or importance score, such as StreamingLLM (Xiao et al., 2024), H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024); *Learned eviction* policies (e.g., KVP (Moschella et al., 2026), LKV (Zhou et al., 2026)) train a lightweight selector to retain a subset of the KV cache. However, these policies permanently evict tokens, rendering them inaccessible in all subsequent decoding steps and causing irreversible information loss.

The most recent *cache reconstruction* approach addresses the limitations of direct eviction policies by reconstructing the KV cache into a compact form that preserves rich information. This line of work includes Cartridges (Eyuboglu et al., 2026), Attention Matching (AM) (Zweiger et al., 2026),

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.

and Still (O'Neill et al., 2026). Notably, AM consistently outperforms rule-based token eviction policies even at high compaction ratios (20×–100×). Its OMP-based variant uses orthogonal matching pursuit (OMP) (Pati et al., 1993) to select original keys through iterative search and attention-mass refitting, then fits compact values to match the full cache's attention output. Still instead trains a compactor to synthesize the compact cache in one forward pass.

OMP-based AM makes a coupled design choice by treating both anchor discovery and cache reconstruction as per-context optimization problems. Our key insight is that these operations have different generalization requirements. A selection policy can learn a reusable prior for identifying effective anchors across contexts of the same frozen model. Reconstruction must approximate the attention mass and outputs of the current cache, motivating direct fitting to that context. We therefore amortize the selection policy across contexts while retaining context-specific reconstruction.

We instantiate this decomposition in ARC-KV. First, *Anchor Selection* utilizes a value-aware indexer to condition on reference activations, queries, and cached KV pairs, scoring and selecting anchors in a single pass. Second, *Deterministic Merging* groups keys based on their responses to reference queries and shifts each anchor toward its group's mass-weighted centroid, absorbing non-anchor information while constraining each compact key to its convex hull. Third, *Closed-Form Fitting* leverages context-specific equations to solve for β and C_v using fixed compact keys. Our contributions are summarized as follows:

- We identify anchor search as a component of reconstruction-based KV compaction that can be amortized across contexts through a reusable selection policy while keeping reconstruction context-specific. Based on this decomposition, we introduce ARC-KV, a selectively amortized KV-cache compaction framework for reusable context prefixes.
- We instantiate this framework with a value-aware indexer whose reconstruction-training objective targets held-out post-fitting attention-output fidelity using disjoint scoring, fitting, and holdout queries, without OMP-generated anchor labels. At inference, response-profile key merging and analytical attention-mass and compact-value fitting construct the compact cache from the selected anchors.
- We evaluate the complete ARC-KV pipeline on QuALITY, RULER, and LongBench v1 with Llama-3.1-8B-Instruct, where it ranks first among the reported compacted methods in most evaluated settings. On QuALITY at 10% nominal retention, it reduces reported compaction time by a factor of 25.73 relative to OMP-based AM while increasing accuracy from 0.6409 to 0.6474.

2 Motivation and Analysis

In this section, we empirically analyze cache reconstruction fidelity, iterative anchor search efficiency and value-influence token ranking. We reveal that (1) a hard KV subset cannot retain sufficient information, (2) iterative anchor search dominates the computational cost, and (3) value vectors provide a consistent signal beyond attention-only anchor scores.

2.1 Cache Reconstruction Outperforms Token Eviction in Fidelity

Token eviction selects an index set S and retains the KV pairs (K_S, V_S) unchanged, so a query q attends through $\mathrm{softmax}(qK_S^\top)V_S$; reconstruction-based methods instead use the selected keys as anchors to build compact keys C_k, a mass bias β, and compact values C_v and evaluate $\mathrm{softmax}(qC_k^\top + \beta^\top)C_v$ (Section 3.1; Appendix B.2).

To isolate representation capacity from anchor quality and ensure a fair comparison, we fix the same 5% anchor set across all five cache constructions. As shown in Table 1, the comparison then separately tests mass calibration, key or value merging, and compact value fitting, all while leaving the anchor indices unchanged.

The hard subset yields a relative ℓ_2 error of 0.8778, and fitting β alone leaves the local output error essentially unchanged. Fitting C_v reduces the error to 0.3546, showing that the original anchor values impose a substantial representation constraint in this setting. Merging keys before fitting C_v gives the lowest error of 0.3200. Thus, selected keys should serve as anchors for constructing the compact cache (C_k, β, C_v), rather than be retained unchanged as the final compact cache.

Table 1: Held-out-query reconstruction scores (mean ± SD over 2,560 article–layer–KV-head cells) on QuALITY with Llama-3.1-8B-Instruct at 5% retention. All constructions use the same fixed anchors. The key/value-merging row is diagnostic; ARC-KV merges only keys and fits C_v globally.

Construction	C_k	β	C_v	Rel. $\ell_2 \downarrow$	Cosine $\uparrow$
Hard subset	K_S	0	V_S	0.8778 ± 0.2973	0.7543 ± 0.1055
Mass calibration	K_S	fitted	V_S	0.8782 ± 0.3189	0.7503 ± 0.1100
Key/value merging	C_k^{mrg}	fitted	C_v^{mrg}	0.5109 ± 0.1481	0.8558 ± 0.0738
Value fitting	K_S	fitted	C_v^{LS}	0.3546 ± 0.1265	0.9135 ± 0.0586
Key merging + value fitting	C_k^{mrg}	fitted	C_v^{LS}	$\mathbf{0.3200 \pm 0.1291}$	$\mathbf{0.9250 \pm 0.0566}$

2.2 Iterative Anchor Search Dominates Compaction Time

We profile the official AM implementation (Zweiger et al., 2026), split into reference-query construction, OMP (Pati et al., 1993) anchor search, β fitting, and C_v fitting (Appendix B.3, Figure 5b). At 20% retention, AM's accelerated OMP search (OMP-fast) alone needs 3.70 minutes at 4K tokens and 15.36 hours at 64K, while every other stage takes at most 1.17 minutes at 64K; amortization should therefore target anchor search.

2.3 Value Vectors Matter in Token Selection

Attention-only selectors (HA) (Zhang et al., 2023; Li et al., 2024) rank keys by attention probabilities alone, whereas value-influence ranking (VI) (Guo et al., 2024; Goel et al., 2025) uses each key's normalized leave-one-out effect on the attention output. Under a matched budget and disjoint held-out queries, VI lowers truncated-output error at every keep ratio, by 1.05–6.23% relative to HA on four layer–KV-head pairs (Appendix B.1, Figure 5a).

3 Method

ARC-KV replaces iterative, per-context anchor search with a value-aware indexer that is trained once per base model and then frozen; for each new context it selects real-key anchors in one scoring pass, merges discarded keys into them, and analytically fits an attention-mass bias and compact values. We describe one layer–KV-head pair under grouped-query attention (GQA); the same procedure is applied independently to all pairs.

3.1 Problem Formulation

Let $K, V \in \mathbb{R}^{T \times d}$ denote the key and value cache for a context of length T, with rotary position embedding (RoPE) (Su et al., 2024) applied to the keys. We write k_j and v_j for their j-th rows and $[T] = \{1, \ldots, T\}$ for the cache positions. Throughout the paper, we omit the usual inverse-square-root scaling from attention-softmax logits. For queries $Q \in \mathbb{R}^{n \times d}$, the full-cache attention probabilities and outputs are $P(Q; K) = \mathrm{softmax}(QK^\top)$ and $O(Q; K, V) = P(Q; K)V$. Given a budget $t \ll T$ and keep ratio $\rho = t/T$, we seek an anchor set $S = \{s_r\}_{r=1}^t \subset [T]$ and a compact representation with t entries. It consists of compact keys $C_k = [c_{k,1}; \ldots; c_{k,t}] \in \mathbb{R}^{t \times d}$, an attention-mass bias $\beta \in \mathbb{R}^t$, and compact values $C_v \in \mathbb{R}^{t \times d}$. Each selected key k_{s_r} serves as a real-key anchor that defines a group of cache positions. The corresponding compact key $c_{k,r}$ aggregates key-side information within that group and is constrained to its convex hull, while β and C_v are fitted against the full cache. Thus, unlike hard retention, ARC-KV need not set $C_k = K_S$ or $C_v = V_S$, where K_S and V_S collect the rows of K and V indexed by S. The resulting compact operator is

$$\widehat{O}(Q; C_k, \beta, C_v) = \mathrm{softmax}\big(QC_k^\top + \mathbf{1}_n \beta^\top\big)\, C_v. \tag{1}$$

Preserving only the compacted context's locally normalized attention output is insufficient once it is attended together with uncompressed KV states. For a query q, define the unnormalized attention mass of the original and compact caches as

$$Z_K(q) = \sum_{j=1}^{T} \exp(qk_j^\top), \qquad Z_C(q; \beta) = \sum_{r=1}^{t} \exp(qc_{k,r}^\top + \beta_r). \tag{2}$$

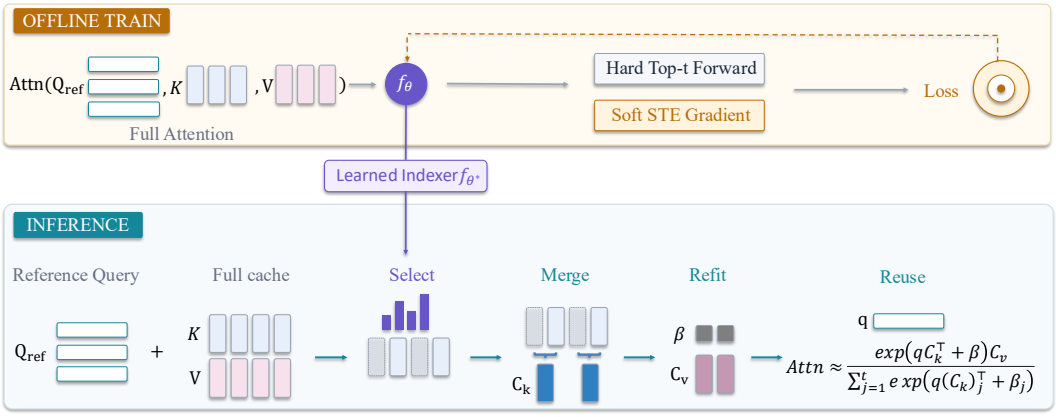


Figure 1: Overview of ARC-KV. Full-attention supervision trains a reusable indexer with hard top-t selection and soft surrogate gradients. At inference, selected anchors undergo context-specific key merging and fitting of β and C_v.

Fitting β so that $Z_C(q;\beta) \approx Z_K(q)$ preserves how strongly the compacted context competes with tokens outside the compacted block, while fitting C_v preserves its attention outputs. Appendix A.7 states the joint objective when the compact block is attended together with uncompressed KV states. Our goal is therefore to train the value-aware indexer to select anchors from which (C_k, β, C_v) can be constructed to preserve the full cache's attention mass and attention outputs for unseen inference queries.

3.2 Method Overview

ARC-KV constructs a compact cache in four context-specific stages. First, a frozen language model produces the full cache (K, V) and a pool of reference activation–query pairs. Second, the trained value-aware indexer scores every cache position, and top-t selection produces the real-key anchor set S. Third, every cached key is assigned to the anchor with the most similar normalized attention-response profile, and each group is merged into one compact key. Fourth, with the merged C_k fixed, analytical fitting determines the attention-mass bias β and compact values C_v.

Only the indexer parameters are learned (Section 3.5); the same frozen indexer is reused across contexts, while reference-query construction, scoring, merging, and fitting are performed for each context (Figure 1).

Reference-query pool. Following KVzip (Kim et al., 2026) and Attention Matching (Zweiger et al., 2026), reference queries come from repeat-prefill: after the initial prefill the model is asked to repeat the context, and we retain each repeated position's pre-query-projection activation x_i and post-RoPE query q_i, forming $\mathcal{D}_{\text{ref}} = \{(x_i, q_i)\}_{i=1}^{n_{\text{ref}}}$ with row-stacked X_{ref} and Q_{ref} (Appendix A.5). All compacted keys and values come from the original-context prefill.

3.3 Value-Aware Indexer

The value-aware indexer is a lightweight scoring model whose role is to replace iterative anchor search with one scoring pass. For each reference pair (x_i, q_i) and full cache (K, V), it produces one logit for every cache position, $I_\theta(x_i, q_i, K, V) = [I_{i,1}, \ldots, I_{i,T}] \in \mathbb{R}^T$. Under GQA, each KV head is shared by multiple query heads. When scoring a KV head, we use the reference queries from all query heads mapped to it.

We extend the low-dimensional query–key scorer of the DeepSeek Sparse Attention indexer (Liu et al., 2025) with one auxiliary value-attention head and represent each position by both k_j and v_j, so keys with similar attention patterns can score differently when their values differ (architecture in Appendix A.1).

To obtain one context-level ranking, we apply the indexer to n_{score} reference pairs. For each row i, we normalize the indexer logits $I_{i,:}$ across the T cache positions. We then aggregate each position across rows by root-mean-square (Rehg, 2024; Zweiger et al., 2026):

$$\pi_{i,j} = \frac{\exp(I_{i,j})}{\sum_{u=1}^{T} \exp(I_{i,u})}, \qquad s_j = \left(\frac{1}{n_{\text{score}}} \sum_{i=1}^{n_{\text{score}}} \pi_{i,j}^2 \right)^{1/2}, \tag{3}$$

Root-mean-square pooling gives additional weight to large query-specific probabilities, retaining positions that are important to only a subset of reference queries. We select the t positions with the largest aggregate scores as real-key anchors, $S = \text{TopK}_{j \in [T]}(s_j, t)$. Unlike query-conditioned sparse attention, ARC-KV selects this set once per context and layer–KV-head pair and reuses it for all future queries.

3.4 Convex-Hull-Constrained Key Merging

Selecting anchors alone discards the key-side geometry of the remaining $T - t$ positions. We recover part of this information with a deterministic post-selection merge that groups keys by their normalized attention-response profiles. Unlike distance in the raw key space, profile similarity directly compares how keys respond to the same reference queries. A schematic is given in Appendix A.8 (Figure 4).

We use all reference queries $Q_{\text{merge}} = Q_{\text{ref}} = [q_i]_{i=1}^{n_{\text{ref}}}$ and define each normalized attention-response profile u_j from the post-RoPE attention logits:

$$\ell_{i,j} = q_i^\top k_j, \qquad z_{\text{merge}} = \max_{i,j} \ell_{i,j}, \qquad u_j = \frac{[\exp(\ell_{i,j} - z_{\text{merge}})]_{i=1}^{n_{\text{ref}}}}{\|[\exp(\ell_{i,j} - z_{\text{merge}})]_{i=1}^{n_{\text{ref}}}\|_2}. \tag{4}$$

Each key is hard-assigned to the anchor with the most similar normalized attention-response profile:

$$\kappa(s_r) = r, \qquad \kappa(j) = \arg\max_{r \in [t]} u_j^\top u_{s_r} \quad (j \notin S), \qquad G_r = \{j \in [T] : \kappa(j) = r\}. \tag{5}$$

Every group therefore contains its own anchor and is nonempty.

Within each group, we define the key-merge weights from query-averaged unnormalized attention mass:

$$\phi_j = \frac{1}{n_{\text{ref}}} \sum_{i=1}^{n_{\text{ref}}} \exp(\ell_{i,j} - z_{\text{merge}}), \qquad \mu_r = \frac{\sum_{j \in G_r} \phi_j k_j}{\sum_{j \in G_r} \phi_j}, \tag{6}$$

and move the anchor toward this weighted centroid:

$$c_{k,r} = (1 - \lambda_m) k_{s_r} + \lambda_m \mu_r, \qquad 0 \le \lambda_m \le 1. \tag{7}$$

Here, λ_m is the key-merging coefficient; $\lambda_m = 0$ leaves $C_k = K_S$, and $\lambda_m = 1$ replaces each anchor by its group centroid. Because $s_r \in G_r$, $c_{k,r} \in \text{conv}\{k_j : j \in G_r\}$, so for any query q its key-derived logit lies between the minimum and maximum real-key logits in that group. The convex-hull constraint therefore keeps each compact key within the geometry of its assigned real keys rather than extrapolating beyond them. The merge operates only on Q_{merge} and K: values influence anchor selection through the value-aware indexer and enter the subsequent compact value fitting, but are not directly averaged within G_r.

Attention mass and compact value fitting. Given the merged C_k and calibration queries Q_{fit} drawn from $\mathcal{D}_{\text{ref}}$, we apply the closed-form calibration of Attention Matching (Zweiger et al., 2026): a projected least-squares solve finds nonnegative $w^\star$ with $\exp(Q_{\text{fit}} C_k^\top) w^\star \approx \exp(Q_{\text{fit}} K^\top) \mathbf{1}_T$ and sets $\beta^\star = \log w^\star$; a ridge regression then fits $C_v^\star$ (Appendix A.9). Neither fit adds learned parameters.

3.5 Training the Reusable Value-Aware Indexer

We train one indexer per frozen base model using the model's full-attention behavior as self-supervision, without anchor labels generated by iterative search. Joint training optimizes reconstruction of held-out attention outputs through a hard-selection forward branch and a differentiable soft surrogate for backpropagation. Both branches operate on real keys before key merging, so the learned objective targets anchor quality directly.

Reference-query episodes. For each training context, we sample three mutually disjoint subsets $\mathcal{D}_{\text{score}}$, $\mathcal{D}_{\text{fit}}$, and $\mathcal{D}_{\text{holdout}}$ from $\mathcal{D}_{\text{ref}}$ without replacement. The scoring subset produces the context-level anchor ranking. The fitting subset supplies Q_{fit} for attention mass and compact value fitting, while the holdout subset supplies Q_{holdout} for evaluating the reconstructed attention outputs. This separation evaluates each selected and fitted cache on queries not used for either operation. The three subsets are resampled deterministically at each epoch.

Reconstruction of held-out attention outputs. For a query q, let $p(q) = \text{softmax}(qK^\top)$ and $y(q) = p(q)V$ denote the full-cache attention distribution and output. The hard branch evaluates the discrete anchors used before inference-time key merging. For a training cache of length T, we set $t = \max(1, \lceil \rho T \rceil)$, aggregate scores over $\mathcal{D}_{\text{score}}$, and select $S = \text{TopK}(s, t)$. We set $C_k^{\text{h}} = K_S$, then obtain β^{h} from Appendix Equation 20 and C_v^{h} from Appendix Equation 21 using Q_{fit}. The reconstructed hard-branch output is $\widehat{y}^{\text{h}}(q) = \widehat{O}(q; C_k^{\text{h}}, \beta^{\text{h}}, C_v^{\text{h}})$. With $n_{\text{holdout}} = |\mathcal{D}_{\text{holdout}}|$, its mean squared relative error is

$$\mathcal{L}_{\text{out}}^{\text{h}} = \frac{1}{n_{\text{holdout}}} \sum_{i=1}^{n_{\text{holdout}}} \frac{\left\| \widehat{y}^{\text{h}}(q_i) - y(q_i) \right\|_2^2}{\|y(q_i)\|_2^2}. \tag{8}$$

The complete hard branch is evaluated without gradient tracking. It determines the forward loss value but contributes no gradient to the indexer.

Because top-t selection is non-differentiable, the backward branch relaxes it over $\mathcal{C} = \text{TopK}(s, \min\{T, 2t\})$: each candidate receives a sigmoid gate g_j on its score margin to the top-t threshold (temperature annealed from 1.0 to 0.1), $\log g$ is added to the attention biases, C_v^{s} comes from a differentiable ridge solve on Q_{fit}, and $\mathcal{L}_{\text{out}}^{\text{s}}$ is evaluated on Q_{holdout} as in Equation 8 (Appendix A.10).

We combine the hard forward value with the gradient of the soft surrogate through

$$\mathcal{L}_{\text{out}}^{\text{STE}} = \text{sg}\big(\mathcal{L}_{\text{out}}^{\text{h}}\big) + \mathcal{L}_{\text{out}}^{\text{s}} - \text{sg}\big(\mathcal{L}_{\text{out}}^{\text{s}}\big)\,. \tag{9}$$

Its forward value is $\mathcal{L}_{\text{out}}^{\text{h}}$, while its gradient comes from $\mathcal{L}_{\text{out}}^{\text{s}}$ through the soft gates and differentiable compact-value fit.

KL warm-up and joint objective. Before joint training, we warm up the indexer by matching its query-specific key distribution to full attention on $\mathcal{D}_{\text{ref}}$, minimizing $\mathcal{L}_{\text{KL}}$, the mean over $(x, q) \in \mathcal{D}_{\text{ref}}$ of $\text{KL}(p(q) \,\|\, \text{softmax}(I_\theta(x, q, K, V)))$ (Appendix Equation 25). The subsequent joint objective is

$$\mathcal{L} = \lambda_{\text{out}} \mathcal{L}_{\text{out}}^{\text{STE}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}. \tag{10}$$

The KL term provides dense query-specific attention supervision, while the reconstruction term supplies the value-aware signal by evaluating outputs $y(q) = p(q)V$ after analytical fitting. The stage-specific weights λ_{out} and λ_{KL} are given in Section 4.1 and Appendix C.2. Both objectives use full-attention behavior rather than anchors generated by OMP. The base language model remains frozen, and only the indexer parameters θ are updated.

Inference. The pool is not partitioned at inference: the same reference pairs serve anchor scoring, key merging, and fitting. The nominal budget is distributed over layer–KV-head pairs with fixed model-specific weights (Appendix A.2, Equation 16), and long contexts select anchors within chunks (Appendix A.3); (C_k, β, C_v) is built once per context and reused for all future queries. Our current scope is one-shot compaction of a reusable context prefix.

4 Experiments and Results

4.1 Experimental Setup

Model and implementation. We used Llama-3.1-8B-Instruct (Grattafiori et al., 2024) for the main evaluation and additionally assessed cross-model generalization with Qwen3-8B (Yang et al., 2025) and Gemma-3-12B-IT (Team et al., 2025) on QuALITY and RULER. Because Gemma-3-12B-IT

uses hybrid attention, we compacted only its eight full-attention layers (Appendix C). Its 40 sliding-window layers remain unchanged, so the retention ratio applies only to the full-attention portion of its cache. For the main evaluations, models were run in FP16 on an NVIDIA B200 GPU with frozen base-model weights; only the value-aware indexer was trained.

Indexer training. For each model we trained the indexers on 600 context-only articles (plus 60 for development) from 11 public long-context datasets, with a 10-epoch KL-only warm-up and five joint epochs of Equation 10 with $(\lambda_{\text{out}}, \lambda_{\text{KL}}) = (2, 1)$. Supervision was generated through repeat-prefill without QA labels, and the trained indexers were frozen for evaluation. The indexers add 0.57–2.13% of the backbone parameters (Table 3) and train once per model in 34–151 B200 GPU-hours (Appendix C.2).

Benchmarks and metrics. We evaluated long-context performance on QuALITY, RULER, and LongBench v1. For QuALITY (Pang et al., 2022), we used 50 articles and all 894 associated questions, and reported multiple-choice accuracy. For RULER (Hsieh et al., 2024), all methods were evaluated on the same 550 examples at a 4K-token context length using the aggregate official string-match score. For LongBench v1 (Bai et al., 2024), we evaluated all 21 tasks using 50 test instances per task (1,050 instances in total) and the official task-specific metrics.

Baselines and cache budget. We compared ARC-KV with SnapKV (Li et al., 2024), Expected Attention (EA) (Devoto et al., 2025), KVzip (Kim et al., 2026), and the default OMP-based Attention Matching (AM) (Zweiger et al., 2026), plus full-context and no-context references (Appendix C.3). With retention ratio $\rho = t/T$, compacted methods were evaluated at $\rho \in \{0.20, 0.10, 0.05, 0.02, 0.01\}$; ARC-KV ratios are nominal under the per-head allocation (Appendix A.2). For the Llama main evaluation, we used $\lambda_m = 0.25$ on QuALITY and RULER and 0.75 on LongBench v1.

4.2 Main Results

ARC-KV ranked first among the reported compacted methods in 13 of the 15 benchmark–retention settings (Figure 2): all five on QuALITY and four of five on both RULER and LongBench. The two exceptions were RULER at $\rho = 0.20$, where ARC-KV essentially matched AM (0.9515 versus 0.9527), and LongBench at $\rho = 0.01$, where it trailed AM-HA (0.2355 versus 0.2451).

ARC-KV improved over the strongest compacted baseline at each retention ratio by 0.23–3.63 percentage points on QuALITY, by 3.34–25.94 points on RULER from $\rho = 0.01$ to 0.10 (e.g., 0.7394 versus 0.4800 for AM at $\rho = 0.05$), and by 1.76–5.51 points on LongBench from $\rho = 0.02$ to 0.20. Relative to full context, ARC-KV reached 0.6437–0.6567 on QuALITY from $\rho = 0.05$ to 0.20 versus 0.6387, retained 98.6% of the RULER score at $\rho = 0.20$, and scored 0.4326 and 0.4481 on LongBench at $\rho = 0.10$ and 0.20 versus 0.4243. Because the OMP-based AM variant was impractically slow over all 1,050 LongBench instances, we used its highest-attention variant, AM-HA, for this benchmark; Appendix D.1 gives the per-benchmark discussion.

4.3 Cross-Model Results

We further evaluated ARC-KV with Qwen3-8B and Gemma-3-12B-IT at a fixed retention ratio of $\rho = 0.05$. As shown in Table 2, ARC-KV ranked first among the compacted methods in three of the four benchmark–model settings. On QuALITY, ARC-KV achieved 0.5034 with Qwen3-8B and surpassed AM by 1.01 percentage points. It achieved 0.6846 with Gemma-3-12B-IT and surpassed the strongest baseline AM by 0.34 points. On RULER with Gemma-3-12B-IT, ARC-KV reached 0.3954 and outperformed KVzip by 16.81 points. With Qwen3-8B, ARC-KV instead ranked second on RULER with 0.3815, trailing KVzip (0.4040) by 2.25 points. Thus, ARC-KV remained competitive across both evaluated architectures, although it did not dominate every benchmark–model setting.

4.4 Efficiency and Ablations

Figure 3 compares QuALITY accuracy with the reported per-sample cache-compaction time at three representative retention ratios. ARC-KV was faster and more accurate than reconstruction-based

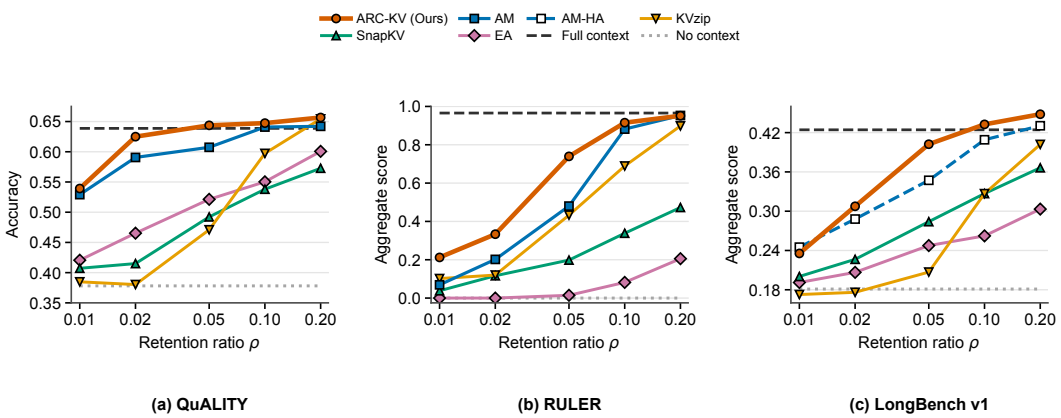


Figure 2: Llama-3.1-8B-Instruct results (higher is better): **(a)** QuALITY accuracy on 894 questions from 50 articles, **(b)** RULER aggregate string-match score on 550 examples at a 4K-token context length, and **(c)** LongBench v1 aggregate score over 21 tasks with 50 instances each. AM uses OMP in **(a,b)** and highest-attention selection (AM-HA) in **(c)**.

Table 2: Cross-model results at $\rho = 0.05$ (higher is better). AM denotes OMP-based Attention Matching; bold marks the best compacted method in each row.

	References		Compacted methods				
Benchmark@Model	Full ctx.	No ctx.	AM	EA	SnapKV	KVzip	**ARC-KV (Ours)**
QuALITY@Qwen3-8B	0.5537	0.2383	0.4933	0.2830	0.2562	0.2125	**0.5034**
RULER@Qwen3-8B	0.9764	0.0000	0.2527	0.0109	0.0818	**0.4040**	0.3815
QuALITY@Gemma-3-12B-IT	0.7114	0.4944	0.6812	0.5962	0.5403	0.4832	**0.6846**
RULER@Gemma-3-12B-IT	0.9727	0.0000	0.1982	0.0509	0.1018	0.2273	**0.3954**

AM at every evaluated budget. At 1% retention, ARC-KV reduced compaction time from 36.34 to 28.0 s and improved accuracy by 1.01 percentage points. At 5% retention, it reduced time from 269.46 to 31.8 s and improved accuracy by 3.63 points. At 10% retention, it reduced time from 959.83 to 37.3 s and improved accuracy by 0.65 points. These reductions corresponded to speedups of 1.30×, 8.47× and 25.73×. The lightweight selectors SnapKV, KVzip and EA remained faster in absolute compaction time. However, the strongest of these alternatives at each budget was 5.01–12.24 points less accurate than ARC-KV. These measurements cover cache compaction only and do not represent end-to-end inference latency.

Compaction time breakdown. A whole-model profile of all 256 layer–KV-head pairs on synthetic 4K–64K contexts at $\rho = 0.05$ (Appendix D.3, Figure 6) shows OMP key selection growing from 14.932 s to 29.49 min while the learned indexer needs 0.516–7.273 s (28.9–243.3×); totals are 2.35 s and 1.66 min versus 16.47 s and 30.91 min for AM, and at 64K repeat prefill is 70.8 s of ARC-KV's 99.8 s. At long context lengths, reference-query construction therefore becomes the dominant remaining cost.

Decode-time attention. At $\rho = 0.20$, compact attention with the fitted bias was 3.19–4.43× faster per layer than full-cache FlashAttention-2 at 4K–32K contexts while storing about 5× fewer K/V states (Appendix D.4, Table 4); this isolates attention, not end-to-end decode latency.

Ablations. On QuALITY with Llama-3.1-8B-Instruct (details in Appendix E), root-mean-square aggregation beats mean and max pooling by 4.59–8.95 points at $\rho \leq 0.05$, and replacing the per-head budget with a uniform one costs 1.68–16.22 points. Every nonzero λ_m lowers the attention-mass error of unmerged anchors (by 43.9–56.9% at the best setting, with a broad optimum; Table 5).

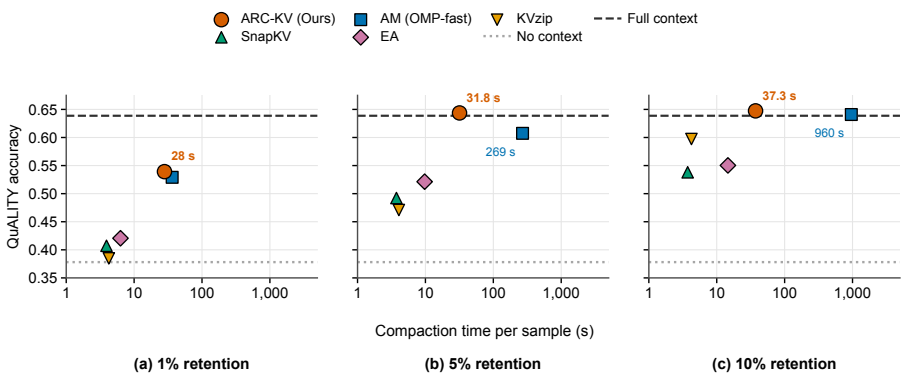


Figure 3: QuALITY accuracy versus compaction time for Llama-3.1-8B-Instruct at **(a)** 1%, **(b)** 5%, and **(c)** 10% retention. The horizontal axis reports per-sample compaction time in seconds on a logarithmic scale. Times include reported query-generation costs and ARC-KV's reference-query construction; horizontal lines are untimed full- and no-context references.

On the first 30 QuALITY articles, the Llama-3.1-8B-Instruct indexer and OMP-fast show limited anchor-set overlap, with mean Jaccard scores of 0.144–0.201 across 1–20% retention under matched per-head budgets (Appendix D.5).

5 Related Work

Token-subset eviction, whether training-free (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024; Kim et al., 2026; Devoto et al., 2025) or learned (Zeng et al., 2024; Moschella et al., 2026; Zhou et al., 2026), deploys a hard subset (K_S, V_S) without value refitting or attention-mass correction, and compensated-retention methods (Chari et al., 2025; Yang et al., 2026) add model components instead. Attention-Gate learns head-specific gates, KVP learns eviction rankings from future-attention rewards, and LKV jointly learns head budgets and token masks through self-distillation. KV-Distill adapts context encoding so retained states aggregate preceding information, whereas IndexMem stores evicted information in an online latent memory. ARC-KV leaves the base context encoding unchanged and fits β and C_v for each context, placing the compensation in the compact cache. Appendix F discusses both lines in detail.

Cache merging, reconstruction, and synthesis. Cartridges (Eyuboglu et al., 2026) learns a new compact KV cache for each context, which is expressive but requires costly gradient-based optimization per context. Attention Matching (Zweiger et al., 2026) restricts C_k to original keys and fits attention-mass weights and C_v in closed form, but its OMP search with repeated NNLS refitting dominates the cost. Still (O'Neill et al., 2026) trains a per-layer Perceiver to synthesize a compact cache in one forward pass, an end-to-end compactor that must be trained separately for each base-model checkpoint. ARC-KV instead learns only one scalar score per key, which keeps offline training lightweight, and constructs C_k, β, and C_v for each context (Appendix F). ARC-KV occupies a distinct point in this design space: it amortizes discrete anchor discovery, while retaining context-specific reconstruction of keys, attention mass, and values.

6 Conclusion

We presented ARC-KV, which uses a value-aware indexer to amortize anchor selection while retaining context-specific convex-hull-constrained key merging and attention-mass and compact-value fitting. On Llama-3.1-8B-Instruct, ARC-KV led reported compacted methods in most settings and led all three benchmark aggregates at 2% retention. On QuALITY at 10% retention, it improved accuracy from 0.6409 to 0.6474 over OMP-based Attention Matching while reducing reported compaction time from 959.8 to 37.3 s. ARC-KV currently targets one-shot compaction of static reusable context prefixes. Extending it to generated-state recompression and reducing reference-query construction and dense key-assignment costs remain future work.

AI USE STATEMENT

We used large language model tools to polish and organize the manuscript, identify and collect related literature, edit code, and revise figures and charts. The authors take full responsibility for the content of this paper, including all AI-assisted text, claims, and references.

REFERENCES

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

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)*, pp. 3119–3137, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Vivek Chari and Benjamin Van Durme. Compactor: Calibrated query-agnostic kv cache compression with approximate leverage scores. *arXiv preprint arXiv:2507.08143*, 2025.

Vivek Chari, Guanghui Qin, and Benjamin Van Durme. Kv-distill: Nearly lossless learnable context compression for llms. *arXiv preprint arXiv:2503.10337*, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

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-Tur, 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*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, volume 2026, pp. 42642–42687, 2026.

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1102. URL https://aclanthology.org/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. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2026.

Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew Morse, Harper Langston, Mingu Lee, and Chris Lott. Caote: Kv cache selection for llms via attention output error-based token eviction. *arXiv preprint arXiv:2504.14051*, 2025.

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.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian Mcauley. LongCoder: A long-range pre-trained language model for code completion. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12098–12107. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/guo23j.html.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21158–21166, 2024.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

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.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, 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*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112/.

Mandar Joshi, Eunsol Choi, Daniel 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 (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/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. *Advances in Neural Information Processing Systems*, 38:167563–167591, 2026.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 05 2018. ISSN 2307-387X. doi: 10.1162/tacl_a_00023. URL https://doi.org/10.1162/tacl_a_00023.

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.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d191ba4c8923ed8fd8935b7c98658b5f-Paper-Conference.pdf.

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.

Luca Moschella, Laura Manduchi, and Ozan Sener. Learning to evict from key-value cache. *arXiv preprint arXiv:2602.10238*, 2026.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized kv cache compaction in a single forward pass. *arXiv preprint arXiv:2606.07878*, 2026.

OpenAI. GPT-5.6 system card. Technical report, OpenAI, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6/gpt-5-6.pdf. Accessed: 2026-09-25.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5336–5358, 2022.

Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pp. 40–44. IEEE, 1993.

Isaac Rehg. Kv-compress: Paged kv-cache compression with variable compression rates per attention head. *arXiv preprint arXiv:2410.00161*, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. Indexmem: Learned kv-cache eviction with latent memory for long-context llm inference. *arXiv preprint arXiv:2605.25475*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Zihao Zeng, Bokai Lin, Tianqi Hou, Hao Zhang, and Zhijie Deng. In-context kv-cache eviction for llms via attention-gate. *arXiv preprint arXiv:2410.12876*, 2024.

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.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. QMSum: A new benchmark for query-based multi-domain meeting summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, 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*, pp. 5905–5921, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.472. URL https://aclanthology.org/2021.naacl-main.472/.

Enshuai Zhou, Yifan Hao, Chao Wang, Rui Zhang, Di Huang, Jiaming Guo, Xing Hu, Zidong Du, Qi Guo, and Yunji Chen. Lkv: End-to-end learning of head-wise budgets and token selection for llm kv cache eviction. *arXiv preprint arXiv:2605.06676*, 2026.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching. *arXiv preprint arXiv:2602.16284*, 2026.

A Additional Method Details

A.1 Value-Aware Indexer Architecture

For physical KV head h, let $G = H_Q/H_{KV}$ and $\mathcal{G}_h = \{hG, \ldots, (h+1)G-1\}$ denote its associated query heads under grouped-query attention. Each scoring row $i = (r, g)$ pairs a reference position r with a query head $g \in \mathcal{G}_h$, uses its post-RoPE query $q_i \in \mathbb{R}^d$, and shares the frozen-model pre-query-projection activation $x_r \in \mathbb{R}^{d_x}$ across the G query heads at position r. The corresponding post-RoPE keys are $k_j \in \mathbb{R}^d$, whereas values $v_j \in \mathbb{R}^d$ do not receive positional rotation. For every layer–KV-head pair, the indexer maintains an independent parameter set with H_I index heads of width d_I and one auxiliary value-attention head of width d_A:

$$L_q, L_k \in \mathbb{R}^{H_I d_I \times d}, \qquad L_x \in \mathbb{R}^{H_I \times d_x}, \qquad b_x \in \mathbb{R}^{H_I}. \tag{11}$$

The value-aware pathway adds

$$U_q, U_k, U_v \in \mathbb{R}^{d_A \times d}, \qquad L_c \in \mathbb{R}^{H_I d_I \times d_A}, \qquad L_v \in \mathbb{R}^{H_I d_I \times d}. \tag{12}$$

We first obtain a query-conditioned summary of the value cache:

$$A^V_{i,j} = \frac{\exp\big((U_q q_i)^\top (U_k k_j)\big)}{\sum_{u=1}^{T} \exp((U_q q_i)^\top (U_k k_u))}, \qquad c_i = \sum_{j=1}^{T} A^V_{i,j} U_v v_j. \tag{13}$$

This is a single auxiliary attention head shared by all H_I index heads, rather than a separate value-attention operation for each index head. The value context enriches the query embedding, while each cached value enriches its colocated key embedding:

$$\bar{q}_{i,a} = \mathcal{N}((L_q q_i + L_c c_i)_a), \qquad \bar{k}_{j,a} = \mathcal{N}((L_k k_j + L_v v_j)_a), \qquad w_{i,a} = (L_x x_r + b_x)_a. \tag{14}$$

Here $a \in [H_I]$, each projected block has width d_I, and $\mathcal{N}(z) = z/\sqrt{\|z\|_2^2/d_I + 10^{-6}}$ is parameter-free RMS normalization. For reference query i and post-RoPE cached key j, the indexer logit is

$$I_{i,j} = \sum_{a=1}^{H_I} w_{i,a} \, \mathrm{LeakyReLU}_{0.1}\left(\frac{\bar{q}_{i,a}^\top \bar{k}_{j,a}}{\sqrt{d_I}}\right). \tag{15}$$

The projections L_c and L_v are initialized to zero, whereas U_q, U_k, and U_v are initialized with entrywise standard deviation $d^{-1/2}$. Consequently, loading a query–key-only checkpoint initially reproduces its logits exactly, after which optimization can learn value-dependent corrections. The frozen language-model weights are not modified; only the independent indexer parameters for each layer–KV-head pair are learned.

A.2 Inference

At inference, ARC-KV constructs one reusable compact cache for each context. We first prefill the original context to obtain its full cache (K, V) and length T. We then obtain $\mathcal{D}_{\mathrm{ref}}$ through the repeat-prefill procedure in Section 3.2. Unlike training, inference does not partition this pool into scoring, fitting, and holdout subsets. The available pool is reused for anchor scoring, key merging, and attention mass and compact value fitting.

Following prior nonuniform allocation schemes (Liu et al., 2023; Feng et al., 2026), we first distribute the nominal cache budget across the compacted layer–KV-head pairs using fixed model-specific weights. Let $\mathcal{P}$ denote this set of pairs, let $a_{\ell,h}$ be normalized such that $\sum_{(\ell,h)\in\mathcal{P}} a_{\ell,h} = 1$, and let t be the nominal per-pair budget. We assign

$$t_{\ell,h} = \mathrm{clip}_{[1,T]}(\lfloor a_{\ell,h} |\mathcal{P}| t \rfloor). \tag{16}$$

Integer rounding and clipping can make the realized aggregate budget differ slightly from its nominal value.

For each layer–KV-head pair, the trained value-aware indexer evaluates the reference pairs and pools its query-specific scores into one score per cache position. The top-$t_{\ell,h}$ positions form the real-key anchor set S. For long contexts, we instead select anchors within contiguous cache chunks and take

their union as S, as detailed in Appendix A.3. ARC-KV assigns every cached key to an anchor by similarity between their normalized attention-response profiles and merges each group to form C_k. It then fits the attention-mass bias β and compact values C_v. The resulting tuple (C_k, β, C_v) defines the compact cache in Equation 1 and is reused for future queries to the same context.

The value-aware indexer therefore replaces iterative OMP anchor search for each layer–KV-head pair with one scoring pass. Appendix A.4 reports the time and memory complexity of these operations. Our current scope is one-shot compaction of a reusable context prefix.

A.3 KV-Based Chunking for Long Contexts

Let T_a denote the length of the article span to be compacted, excluding any prefix or suffix states retained verbatim. When fixed chunking is enabled and $T_a > 10{,}000$, we partition the score vector into contiguous chunks of configured length L_{chunk} and retain the local top $t_{\ell,h,c} = \text{clip}_{[1,T_c]}(\text{round}(t_{\ell,h}T_c/T_a))$ anchors from each chunk. The union of these anchors defines S; response-profile assignment, convex key merging, and the fits of β and C_v are then performed once against the full article cache rather than independently within each chunk. This rounding rule can make the realized anchor count differ slightly from $t_{\ell,h}$ and from the ceiling rule used by the output-training surrogate, while fixed chunking is bypassed when $T_a \leq 10{,}000$.

A.4 Complexity Analysis

For n scoring queries, the indexer evaluates the auxiliary value attention and low-dimensional index scores in $O(nT(H_I d_I + 2d_A))$ time, followed by one top-budget anchor selection. The merge uses all n queries; constructing response profiles and mass weights, assigning all keys to t anchors, and accumulating the centroids require $O(nT(d+t) + Td)$ time and $O(nT + Tt)$ intermediate memory in the current dense implementation. The learned scorer removes the repeated support selection and coefficient refitting performed across the t rounds of OMP, but reference-query construction, dense key-to-anchor assignment, and the final mass and value solves remain, with assignment potentially dominating at larger budgets.

A.5 Method Overview and Reference-Query Construction

Method summary. ARC-KV replaces iterative, per-context anchor search with a value-aware indexer that selects real-key anchors in one scoring pass. The indexer is trained once per base model using supervision derived from full-attention outputs, after which its frozen parameters are reused across contexts. For each new context, ARC-KV applies the trained indexer, merges discarded keys into the selected anchors, and analytically fits an attention-mass bias and compact values. Throughout this section, we describe one layer–KV-head pair under grouped-query attention (GQA). The same procedure is applied independently to all layer–KV-head pairs.

Learned and context-specific components. Only the indexer parameters are learned. Section 3.5 describes how full-attention supervision trains the indexer to select anchors that preserve held-out attention outputs after analytical fitting. Once trained, the same frozen indexer is reused across contexts, while reference-query construction, scoring, merging, and fitting are performed separately for each context. Figure 1 summarizes the training and cache-construction procedures.

Reference-query pool. Reference queries should probe the cache after the model has observed the complete context. We obtain such queries through the repeat-prefill procedure used by KVzip (Kim et al., 2026) and Attention Matching (Zweiger et al., 2026). An initial prefill produces the full cache (K, V). We then append the instruction "Repeat the previous context verbatim." using the model's chat template and teacher-force the original context as the assistant response. At each repeated-context position, we retain the pre-query-projection activation x_i and its post-RoPE query q_i. These pairs form $\mathcal{D}_{\text{ref}} = \{(x_i, q_i)\}_{i=1}^{n_{\text{ref}}}$, with X_{ref} and Q_{ref} denoting the matrices that row-stack the activations and queries, respectively. The repeated copy supplies only the reference pairs, while all keys and values being compacted come from the original-context prefill. Because these queries revisit the context after the model has observed it in full, they provide context-conditioned probes of the original cache.

Full caption of Figure 1. Overview of ARC-KV training and inference. During offline training, full-attention behavior supervises a reusable value-aware indexer. A hard top-t branch evaluates the selected anchors, while a differentiable soft surrogate supplies gradients to the indexer. At inference, the frozen indexer scores a new cache once to select real-key anchors. ARC-KV then merges the corresponding key groups and fits the attention-mass bias β and compact values C_v to materialize a reusable compact cache. The indexer parameters are shared across contexts, whereas all cache-construction operations are context-specific.

A.6 Value-Aware Indexer: Design Walk-Through

To make anchor selection sensitive to value-side information, we instantiate the indexer as a value-aware extension of the low-dimensional query–key scorer used by the DeepSeek Sparse Attention indexer (Liu et al., 2025). A single auxiliary value-attention head forms a query-conditioned summary of the value cache, which augments the query-side features. The indexer represents each cache position using both its key k_j and value v_j. For each reference query i, it derives weights from x_i to combine the scores produced by its low-dimensional index heads. This design allows two keys with similar attention patterns to receive different scores when they carry different value-side information. The complete architecture and initialization are given in Appendix A.1. Section 3.5 describes the reconstruction-based procedure used to train and reuse this scorer.

Anchor selection. We select the t positions with the largest aggregate scores as real-key anchors:

$$S = \underset{j\in[T]}{\mathrm{TopK}}(s_j, t). \tag{17}$$

We write $K_S \equiv K_{S,:}$ for the selected real keys. These anchors define the subsequent key groups but need not equal the final compact keys C_k. Unlike query-conditioned sparse attention, ARC-KV selects this set once per context and layer–KV-head pair and reuses it for all future queries.

A.7 Joint Preservation Objective

Full-cache attention. For queries $Q \in \mathbb{R}^{n\times d}$, the full-cache attention probabilities and outputs are

$$P(Q;K) = \mathrm{softmax}\big(QK^\top\big), \qquad O(Q;K,V) = P(Q;K)V. \tag{18}$$

Joint objective with an uncompressed block. For an uncompressed KV block $(K_{\mathrm{fixed}}, V_{\mathrm{fixed}})$ of length F, such as retained prompt states or subsequently generated states, let $b = [\beta; \mathbf{0}_F]$ append zero bias to its logits. The joint preservation objective can then be written as

$$\frac{\exp\left(q\begin{bmatrix}K\\ K_{\mathrm{fixed}}\end{bmatrix}^\top\right)\begin{bmatrix}V\\ V_{\mathrm{fixed}}\end{bmatrix}}{\sum_{j=1}^{T+F}\exp\left(q\begin{bmatrix}K\\ K_{\mathrm{fixed}}\end{bmatrix}_j^\top\right)} \approx \frac{\exp\left(q\begin{bmatrix}C_k\\ K_{\mathrm{fixed}}\end{bmatrix}^\top + b^\top\right)\begin{bmatrix}C_v\\ V_{\mathrm{fixed}}\end{bmatrix}}{\sum_{j=1}^{t+F}\exp\left(q\begin{bmatrix}C_k\\ K_{\mathrm{fixed}}\end{bmatrix}_j^\top + b_j\right)}. \tag{19}$$

A.8 Key-Merging Schematic and Remarks

Figure 4 summarizes the grouping and merging procedure.

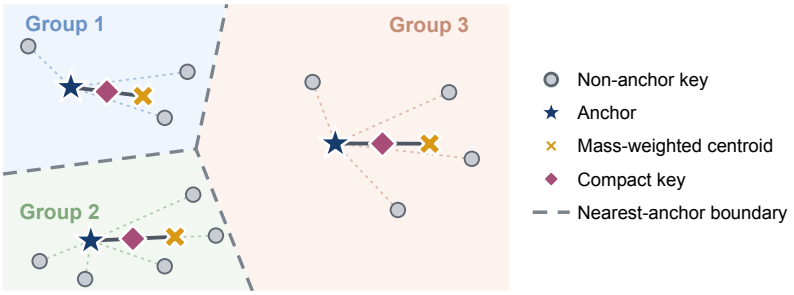


Figure 4: Schematic of convex-hull-constrained key merging. Non-anchor keys are grouped by normalized attention-response similarity, and query-averaged unnormalized attention mass defines the group centroid μ_r. The compact key $c_{k,r}$ interpolates between μ_r and the selected real-key anchor ($\lambda_m = 0.5$ shown). The two-dimensional regions schematically represent assignment in attention-response-profile space rather than Euclidean distance between the plotted keys.

Tie-breaking. Ties for non-anchor keys are resolved by a fixed deterministic rule.

Key-merging coefficient. Here, λ_m is the key-merging coefficient that controls how far each selected anchor moves toward its attention-mass-weighted group centroid. The endpoint $\lambda_m = 0$ leaves the anchor keys unmerged, so $C_k = K_S$, while $\lambda_m = 1$ replaces each anchor by its group centroid.

A.9 Attention Mass and Compact Value Fitting

Given the merged C_k and a calibration-query matrix Q_{fit} drawn from $\mathcal{D}_{\text{ref}}$, we apply the data-dependent calibration introduced by Attention Matching (Zweiger et al., 2026). Let n_{fit} be the number of rows in Q_{fit}, let $z_i^{\text{fit}} = \max_j(q_i^\top k_j)$, and let $z^{\text{fit}} = [z_i^{\text{fit}}]_i$. We define $\Phi_K = \exp(Q_{\text{fit}}K^\top - z^{\text{fit}}\mathbf{1}_T^\top)$ and $\Phi_C = \exp(Q_{\text{fit}}C_k^\top - z^{\text{fit}}\mathbf{1}_t^\top)$. We first approximate the nonnegative attention-mass fit with an unconstrained least-squares solve followed by elementwise projection:

$$\begin{aligned} \widetilde{w} &= \arg\min_{w\in\mathbb{R}^t} \|\Phi_C w - \Phi_K \mathbf{1}_T\|_2^2, \\ w^\star &= \max\{\widetilde{w}, \mathbf{0}_t\}, \\ \beta^\star &= \text{clip}_{[b_{\min}, b_{\max}]}(\log(\max\{w^\star, \epsilon_w \mathbf{1}_t\})). \end{aligned} \tag{20}$$

Here, both maxima are elementwise, $\epsilon_w > 0$ prevents taking the logarithm of zero, and $[b_{\min}, b_{\max}]$ provides a finite numerical guard. With $\widehat{P} = \text{softmax}(Q_{\text{fit}}C_k^\top + \mathbf{1}_{n_{\text{fit}}}(\beta^\star)^\top)$ and $Y = \text{softmax}(Q_{\text{fit}}K^\top)V$, we then fit

$$C_v^\star = \arg\min_{C\in\mathbb{R}^{t\times d}} \left\|\widehat{P}C - Y\right\|_F^2 + \lambda_v \|C\|_F^2. \tag{21}$$

Here, $\lambda_v \geq 0$ is the compact-value ridge coefficient. Together, the two fits approximate the original cache's unnormalized attention mass and attention outputs without adding learned parameters.

A.10 Indexer Training: Additional Derivations

Training overview. The indexer architecture in Section 3.3 is deliberately lightweight. The central learning contribution is the training formulation that turns this scorer into a reusable anchor-selection policy. We train one indexer per frozen base model using the model's full-attention behavior as self-supervision, without anchor labels generated by iterative search. Training begins with a KL warm-up that teaches query-specific token relevance. Joint training then optimizes reconstruction of held-out attention outputs through a hard-selection forward branch and a differentiable soft surrogate for backpropagation. Both branches operate on real keys before key merging, so the learned objective targets anchor quality directly. Deterministic grouping and key merging are applied afterward when constructing the deployed compact cache.

Held-out attention targets. For a query q, define the full-cache attention distribution and output as

$$p(q) = \mathrm{softmax}\big(qK^\top\big), \qquad y(q) = p(q)V. \tag{22}$$

Soft surrogate branch. Because top-t selection is non-differentiable, the backward branch relaxes the decision over the candidate set $\mathcal{C} = \mathrm{TopK}(s, \min\{T, 2t\})$, which contains S. Let sg denote stop-gradient and let $s_{(t)}$ be the hard-selection threshold. For each $j \in \mathcal{C}$, we define

$$\tau = \mathrm{sg}(s_{(t)}), \qquad \sigma_s = \mathrm{sg}(\mathrm{std}(s)), \qquad g_j = \sigma\left(\frac{s_j - \tau}{\gamma_e \sigma_s}\right), \tag{23}$$

where $\sigma(\cdot)$ is the logistic sigmoid. The gate softly relaxes membership near the top-t boundary and is not normalized to enforce $\sum_j g_j = t$. During backpropagation, we treat $\mathcal{C}$, τ, and σ_s as constants, so gradients reach the indexer through the candidate scores in g. The temperature γ_e is annealed linearly from 1.0 to 0.1 over joint training.

The soft branch uses the real candidate keys $C_k^{\mathrm{s}} = K_{\mathcal{C},:}$. For an anchor candidate $j = s_r \in S$, we set its base bias to $b_j^{\mathrm{s}} = \beta_r^{\mathrm{h}}$. Every additional candidate receives $b_j^{\mathrm{s}} = 0$. The effective bias is $b^{\mathrm{s}} + \log(g)$, which gives

$$\widehat{p}^{\mathrm{s}}(q) = \mathrm{softmax}\big(q(C_k^{\mathrm{s}})^\top + (b^{\mathrm{s}} + \log(g))^\top\big). \tag{24}$$

Using the corresponding attention matrix on Q_{fit}, we obtain C_v^{s} with a differentiable ridge solve and set $\widehat{y}^{\mathrm{s}}(q) = \widehat{p}^{\mathrm{s}}(q) C_v^{\mathrm{s}}$. We evaluate $\mathcal{L}_{\mathrm{out}}^{\mathrm{s}}$ on Q_{holdout} using the same squared relative-error form as Equation 8.

KL warm-up and joint objective. Before joint training, we warm up the indexer by matching its query-specific key distribution to full attention on $\mathcal{D}_{\mathrm{ref}}$:

$$\mathcal{L}_{\mathrm{KL}} = \frac{1}{|\mathcal{D}_{\mathrm{ref}}|} \sum_{(x,q)\in\mathcal{D}_{\mathrm{ref}}} \mathrm{KL}(p(q) \,\|\, \mathrm{softmax}(I_\theta(x, q, K, V))). \tag{25}$$

B Additional Motivation Details

B.1 Anchor Selector Definitions

The motivating selector comparison uses disjoint query sets for anchor scoring and held-out evaluation. For n scoring queries, let $p_{ij} = \mathrm{softmax}_j(q_i K^\top)$ and $O_i = \sum_{u=1}^{T} p_{iu} v_u$. Highest-attention ranking pools the attention probabilities, whereas value-influence ranking pools each position's normalized leave-one-out output change:

$$\begin{aligned} D_{ij} &= \left\| O_i - O_i^{(-j)} \right\|_2 = \frac{p_{ij}}{1 - p_{ij}} \left\| v_j - O_i \right\|_2, \\ s_j^{\mathrm{HA}} &= \left(\frac{1}{n} \sum_i p_{ij}^2 \right)^{1/2}, \qquad s_j^{\mathrm{VI}} = \left(\frac{1}{n} \sum_i \frac{D_{ij}^2}{\|O_i\|_2^2 + \epsilon} \right)^{1/2}. \end{aligned} \tag{26}$$

Here, $O_i^{(-j)}$ is the attention output after removing position j and renormalizing over the remaining positions. The top-budget positions under s^{HA} or s^{VI} define the corresponding anchor set.

Evaluation protocol (from Section 2). Traditional KV cache compaction methods rank keys from attention probabilities alone, ignoring value information (Zhang et al., 2023; Li et al., 2024). We represent this family with Highest-Attention ranking (HA), which selects keys with the largest aggregated attention scores, without considering their value vectors. A separate line of work incorporates value-side information when estimating token importance (Guo et al., 2024; Goel et al., 2025); we represent this family with Value-Influence ranking (VI), which selects keys by their normalized leave-one-out effect on the attention output and therefore explicitly accounts for values (Appendix B.1).

To test whether values help select keys, we rank the same cache with HA and VI under a matched budget, then evaluate the selected sets on disjoint held-out queries. For a selected set S, we measure the full-cache output contribution it fails to capture:

$$\widetilde{O}_i(S) = \sum_{j \in S} p_{ij} v_j, \qquad E(S) = \frac{1}{|Q_{\text{eval}}|} \sum_i \frac{\left\| O_i - \widetilde{O}_i(S) \right\|_2}{\|O_i\|_2 + \epsilon}. \tag{27}$$

Here, p_{ij} and O_i are computed from the full cache, so lower $E(S)$ indicates better key selection. Across the four selected layer–KV-head pairs for visual clarity in Figure 5a, VI reduces held-out error at every keep ratio, by 1.05–6.23% relative to HA. This consistent gain shows that values provide useful information beyond attention alone when selecting keys.

B.2 Hard-Subset and Reconstructed Attention Operators

For simplicity, all attention equations omit the standard inverse-square-root scaling of query–key logits.

To compare the fidelity of cache reconstruction under token eviction methods versus cache reconstruction, we track the attention output of each and compare it against the full-cache attention. Token eviction methods (rule-based or learned) construct a compact cache by selecting an index set S and retaining the corresponding KV pairs (K_S, V_S) unchanged, where $K_S = [k_j]_{j \in S}$ and $V_S = [v_j]_{j \in S}$. For a query $q \in \mathbb{R}^d$, these retained states produce

$$\widehat{O}_{\text{hard}}(q; S) = \text{softmax}\big(q K_S^\top\big)\, V_S \tag{28}$$

Reconstruction-based methods, instead use the selected keys as anchors to construct compact keys C_k, an additive mass bias β, and compact values C_v from the full cache. With $C_k, C_v \in \mathbb{R}^{|S| \times d}$ and $\beta \in \mathbb{R}^{|S|}$, the same query is evaluated as

$$\widehat{O}_{\text{rec}}(q; C_k, \beta, C_v) = \text{softmax}\big(q C_k^\top + \beta^\top\big)\, C_v \tag{29}$$

Each row of C_k corresponds to one selected anchor, β calibrates its attention mass, and the corresponding row of C_v supplies its compact value.

B.3 Attention Matching Pipeline Profile

To identify the computational bottleneck of the AM method (Zweiger et al., 2026), we profile the complete AM pipeline using the authors' official open-source implementation. AM compacts a cache in four stages. It first constructs a reference-query set Q_{ref} from the input context. It then applies OMP (Pati et al., 1993) to select an anchor set S from the original keys. With K_S fixed, AM performs a final fit of β to match the full cache's attention mass and then fits C_v to reconstruct its attention outputs. Accordingly, we break compaction time into four parts: reference-query construction (context prefill), iterative anchor search, β fitting, and C_v fitting.

As shown in Figure 5, iterative anchor search dominates the profile and scales much more steeply than the other stages. The OMP-fast search alone requires 3.70 minutes at 4K tokens and 15.36 hours at 64K tokens. At 64K, each remaining profiled stage takes at most 1.17 minutes. This gap arises from dependent selection and mass-refitting steps. The principal opportunity for amortization therefore lies in anchor search, not in the final context-specific reconstruction.

B.4 Motivation Figure

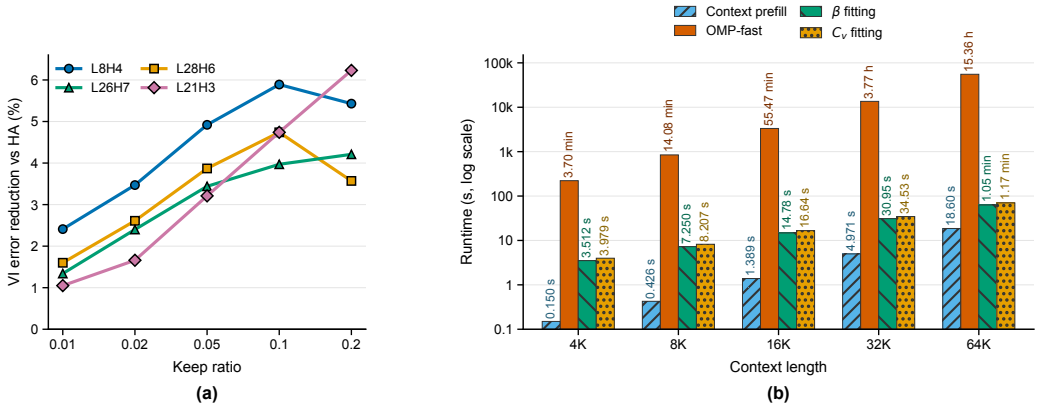


Figure 5: **(a)** Percentage reduction in held-out truncated-output relative error achieved by VI over HA for four selected layer–KV-head pairs on Llama-3.1-8B-Instruct and QuALITY. At keep ratio ρ, for layer ℓ and KV head h, the plotted quantity is $\big(E_{\ell,h}(S_{\mathrm{HA}}(\rho)) - E_{\ell,h}(S_{\mathrm{VI}}(\rho))\big)/E_{\ell,h}(S_{\mathrm{HA}}(\rho))$. Positive values favor VI. LxHy denotes layer x and KV head y. **(b)** Runtime of context prefill, OMP-fast anchor search, nonnegative mass-weight fitting, and compact-value fitting across context lengths at a 20% retention ratio on Llama-3.1-8B-Instruct. Runtime is shown on a logarithmic scale, and every bar is annotated with its measured value.

C Additional Experimental Setup

C.1 Model and Implementation Details

Because Gemma-3-12B-IT uses hybrid attention, we compacted only its eight full-attention layers. Its 40 sliding-window layers were left unchanged because each caches at most 1,024 tokens. The eight full-attention layers therefore account for 44% of the KV cache at 4K tokens and 93% at 64K. All models were run in FP16 on an NVIDIA B200 GPU. The base-model weights remained frozen for ARC-KV, and only the value-aware indexer was trained. At inference time, ARC-KV used the repeat-prefill reference-query construction described in Section 3.2 to build a reusable compact cache for each context.

C.2 Indexer Training Details

Indexer training. Our training sources comprised 11 datasets spanning multihop reasoning (HotpotQA, MuSiQue, and 2WikiMultihopQA) (Yang et al., 2018; Trivedi et al., 2022; Ho et al., 2020), summarization (GovReport, Multi-News, and QMSum) (Huang et al., 2021; Fabbri et al., 2019; Zhong et al., 2021), code (RepoBench-Python v1.1 and LCC-Python) (Liu et al., 2024; Guo et al., 2023), and question answering (TriviaQA, NarrativeQA, and Qasper) (Joshi et al., 2017; Kočiský et al., 2018; Dasigi et al., 2021). From the source training splits, we randomly downsampled each category to 150 training articles and 15 development articles, yielding 600 training and 60 development articles in total. We chose this scale to keep indexer training within about 150 GPU-hours per model. Under the Qwen3-8B tokenizer, these 660 articles contained approximately 4.24M tokens in total, with individual contexts ranging from 1,102 to 16,884 tokens. We retained only the context text and generated supervision through repeat-prefill without QA labels. We used the same two-stage schedule for all three models. For the first 10 epochs, we optimized Equation 10 with $(\lambda_{\mathrm{out}}, \lambda_{\mathrm{KL}}) = (0, 1)$ as a KL-only warm-up. We then trained for five additional epochs with $(\lambda_{\mathrm{out}}, \lambda_{\mathrm{KL}}) = (2, 1)$, corresponding to $2\mathcal{L}_{\mathrm{out}}^{\mathrm{STE}} + \mathcal{L}_{\mathrm{KL}}$. Both stages used AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.01, and global gradient-norm clipping at 1.0. The peak learning rates were 10^{-4} for the KL-only stage and 3×10^{-5} for the joint stage. In each stage, the

learning rate increased linearly over the first 3% of steps and then followed cosine decay to 10% of its peak value.

Table 3: Total trainable parameter count of the value-aware indexers across all compacted layer–KV-head pairs, with $H_I = 8$ and $d_I = d_A = 128$ per pair. The Pairs column reports the number of independently parameterized layer–KV-head indexers, and ratios are relative to each model's text backbone. Gemma includes only the eight full-attention layers used for compaction.

Model	Pairs	Total indexer	Base model	Ratio
Llama-3.1-8B	256	155M	8.03B	1.93%
Qwen3-8B	288	175M	8.19B	2.13%
Gemma-3-12B-IT	64	67M	11.77B	0.57%

Training cost. Training was layer-sharded across eight independent NVIDIA B200 GPU jobs (183 GB each), without inter-GPU communication. Including initial reference-query construction, total costs were 147 GPU-hours for Llama, 151 GPU-hours for Qwen, and 34 GPU-hours for Gemma. Peak per-GPU memory was 35.4, 40.3, and 23.7 GB, respectively. Each indexer was trained once per base model and then frozen for reuse across inference contexts.

C.3 Baselines and Cache Budget

Baselines. We compared ARC-KV with two reference conditions and five KV-cache compaction baselines. *Full context* retained the complete context cache without compaction, whereas *No context* omitted the document context and evaluated the model from the task input alone. SnapKV ranked prompt states using attention from a prompt-final observation window (Li et al., 2024). Attention Matching (AM) used its default OMP-based variant, which greedily selected original keys to reconstruct the full-cache attention mass and then fitted attention-mass weights and compact values (Zweiger et al., 2026). KV Policy (KVP) learned per-head eviction rankings from future-attention rewards (Moschella et al., 2026). KVzip scored cached states through teacher-forced context reconstruction (Kim et al., 2026), whereas Expected Attention estimated their contributions under a model of the future-query distribution (Devoto et al., 2025).

Cache budget. For a layer–KV-head cache with original length T and target compacted length t, we defined the retention (keep) ratio as $\rho = t/T$. We evaluated each compacted method at $\rho \in \{0.20, 0.10, 0.05, 0.02, 0.01\}$. For ARC-KV, we distributed the nominal budget across layer–head pairs using the fixed model-specific allocation described in Section A.2; reported retention ratios are nominal values. Full context corresponded to $\rho = 1$ and was never compacted. Its score is therefore independent of the compacted-cache budget and is repeated across retention-ratio columns only as a common reference.

D Additional Results

D.1 Per-Benchmark Discussion of the Main Results

QuALITY. As shown in Figure 2(a), ARC-KV improved over the strongest compacted baseline at each retention ratio by 0.23–3.63 percentage points. The gains were most pronounced at moderate-to-low budgets: ARC-KV exceeded AM by 3.63 points at $\rho = 0.05$ and by 3.44 points at $\rho = 0.02$. Moreover, from $\rho = 0.05$ to 0.20, ARC-KV achieved accuracies of 0.6437–0.6567, exceeding the full-context reference of 0.6387 while retaining only 5–20% of the cache. Thus, under these moderate retention budgets, compaction did not impose an accuracy penalty and slightly improved the point estimates over full-context inference.

RULER. On RULER (Figure 2(b)), ARC-KV outperformed all compacted baselines from $\rho = 0.01$ to 0.10, with margins of 3.34–25.94 percentage points over the strongest competitor at each budget. At $\rho = 0.05$, for example, ARC-KV reached 0.7394 compared with 0.4800 for AM. At $\rho = 0.20$, ARC-KV approached the full-context score, attaining 0.9515 versus 0.9655 and thereby

retaining 98.6% of full-context performance after removing 80% of the cache. On RULER, ARC-KV therefore combined near-full-context performance at $\rho = 0.20$ with substantial gains over compacted baselines at lower retention ratios.

LongBench. Because the OMP-based AM variant was impractically slow over all 1,050 LongBench instances, we used its highest-attention variant, AM-HA, for this benchmark. As shown in Figure 2(c), ARC-KV achieved the highest aggregate score from $\rho = 0.02$ to 0.20, exceeding the strongest reported compacted baseline by 1.76–5.51 percentage points. At $\rho = 0.01$, ARC-KV scored 0.2355 versus 0.2451 for AM-HA. ARC-KV also exceeded the full-context point estimate at $\rho = 0.10$ and 0.20 (0.4326 and 0.4481 versus 0.4243), while outperforming SnapKV, EA, and KVzip at every evaluated ratio.

Full caption of Figure 2. Main results for Llama-3.1-8B-Instruct across three long-context benchmarks and five KV-cache retention ratios (higher is better). **(a)**, QuALITY multiple-choice accuracy on 894 questions from 50 articles; ARC-KV ranked first among reported compacted methods at every ratio. **(b)**, RULER aggregate string-match score on 550 examples at a 4K-token context length; ARC-KV ranked first at four of five ratios and trailed AM only at $\rho = 0.20$. **(c)**, LongBench v1 aggregate score over 21 tasks with 50 test instances per task; ARC-KV ranked first at four of five ratios and trailed AM-HA only at $\rho = 0.01$. EA denotes Expected Attention. AM denotes the default OMP-based Attention Matching variant in **(a)** and **(b)**, whereas AM-HA denotes its highest-attention variant in **(c)**. ARC-KV uses $\lambda_m = 0.25$ in **(a)** and **(b)** and $\lambda_m = 0.75$ in **(c)**; the full- and no-context lines are budget-independent references.

D.2 Quality–Efficiency Trade-off: Additional Remarks

Compared with OMP-based AM, ARC-KV substantially reduced compaction time. It also achieved higher accuracy than the faster lightweight selectors. However, these measurements cover cache compaction only and do not represent end-to-end inference latency.

Full caption of Figure 3. Accuracy–time trade-off for Llama-3.1-8B-Instruct on QuALITY at **(a)** $\rho = 0.01$, **(b)** $\rho = 0.05$, and **(c)** $\rho = 0.10$. Each point represents a compacted-cache method; the horizontal lines show the full- and no-context accuracy references, which do not perform cache compaction and are therefore not treated as timed scatter points. The horizontal axis uses a logarithmic scale and reports per-sample compaction time in seconds. The baseline totals already include their reported query-generation time, whereas ARC-KV uses the single-deployment total including reference-query construction.

D.3 Compaction Time Breakdown

Figure 6 profiles whole-model cache construction for all 256 layer–KV-head pairs of Llama-3.1-8B-Instruct on synthetic 4K–64K contexts at $\rho = 0.05$. Each context length contains paired stacked bars for ARC-KV and AM, with repeat prefill treated as the query-generation stage for both methods. OMP key selection increased from 14.932 s at 4K to 29.49 min at 64K, whereas the learned indexer required only 0.516–7.273 s over the same range. Replacing iterative search therefore reduced selection time by 28.9× at 4K and 243.3× at 64K.

The component-wise total for ARC-KV was 2.35 s at 4K and 1.66 min at 64K, compared with 16.47 s and 30.91 min for AM, corresponding to 7.01× and 18.59× speedups. At 64K, repeat prefill accounted for 70.8 s of ARC-KV's 99.8 s total, whereas learned key selection required only 7.273 s. Thus, amortizing anchor selection removes the iterative-search bottleneck and shifts the dominant remaining cost to reference-query generation at long context lengths.

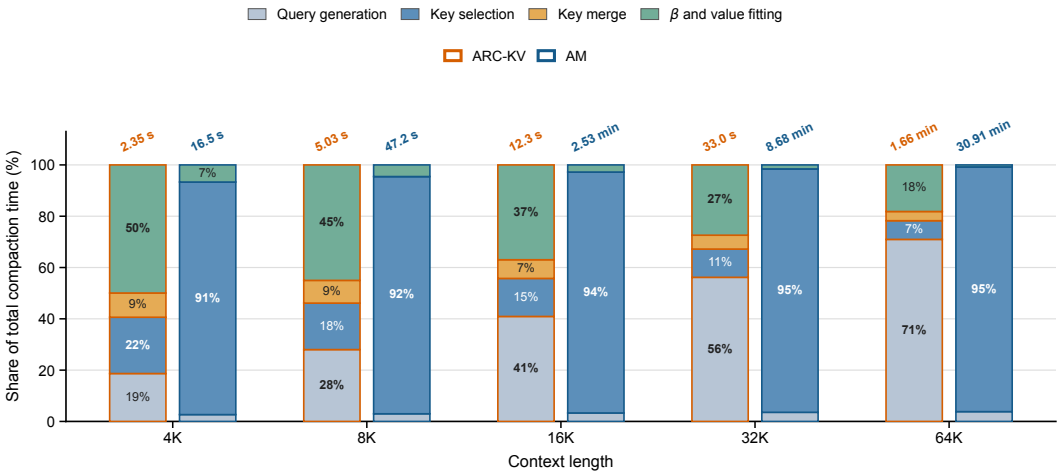


Figure 6: Whole-model cache-compaction time for Llama-3.1-8B-Instruct on synthetic contexts at $\rho = 0.05$. Timings cover all 256 layer–KV-head pairs and were measured in FP16 on an NVIDIA B200 GPU. Each context length contains one stacked bar for ARC-KV and one for AM; fill colors identify stages and outline colors identify methods. Each bar is normalized to 100%, so segment heights show the percentage of total compaction time attributable to each stage; labels above the bars report the corresponding absolute totals. Query generation uses repeat prefill for both methods. For ARC-KV, the remaining segments are learned-indexer top-t selection, mass-weighted key merging, and the sum of guarded attention-mass fitting and merged-key least squares. For AM, they are OMP-fast selection and the sum of the reported baseline NNLS and unmerged least-squares fitting times. OMP-fast uses $k = 4$ and refits every two selection steps. The plotted totals are sums of the measured components rather than independent end-to-end measurements.

D.4 Decode-Time Attention Speedup

KV-cache compaction reduces both memory use and the cost of attending to the cached context during decoding. We measured one decode step with Llama-3.1-8B-Instruct at batch size 64 on an NVIDIA B200 GPU in BF16. The full-cache baseline used FlashAttention-2, whereas ARC-KV retained 20% of the KV states and applied its fitted attention-mass bias β. Measurements used eager execution and were reproducible within approximately 1–3%. Table 4 reports the per-layer latency and the corresponding sum over all 32 attention layers.

Table 4: Decode-time attention efficiency with a full cache and an ARC-KV cache at $\rho = 0.20$. K/V memory and latency are reported per attention layer for a batch of 64. The latency column compares full-cache FlashAttention-2 with compact attention including the fitted bias β. The final column sums the per-layer latency over all 32 attention layers.

Context	K/V per layer (GB) Full → compact	Latency per layer (μs) FA2 → ARC-KV	Speedup	32-layer attention (ms/token)
4K	1.07 → 0.21	174.3 → 54.6	3.19×	5.58 → 1.75
8K	2.15 → 0.43	341.4 → 89.4	3.82×	10.92 → 2.86
16K	4.29 → 0.86	667.9 → 166.3	4.02×	21.37 → 5.32
32K	8.59 → 1.72	1472.5 → 332.2	4.43×	47.12 → 10.63

At $\rho = 0.20$, ARC-KV reduced the stored K/V states by approximately 5×. Per-layer attention latency improved by 3.19× at 4K and 4.43× at 32K. Across 32 layers, the attention component decreased from 5.58 to 1.75 ms per token at 4K and from 47.12 to 10.63 ms at 32K. The full-length bias-aware path was only 1.06–1.09× slower than FlashAttention-2. The measured speedup therefore came primarily from shortening the attended cache rather than omitting the fitted bias. These measurements isolate attention and do not represent end-to-end decode latency. CUDA Graph capture may change the absolute latency but does not change the cache-size reduction.

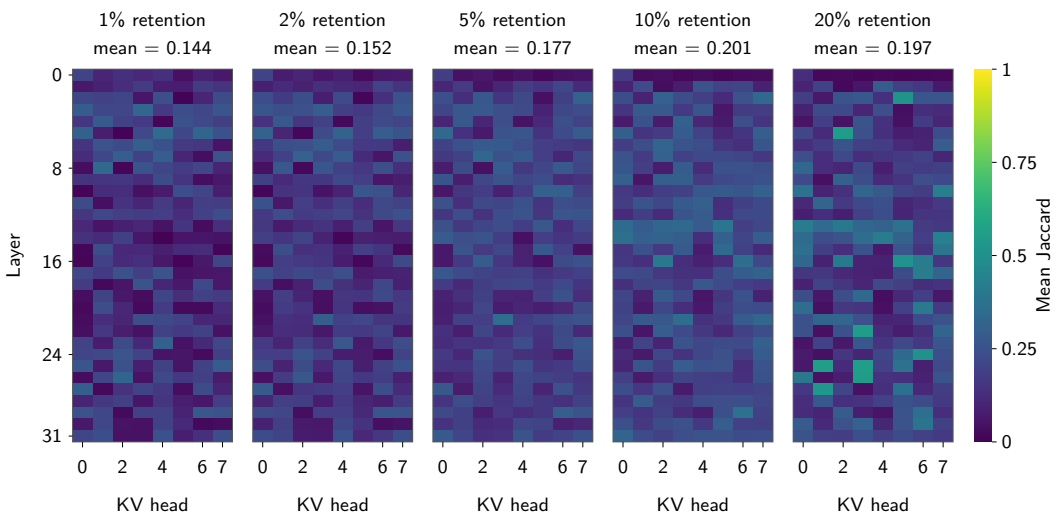


Figure 7: Anchor-set overlap between the Llama-3.1-8B-Instruct indexer and OMP-fast on the first 30 QuALITY development articles under matched uniform per-head budgets. Each cell is the mean Jaccard overlap over the 30 articles for one layer–KV-head pair; layer and head indices are zero-based. The five retention ratios share a color scale from 0 (disjoint sets) to 1 (identical sets). Panel means average all 7,680 article–layer–KV-head observations at that ratio. The comparison concerns selected real-key anchors before key merging and the final cache-reconstruction fits.

D.5 Anchor-Selection Overlap with OMP

We compared the trained Llama-3.1-8B-Instruct indexer with OMP-fast on the first 30 articles of the QuALITY development split, covering all 32 layers and eight KV heads at retention ratios $\rho \in \{0.01, 0.02, 0.05, 0.10, 0.20\}$. This diagnostic uses the same anchor count $k = \lfloor \rho T_a \rfloor$ for both selectors in each layer–KV-head pair, where T_a is the article length; budgets are uniform across pairs, unlike the default nonuniform allocation in Section A.2. The articles contain 2,722–8,094 tokens and each occupies one chunk under the configured chunk size of 8,192. Anchor positions are measured relative to the article, excluding prompt and suffix tokens.

The run used repeat-prefill reference queries with a query budget of 2,048, a query-construction budget of 50,000, and seed 0. The recorded KV states came from BF16 model execution and were cast to FP32 for this diagnostic. The indexer used softmax normalization followed by RMS aggregation. The recorded OMP-fast settings were `k_choice=4`, `nnls_interval=2`, and `nnls_iters=0`.

For each article–layer–KV-head observation, let S_A and S_O be the indexer's and OMP-fast's selected token-index sets before key merging and the final cache-reconstruction fits, with $|S_A| = |S_O| = k$. We measure Jaccard overlap and the fraction of selected anchors shared by the two methods as

$$J = \frac{|S_A \cap S_O|}{|S_A \cup S_O|}, \qquad R = \frac{|S_A \cap S_O|}{k}. \tag{30}$$

Figure 7 averages J over the 30 articles separately for each layer–KV-head pair. Overall means give equal weight to each of the $30 \times 32 \times 8 = 7{,}680$ observations per retention ratio; R is likewise computed per observation before averaging.

The mean Jaccard scores are 0.144, 0.152, 0.177, 0.201, and 0.197 at the five increasing retention ratios. The mean shared-anchor fraction R ranges from 24.1% to 32.6%, indicating that the two selectors often choose different supports at matched cardinality. Overlap varies across layers and heads: at 20% retention, the article-averaged Jaccard scores range from 0.012 to 0.558 across the 256 pairs. This diagnostic characterizes agreement between anchor identities; it does not establish a downstream quality advantage for either selector or isolate the causes of ARC-KV's task-level results.

E Ablation Studies

E.1 Key-Merging Coefficient

We varied the key-merging coefficient λ_m in Equation 7 while holding the remaining ARC-KV components fixed. For each layer–KV-head pair, let $Z_f = \Phi_K \mathbf{1}_T$ and $Z_c = \Phi_C w^\star$ denote the full- and compact-cache unnormalized attention-mass responses in Equation 20. We measured the relative attention-mass error $E_{\text{mass}} = \|Z_c - Z_f\|_2 / \|Z_f\|_2$ on the 50 QuALITY articles with Llama-3.1-8B-Instruct. Because the full-response magnitudes vary by orders of magnitude across layer–KV-head pairs, we normalize each error by $\|Z_f\|_2$ before aggregation rather than report raw ℓ_2 error.

Table 5: Relative attention-mass ℓ_2 error for different key-merging coefficients λ_m on 50 QuALITY articles with Llama-3.1-8B-Instruct (lower is better). Bold indicates the minimum at each retention ratio at the reported precision.

λ_m	$\rho = 0.20$	$\rho = 0.10$	$\rho = 0.05$	$\rho = 0.02$	$\rho = 0.01$
0.0	0.791	1.019	1.279	1.505	1.539
0.2	0.534	0.625	0.745	0.887	0.948
0.3	0.460	0.523	0.632	0.792	0.879
0.4	0.417	0.470	0.584	**0.764**	**0.864**
0.5	0.395	0.446	**0.571**	**0.764**	0.868
0.6	0.386	**0.439**	0.573	0.774	0.877
0.7	**0.384**	0.440	0.582	0.786	0.886
1.0	0.392	0.456	0.613	0.821	0.911

Every nonzero λ_m reduced the relative attention-mass error compared with the unmerged anchors ($\lambda_m = 0$) at every retention ratio. The best setting reduced the error by 43.9–56.9%, with the minimizing coefficient increasing from 0.4 at $\rho = 0.01$ to 0.7 at $\rho = 0.20$. At $\rho = 0.02$, $\lambda_m = 0.4$ and 0.5 were tied at the reported precision. At every ratio, at least one neighboring setting was within 0.004 of the minimum, indicating a broad optimum rather than sensitivity to one exact coefficient. These results show that moving anchors toward their group centroids consistently improves attention-mass reconstruction, while the preferred coefficient increases with the retention ratio.

E.2 Aggregation Method

We compared the root-mean-square aggregation in Equation 3 against mean and max aggregation. It applies a softmax over keys for each reference query before pooling each key across queries. Table 6 reports the change in QuALITY accuracy relative to root-mean-square aggregation. Negative values indicate lower accuracy.

Table 6: Aggregation ablation on QuALITY with Llama-3.1-8B-Instruct. Entries are accuracy differences in percentage points relative to root-mean-square aggregation across KV-cache retention ratios (higher is better). An asterisk marks a statistically significant difference from root-mean-square aggregation.

Aggregation	0.20	0.10	0.05	0.02	0.01
Root-mean-square	0.00	0.00	0.00	0.00	0.00
Mean	-1.12	-3.69^*	-6.60^*	-5.37^*	-4.59^*
Max	-1.45	-2.24	-7.27^*	-8.95^*	-7.05^*

Root-mean-square aggregation achieved the highest accuracy at every retention ratio. At $\rho \leq 0.05$, mean aggregation was worse by 4.59–6.60 points and max aggregation by 7.05–8.95 points. This pattern is consistent with max aggregation being sensitive to a single noisy query-specific peak. Root-mean-square aggregation first normalizes scores across keys for each query. It then pools the

bounded responses across queries. This preserves strong evidence from a subset of queries without allowing one maximum response to determine the ranking.

E.3 Per-head Budget Allocation

We ablated the fixed model-specific allocation across layer–KV-head pairs described in Section A.2 by replacing it with a uniform allocation. Both variants used the same total nominal cache budget and differed only in how that budget was distributed across layer–head pairs. Table 7 reports the change in QuALITY accuracy relative to the default per-head allocation.

Table 7: Per-head budget-allocation ablation on QuALITY with Llama-3.1-8B-Instruct. Entries are accuracy differences in percentage points relative to the default model-specific per-head allocation across KV-cache retention ratios (higher is better). Both variants use the same total nominal cache budget. An asterisk marks a statistically significant difference from the default allocation.

Allocation	0.20	0.10	0.05	0.02	0.01
Per-head (default)	0.00	0.00	0.00	0.00	0.00
Uniform	-1.68	-4.59^*	-10.29^*	-16.22^*	-10.96^*

Uniform allocation reduced QuALITY accuracy at every retention ratio, with losses ranging from 1.68 to 16.22 percentage points. The penalty exceeded 10 points at all ratios below 10% retention and was largest at $\rho = 0.02$.

F Extended Related Work

Token-subset cache eviction. Training-free eviction methods bound cache growth by retaining selected original KV pairs according to positional or observed-attention signals. StreamingLLM (Xiao et al., 2024) preserves initial attention sinks and a recent window, whereas Scissorhands (Liu et al., 2023), H2O (Zhang et al., 2023), and Keyformer (Adnan et al., 2024) estimate importance from different forms of historical attention. SnapKV (Li et al., 2024) uses attention from a prompt-final observation window to rank historical tokens and retain the highest-scoring KV pairs. These inexpensive policies can fail when positional, historical, or visible-query proxies do not predict a later attention shift. Reusable-prefix compression must select a context-level cache before any downstream task query is available. KVzip (Kim et al., 2026) scores KV pairs through teacher-forced context reconstruction, Compactor (Chari & Van Durme, 2025) combines approximate leverage with non-causal context-attention scores, and Expected Attention (Devoto et al., 2025) estimates KV pair contributions under a Gaussian model of future queries. These proxy objectives average future-query utility and can therefore miss rare queries whose required tokens receive low proxy scores. PyramidKV (Cai et al., 2024) assigns a fixed pyramidal budget across layers, while AdaKV (Feng et al., 2026) reallocates each layer's fixed budget across KV heads. These refinements improve resource allocation, but most subset methods still bind each selected key to its original value and provide no post-selection correction for discarded softmax mass or value contributions. Our method improves attention-output fidelity by using selected real keys as anchors, merging each anchor group into a compact key, and fitting β and C_v against the full cache rather than retaining the corresponding original KV pairs unchanged.

Learned and compensated retention. Learned eviction methods replace hand-designed importance rules with model-specific selectors. Attention-Gate (Zeng et al., 2024) learns head-specific gates, KVP (Moschella et al., 2026) learns per-head rankings from future-attention rewards, and LKV (Zhou et al., 2026) jointly learns head budgets and token masks through self-distillation. These methods learn which original KV entries to retain, but their deployed caches remain hard subsets $(K[S], V[S])$ without value refitting or attention-mass correction. Other methods preserve discarded information through additional model components: KV-Distill (Chari et al., 2025) adapts context encoding so retained states aggregate preceding information, whereas IndexMem (Yang et al., 2026) stores evicted information in an online latent memory. Our method instead leaves the base context encoding unchanged and fits β and C_v for each context, placing the compensation directly in the reusable compact cache.

Cache merging, reconstruction, and synthesis (full discussion). Cartridges (Eyuboglu et al., 2026) learns a new compact KV cache for each context by matching the full-context model on synthetic self-study examples. Because its cache entries are freely optimized rather than selected from the original cache, it is expressive but requires costly gradient-based optimization for every new context. Attention Matching (Zweiger et al., 2026) restricts C_k to original keys, fits nonnegative attention-mass weights by NNLS, and fits C_v by least squares. But its OMP process improves mass reconstruction through repeated greedy selection and NNLS refitting, making support search the dominant cost. Still (O'Neill et al., 2026) trains a per-layer Perceiver to synthesize a compact cache in one forward pass, thereby avoiding per-context optimization. This one-pass inference efficiency comes at the cost of an end-to-end compactor that requires non-trivial offline training resources and must be trained separately for each base-model checkpoint. Our method scores the original keys and selects real-key anchors in a single indexer pass, making anchor selection efficient. It then constructs C_k by merging each anchor group and fits β and C_v for each context to preserve fidelity to the full-cache attention output. Because the indexer learns only one scalar score per key rather than synthesizing the compact cache, it requires lightweight offline training.