

SurrogateKV: Representation-Preserving KV Cache Compression for Long-Context LLMs

Junwon Kim Junghyun Ryu Farid Talibli Jungmin So Youngjae Kim*

Department of Computer Science, Sogang University

Seoul, Republic of Korea

{kimjunewon, jhryu, faridt, jsol, youkim}@sogang.ac.kr

Abstract

KV cache compression is essential for long-context LLM inference. Most token-level KV compressors reduce cache length by retaining selected token KV entries and evicting the rest. Once evicted, those spans no longer receive attention, even though some may still carry information useful enough to preserve in compact form. We propose SurrogateKV, a training-free, runtime-only method that re-frames KV compression as region-level representation allocation. Under the same KV-slot budget, SurrogateKV assigns historical regions to raw KV, surrogate KV, or drop. It does not train a compressor, modify model weights, or introduce a new attention operator. After prefill, SurrogateKV constructs lightweight surrogate entries as ordinary KV pairs, keeping their information available during standard attention decoding. This representation decision is orthogonal to layer- and head-wise budget allocation. We evaluate SurrogateKV-Snap, SurrogateKV-Dynamic, and SurrogateKV-Ada, obtained by applying SurrogateKV to SnapKV, DynamicKV, and Ada-KV, respectively. As representative results, SurrogateKV-Snap raises Needle-in-a-Haystack (NIAH) retrieval accuracy from 87.51 to 98.84 at 128 slots per layer. SurrogateKV-Ada raises the LongBench average score from 40.77 to 41.26 at 512 slots per layer while preserving 98.4% of full-cache performance with only 10.5% of the prefill KV cache. Code is available at <https://github.com/kimjunewon/SurrogateKV>.

1 Introduction

Large language models (LLMs) (Brown et al., 2020) are widely used across natural language processing (NLP) tasks, including summarization, translation, code generation, and language understanding. Recent work has extended LLMs to longer contexts (Chen et al., 2023; Peng et al.,

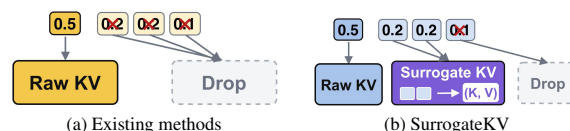


Figure 1: Conventional token eviction vs. SurrogateKV. SurrogateKV adds an intermediate surrogate tier between retaining raw KV entries and dropping them.

2024; Chen et al., 2024), but longer contexts increase inference latency and memory costs. A major source of these costs is the per-request KV cache (Kwon et al., 2023), which occupies GPU memory during autoregressive decoding. In disaggregated serving, the cache is also transferred from prefill to decode workers (Zhong et al., 2024). For example, the KV cache for one 2,048-token OPT-13B request can occupy up to 1.6 GB (Kwon et al., 2023). Reducing KV cache size is essential for lowering GPU memory pressure, prefill-to-decode transfer overhead, and end-to-end latency.

Prior work reduces KV-cache overhead by compressing individual entries or retaining fewer of them. KV quantization compresses key/value states (Hooper et al., 2024); we focus on token-level KV compression, which retains selected historical entries. Among these methods, StreamingLLM (Xiao et al., 2024), H₂O (Zhang et al., 2023), and SnapKV (Li et al., 2024) select retained entries, while PyramidKV (Cai et al., 2025), DynamicKV (Zhou et al., 2025), and Ada-KV (Feng et al., 2025) distribute cache budgets across layers or attention heads.

Despite these advances, eviction-based compressors still retain selected entries and remove the rest from subsequent attention (Figure 1). This changes where attention can flow, and removing important KV states can degrade performance (Zhang et al., 2023). Section 3 quantifies the resulting attention-mass shifts on evicted regions. A SnapKV-based diagnostic further shows greater key and value coherence in contiguous evicted chunks than in matched

*Corresponding author.

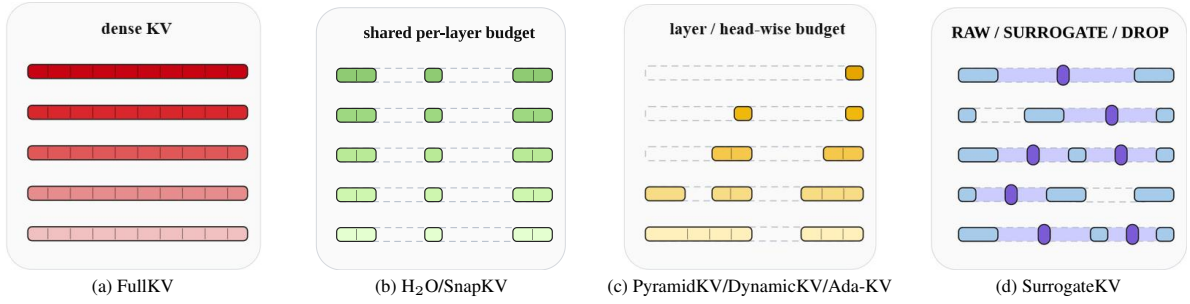


Figure 2: Positioning of SurrogateKV. Token-eviction methods select token- or chunk-level KV entries, while adaptive-budget methods distribute cache slots across layers or attention heads. Given the budget supplied by a base compressor, SurrogateKV forms historical regions and assigns each region to RAW, SURROGATE, or DROP.

random sets. Pure eviction therefore leaves no intermediate representation: some lower-salience regions may not justify retaining all of their raw entries but may still benefit from a compact representation. Merging and compensation methods mitigate this loss by merging similar states around pivots, folding discarded values into retained anchors, or using compensation tokens (Wang et al., 2024; Tang et al., 2025; Yuan et al., 2025).

We therefore propose **SurrogateKV**, a training-free, representation-level KV compression method that assigns each historical region to RAW, SURROGATE, or DROP under a fixed KV-slot budget. Unlike similarity-based merging, SurrogateKV uses recent-query salience rather than similarity grouping or pivot selection to make this allocation: high-salience regions remain RAW, selected lower-salience runs receive one SURROGATE slot, and the rest are dropped. As shown in Figure 2, this representation decision is orthogonal to layer-wise and head-wise budget allocation. In our instantiations, the base compressor supplies the cache budget and allocation granularity, while SurrogateKV determines each region’s representation within that budget. We instantiate SurrogateKV with three representative base compressors: SnapKV’s shared token/chunk selection (Li et al., 2024), DynamicKV’s layer-adaptive allocation (Zhou et al., 2025), and Ada-KV’s head-aware allocation (Feng et al., 2025). We evaluate the resulting variants, denoted SurrogateKV-Snap, SurrogateKV-Dynamic, and SurrogateKV-Ada.

As representative results, SurrogateKV-Snap achieves 98.84 average retrieval accuracy on the Needle-in-a-Haystack diagnostic (Kamradt, 2023) at $B_{KV} = 128$, compared with 87.51 for SnapKV, indicating improved retrieval robustness under aggressive compression. On LongBench (Bai et al., 2024), at $B_{KV} = 512$, SurrogateKV-Ada improves

Ada-KV, the strongest baseline in our evaluation, from 40.77 to 41.26. This LongBench setting uses 10.5% of FullKV’s prefill KV cache on average, preserves 98.4% of its score, and closes 42.5% of the remaining performance gap.

2 Background and Related Work

2.1 Token-Level KV Eviction and Adaptive Budget Allocation

Token-level KV compression reduces the attended cache by retaining only a subset of historical KV states. StreamingLLM retains attention sinks and recent tokens (Xiao et al., 2024), ScissorHands exploits persistent token importance (Liu et al., 2023), H₂O keeps recent and heavy-hitter tokens (Zhang et al., 2023), and SnapKV selects prompt KV positions using recent-window attention (Li et al., 2024). Later studies observe that KV importance is uneven across model components and allocate the limited cache budget non-uniformly: PyramidKV assigns layer-wise budgets (Cai et al., 2025), DynamicKV adapts layer-wise budgets to task-specific attention patterns (Zhou et al., 2025), and Ada-KV allocates budgets across attention heads (Feng et al., 2025).

Although these methods differ in scoring rules and budget granularity, the compressed cache remains a subset of original token-level KV entries. Regions outside the retained set are removed from subsequent attention, yielding a binary representation: keep the original KV state or drop it entirely.

2.2 Representation-Preserving KV Compression

Representation-preserving methods relax this binary view by carrying information from removed or compressed KV states. CaM merges evicted states into retained entries, while D2O recalls or merges selected discarded states (Zhang et al.,

2024; Wan et al., 2025). KVMerger merges similar KV states (Wang et al., 2024); WeightedKV merges values into retained anchors (Yuan et al., 2025); RazorAttention adds compensation tokens for non-retrieval heads (Tang et al., 2025); and KVSculpt optimizes replacement KV pairs to preserve attention behavior (Jiang and Jin, 2026).

These methods show that information outside the retained token set can remain useful. SurrogateKV follows this representation-preserving direction and stores each admitted surrogate as an ordinary KV entry. Its main distinction is fixed-budget admission: the base compressor supplies capacity and granularity, and SurrogateKV assigns each candidate region to RAW, SURROGATE, or DROP. Unlike pivot-based merging, SurrogateKV needs neither similarity grouping nor pivot assignment and packs the cache once after prefill. QUEST and SparQ instead make query-dependent cache-access decisions during decoding (Tang et al., 2024; Ribar et al., 2024).

3 Motivation

Using LongBench examples (Bai et al., 2024) at budget $B_{KV} = 512$, we examine changes in attention mass and coherence among dropped key and value vectors. Dropping KV entries changes attention mass on highly attended historical chunks, while contiguous SnapKV-dropped chunks show greater coherence for both keys and values than matched random sets.

Attention mass on highly attended chunks. Figure 3(a) measures the absolute change in attention mass on the top-5 chunks selected by recent-query attention under FullKV. Across compressors, the mass on these chunks changes by 3.35–5.96 percentage points after KV entries are dropped. This shift is consistent with prior KV-compression studies showing that removing important KV states or losing information from dropped tokens can degrade long-context benchmark performance (Zhang et al., 2023; Wang et al., 2024; Tang et al., 2025; Yuan et al., 2025). Once a span is dropped, its FullKV attention mass is redistributed over the retained cache, and the dropped span becomes indistinguishable from other removed regions during decoding.

Coherence of keys and values in dropped chunks. For keys or values $X \in \{K, V\}$, we

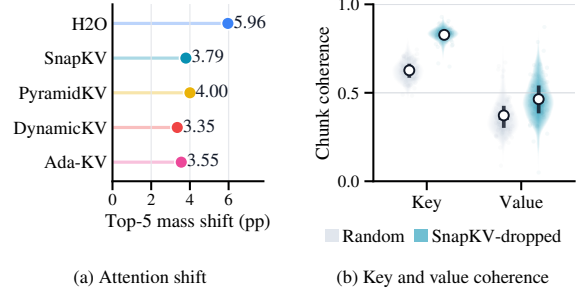


Figure 3: Motivation for surrogate KV. (a) Dropping KV entries changes the attention mass assigned to highly attended historical chunks relative to FullKV. (b) Contiguous SnapKV-dropped chunks have higher cosine coherence for keys and values than matched random sets. Shading shows distributions; circles, vertical lines, and horizontal ticks denote means, interquartile ranges, and medians, respectively.

measure within-set coherence for a token set C as

$$\text{coh}_X(C) = \frac{1}{|C|} \sum_{t \in C} \cos(x_t, \bar{x}_C),$$

$$\bar{x}_C = \frac{1}{|C|} \sum_{t \in C} x_t.$$

Local sets are contiguous 32-token chunks dropped by SnapKV; random controls sample 32 dropped positions from the same example, layer, and KV head. Figure 3(b) shows higher key and value coherence for the local sets, indicating non-random structure without implying that the chunks should be retained at full resolution.

Implication. These observations suggest that KV compression should consider not only which tokens are selected, but also how discarded regions are represented. Rather than forcing every region into keep or drop, SurrogateKV assigns historical regions to raw KV, surrogate KV, or drop: salient regions remain raw, low-scoring regions are removed, and selected lower-salience regions stay available through surrogate KV.

4 SurrogateKV

SurrogateKV is a training-free, representation-level KV cache compressor. Figure 4 summarizes the runtime pipeline. Given a layer or head budget, it assigns each non-recent cache region to RAW, SURROGATE, or DROP. RAW keeps original token-level KV entries, SURROGATE replaces a selected contiguous region with one compact KV pair, and DROP removes it. The resulting packed cache satisfies the budget and is used by the standard attention operator during decoding. We instantiate this step

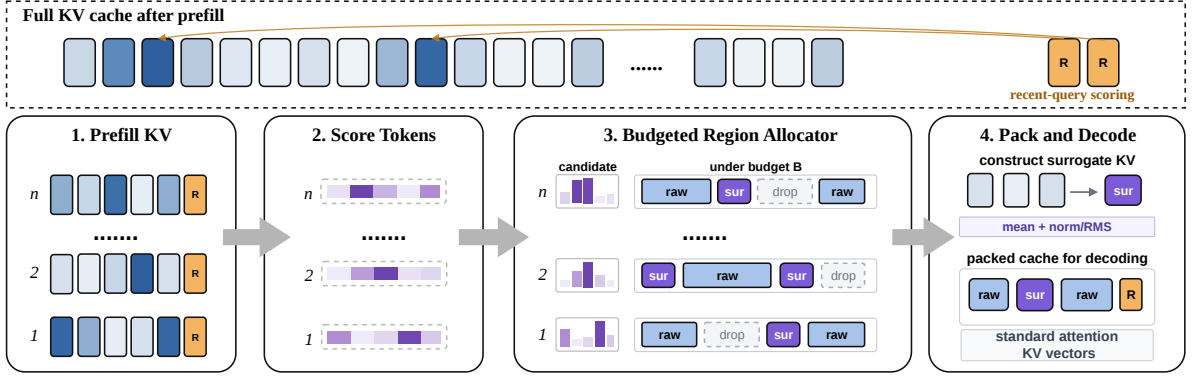


Figure 4: Overview of SurrogateKV. For each supplied layer or head budget, SurrogateKV scores non-recent cache tokens, assigns regions to raw, surrogate, or drop, and decodes with the resulting packed cache.

on SnapKV, DynamicKV, and Ada-KV, denoted SurrogateKV-Snap, SurrogateKV-Dynamic, and SurrogateKV-Ada, respectively.

4.1 Saliency Estimation and Allocation Atoms

For one transformer layer, let $K_h, V_h \in \mathbb{R}^{T \times d}$ be the prefill key and value states for head h ; we omit the head index where the same operation is applied independently to each head. We split the cache into a non-recent part U and a recent suffix R :

$$\{1, \dots, T\} = U \cup R. \quad (1)$$

The recent suffix R is retained, while U is compressed.

Following attention-based KV retention methods (Zhang et al., 2023; Li et al., 2024), we use recent-query attention as a saliency signal. For recent query position $r \in R$ and non-recent token $t \in U$, we compute

$$a_{h,r,t} = \text{softmax}_{\tau \leq r} \left(\frac{q_{h,r} k_{h,\tau}^\top}{\sqrt{d}} \right)_t, \quad (2)$$

for query head h . We aggregate attention over recent queries and apply local max pooling:

$$s_t = \frac{1}{N_h} \sum_{h=1}^{N_h} \max_{t' \in W_\omega(t)} \left(\sum_{r \in R} a_{h,r,t'} \right), \quad (3)$$

where N_h is the number of query heads and $W_\omega(t) = \{t' \in U : |t' - t| \leq \lfloor \omega/2 \rfloor\}$ with boundary clipping. The shared-token and layer-wise variants use the head average in Eq. 3; the head-aware variant retains each query head’s pooled score and performs allocation separately under that head’s budget.

We divide U into contiguous allocation atoms $\{A_j\}$. Atoms define the allocation granularity,

while final surrogate regions can span multiple adjacent atoms. This protects sharp saliency peaks as raw KV while allowing long low-saliency runs to become surrogate regions. Each atom receives average and peak saliency scores:

$$m_j = \text{mean}_{t \in A_j}(s_t), \quad p_j = \max_{t \in A_j} s_t. \quad (4)$$

The allocator combines them after rank normalization:

$$\phi_j = \max(\text{rank}_{[0,1]}(m_j), \text{rank}_{[0,1]}(p_j)), \quad (5)$$

where larger ranks indicate higher saliency. Thus, raw retention protects both broad regional evidence and sharp local evidence.

4.2 Budgeted Region Allocation

Given a budget B , SurrogateKV assigns each non-recent region C_i an action

$$z_i \in \{\text{RAW}, \text{SURROGATE}, \text{DROP}\}. \quad (6)$$

The packed cache satisfies

$$|R| + \sum_{i: z_i = \text{RAW}} |C_i| + \sum_{i: z_i = \text{SURROGATE}} 1 \leq B. \quad (7)$$

The allocator first forms a raw/drop frontier by retaining high-saliency atoms. Contiguous dropped runs seed surrogate candidates. Neighboring candidates may be combined across intervening raw atoms when the merged gain exceeds the sum of their separate gains; an admitted span can therefore displace initially retained atoms. For a candidate D , the allocator evaluates

$$G_B(D) = \underbrace{v_S(D) + b_R(D)}_{\text{preserved value}} - \underbrace{\ell_{\text{in}}(D) + d_{\text{in}}(D) + \ell_{\text{pay}}(D)}_{\text{raw-evidence cost}} - \underbrace{c_{\text{reg}}[\Delta r(D)]_+ + p_B[\delta_B(D)]_+}_{\text{region and slot cost}}. \quad (8)$$

Here $v_S(D)$ is the coherent residual score of D , computed from its atom salience, and $b_R(D)$ credits its whole raw atoms restored with freed slots. The terms ℓ_{in} , d_{in} , and ℓ_{pay} account for raw value displaced inside the span, its unrecovered component, and raw atoms removed outside the span to pay for admission. Δr denotes the change in surrogate-region count, while δ_B is the residual slot change after explicit raw-payer exchanges. Their positive parts are priced by c_{reg} and the current raw-slot price p_B . Before admission, the allocator recomputes $G_B(D)$ under the current allocation and accepts only feasible candidates with positive gain. Adjacent atoms with the same action are then merged into maximal regions before packing. This step is agnostic to the budget source: shared-token, layer-adaptive, or head-aware methods supply the budget, while SurrogateKV decides whether each region should be raw, surrogate, or dropped.

4.3 Surrogate Construction and Decoding

For each surrogate region C_i , SurrogateKV constructs one region-level KV pair by mean-pooling its keys and values separately:

$$\mu_i^K = \frac{1}{|C_i|} \sum_{t \in C_i} k_t, \quad \mu_i^V = \frac{1}{|C_i|} \sum_{t \in C_i} v_t. \quad (9)$$

Because high-mean or high-peak atoms are retained as raw KV first, surrogate regions mainly cover lower-salience contiguous spans. For keys, the scale factor is capped at one to avoid amplifying a low-norm mean key; for values, the constructor restores the source-region RMS norm:

$$\hat{k}_i = \mu_i^K \min \left(\frac{\frac{1}{|C_i|} \sum_{t \in C_i} \|k_t\|_2}{\|\mu_i^K\|_2 + \epsilon}, 1 \right), \quad (10)$$

$$\hat{v}_i = \mu_i^V \frac{\sqrt{\frac{1}{|C_i|} \sum_{t \in C_i} \|v_t\|_2^2}}{\|\mu_i^V\|_2 + \epsilon}. \quad (11)$$

Each allocated region C_i contributes

$$P_i = \begin{cases} \text{KV}_{C_i}, & z_i = \text{RAW}, \\ \widehat{\text{KV}}_i, & z_i = \text{SURROGATE}, \\ \emptyset, & z_i = \text{DROP}. \end{cases} \quad (12)$$

Here KV_{C_i} denotes original token-level KV entries, and $\widehat{\text{KV}}_i = \{(\hat{k}_i, \hat{v}_i)\}$ denotes the single surrogate pair. The compressed cache is

$$\widetilde{\text{KV}} = \text{Pack}(P_1, \dots, P_M, \text{KV}_R), \quad (13)$$

Algorithm 1 SurrogateKV compression

Require: Q, K, V , budget B , recent size n_r

Ensure: Compressed cache $\widetilde{\text{KV}}$

```

1: Split cache into non-recent  $U$  and recent suffix  $R$ 
2: Keep  $R$  raw
   Scoring
3:  $s_U \leftarrow$  recent-query token salience on  $U$   $\triangleright$  Eq. 3
4:  $\{A_j\} \leftarrow$  contiguous atoms over  $U$ 
5:  $\{\phi_j\} \leftarrow$  atom salience from mean/peak ranks  $\triangleright$  Eq. 5
   Allocation
6: Initialize raw/drop atom actions under budget  $B - |R|$ 
7: Seed candidates from dropped runs; merge across raw gaps when joint gain increases
8: Admit feasible candidates with  $G_B(D) > 0$   $\triangleright$  Eq. 8
9:  $\{(C_i, z_i)\}_{i=1}^M \leftarrow$  merge adjacent equal actions
   Packing
10: for  $i = 1, \dots, M$  do
11:   if  $z_i = \text{SURROGATE}$  then
12:      $\widehat{\text{KV}}_i \leftarrow \{(\hat{k}_i, \hat{v}_i)\}$   $\triangleright$  Eqs. 10–11
13:   end if
14: end for
15:  $\widetilde{\text{KV}} \leftarrow$  packed raw entries, surrogate pairs, and  $R$ 
16: return  $\widetilde{\text{KV}}$ 
```

where $\text{Pack}(\cdot)$ concatenates raw entries, surrogate pairs, and the recent suffix along the cache dimension. Since every packed entry is a standard KV pair, decoding uses the ordinary attention kernel. Algorithm 1 gives a high-level pseudocode summary; Appendix B provides a more detailed compression procedure.

5 Experiments

We evaluate whether SurrogateKV improves KV compression under matched KV-slot budgets. Each retained raw KV pair and each surrogate KV pair consumes one slot, so gains reflect representation choice rather than cache size. We report LongBench quality, Needle-in-a-Haystack retrieval, attention analysis, ablation, and serving efficiency.

5.1 Evaluation Setup

Benchmarks and metrics. We evaluate LongBench (Bai et al., 2024) English workloads covering QA, summarization, few-shot learning, synthetic retrieval, and code completion, using each workload’s official metric and reporting the dataset average. We also evaluate Needle-in-a-Haystack (NIAH) retrieval (Kamradt, 2023) over context-length and needle-depth grids. Appendix A gives dataset and metric details.

Models, baselines, and variants. LongBench uses LLaMA-3-8B-Instruct (Grattafiori et al., 2024); NIAH uses Mistral-7B-Instruct-v0.2 (Jiang et al., 2023). Baselines include FullKV,

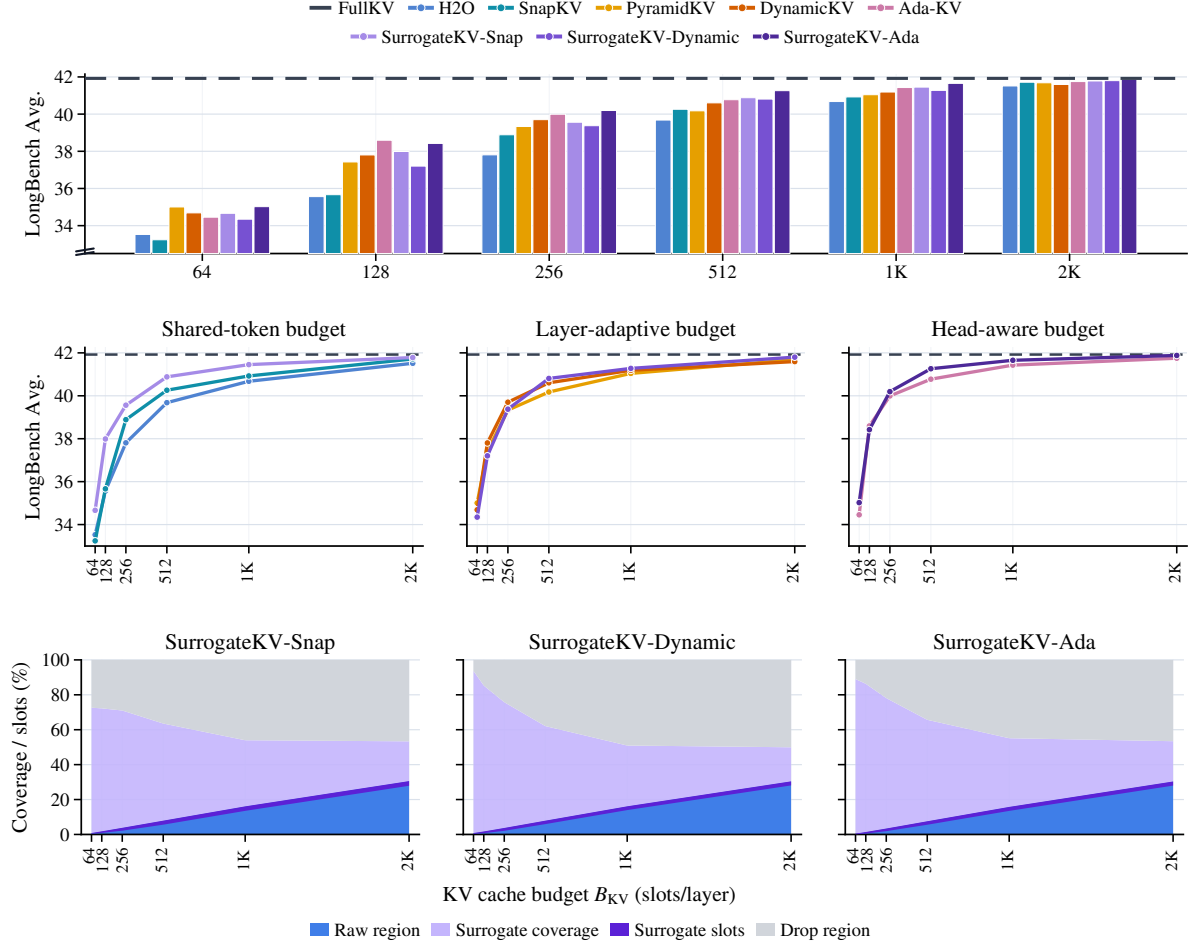


Figure 5: LongBench results under matched per-layer KV cache budgets. Top: all methods by budget; middle: the same curves grouped by budget source, with dashed lines denoting FullKV. Bottom: source-position allocation into raw, surrogate coverage, and drop; surrogate slots use packed-cache slot count, are not part of the stack, and are overlaid downward and magnified.

H₂O (Zhang et al., 2023), SnapKV (Li et al., 2024), PyramidKV (Cai et al., 2025), DynamicKV (Zhou et al., 2025), and Ada-KV (Feng et al., 2025). We group compressed baselines by budget source: H₂O/SnapKV use shared token budgets, PyramidKV/DynamicKV layer-adaptive budgets, and Ada-KV head-aware budgets. We evaluate matched variants SurrogateKV-Snap, SurrogateKV-Dynamic, and SurrogateKV-Ada, comparing each primarily with the baseline using the same budget source. Representation-preserving methods are discussed in Section 2.2; Section 5.3 compares with CaM and D2O.

Budgets and implementation. All compressed methods use matched per-layer KV cache budgets $B_{KV} \in \{64, 128, 256, 512, 1024, 2048\}$. Each budget counts packed prefill KV pairs per layer, including raw entries, surrogate entries, and the recent suffix; generated-token KV is appended identically for all methods. For all SurrogateKV vari-

ants, recent-query salience follows SnapKV’s local max-pooling window $\omega = 7$ (Li et al., 2024), and allocation uses atom size $c = 4$. We fix these values across datasets, budgets, and budget sources so differences reflect representation allocation rather than per-setting tuning. Unless otherwise stated, decoding is deterministic and follows the official benchmark settings.

5.2 LongBench Results

The top and middle panels of Figure 5 report the official LongBench score, averaging the task-specific metrics used by each dataset; higher scores indicate better task quality. Across the sweep, SurrogateKV-Snap improves SnapKV at all six budgets, and SurrogateKV-Ada improves Ada-KV at five of six budgets. Under the shared-token budget, SurrogateKV-Snap improves over SnapKV throughout the sweep, including gains of 2.32 points at $B_{KV} = 128$ and 0.53 at

Budget	Allocation	Method	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		
			NrrvQA	Qasper	MF-en	HotpotQA	2WikiMQA	MuSiQue	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	LCC	RB-P	Avg.
Full cache			25.56	32.15	39.71	43.56	35.72	21.18	28.76	23.21	26.62	74.00	90.48	42.59	4.71	69.25	59.19	54.05	41.92
B _{KV} = 64	Shared token	H2O	19.31	11.33	20.54	34.00	29.85	18.48	19.54	21.92	22.12	40.00	84.30	37.07	5.30	69.25	54.08	49.32	33.53
		SnapKV	19.77	9.02	27.51	35.97	27.85	18.03	16.14	20.71	16.32	38.50	86.40	36.34	4.78	68.83	53.84	51.83	33.24
		SurrogateKV-Snap	21.13	12.15	33.13	37.02	28.92	17.21	15.88	21.38	15.31	51.00	87.64	33.18	5.38	69.33	54.08	51.88	34.66
	Layer adaptive	PyramidKV	21.24	13.93	29.56	34.45	23.49	16.03	18.74	21.83	19.14	58.50	88.31	37.01	4.79	69.50	54.08	49.48	35.01
		DynamicKV	21.82	11.94	30.08	33.93	24.98	17.42	18.54	22.13	18.99	53.50	87.12	35.45	4.98	69.50	54.17	50.44	34.69
		SurrogateKV-Dynamic	20.08	10.59	34.68	37.74	28.15	18.13	16.81	21.08	16.23	47.00	85.77	32.48	5.50	68.45	54.48	52.39	34.35
	Head aware	Ada-KV	20.69	9.67	30.89	40.46	27.77	18.03	16.40	20.83	17.85	39.50	87.78	37.11	4.80	68.33	56.39	54.78	34.45
		SurrogateKV-Ada	21.63	9.98	31.52	39.99	30.61	18.59	16.54	21.44	17.81	40.50	88.62	36.55	5.08	69.33	56.94	55.22	35.02
B _{KV} = 512	Shared token	H2O	23.39	23.16	31.76	41.11	31.63	18.42	24.68	22.36	25.57	69.00	90.37	40.87	5.38	69.50	60.01	57.67	39.68
		SnapKV	24.43	23.42	37.40	42.75	34.44	20.36	22.31	22.61	24.08	70.50	90.52	40.45	5.31	69.50	59.94	56.14	40.26
		SurrogateKV-Snap	23.14	25.95	39.47	44.22	33.89	21.36	21.89	22.96	23.72	72.00	90.52	40.73	5.42	69.50	61.90	57.44	40.88
	Layer adaptive	PyramidKV	24.71	23.60	35.19	43.26	32.12	20.43	23.34	22.97	24.27	72.00	90.61	40.80	5.71	69.50	59.22	55.10	40.18
		DynamicKV	24.79	24.70	36.63	43.60	32.51	20.13	22.98	22.81	24.57	72.00	90.39	40.79	5.61	69.50	61.33	57.34	40.60
		SurrogateKV-Dynamic	24.35	25.86	37.57	43.54	35.49	21.00	22.20	22.25	23.88	72.00	90.61	40.19	5.25	69.00	62.26	57.49	40.81
	Head aware	Ada-KV	25.09	24.57	38.08	43.28	35.02	20.64	22.83	22.62	24.25	73.00	90.61	40.72	5.22	69.25	60.38	56.83	40.77
		SurrogateKV-Ada	25.28	25.72	38.53	44.02	35.38	20.26	22.61	23.12	24.27	73.50	90.48	40.49	5.49	69.25	62.48	59.30	41.26

Table 1: LongBench scores on LLaMA-3-8B-Instruct at $B_{KV} = 64$ and $B_{KV} = 512$ slots/layer. Each cell reports the official score for the corresponding dataset, using that dataset’s task-specific metric; the Avg. column averages these scores across datasets. Rows are grouped by allocation type. Blue shading identifies SurrogateKV variants. Bold marks the best score within each allocation group at the same budget.

1024. The head-aware variant is strongest in our sweep: SurrogateKV-Ada improves over Ada-KV at five of six budgets and reaches the best compressed score at $B_{KV} = 512$, where it remains close to FullKV with a much smaller pre-fill cache. Its gains are 0.49/0.23/0.13 points at 512/1024/2048. On matched Qwen2.5-14B/72B LongBench tasks (Yang et al., 2024), SurrogateKV-Snap exceeds SnapKV by 2.72/2.16 points at $B_{KV} = 512$ (Appendix F).

The layer-adaptive results show that surrogate allocation requires sufficient raw-token capacity: SurrogateKV-Dynamic is below DynamicKV at $B_{KV} \in \{64, 128, 256\}$, but improves over it from $B_{KV} = 512$ onward. From $B_{KV} = 128$ to 512, the raw allocation grows and the measured source tokens represented per surrogate fall from 972 to 290. Over the same range, the gap between SurrogateKV-Dynamic and DynamicKV changes from -0.60 to $+0.20$ points.

The bottom panels distinguish source-position coverage from resident slots: one surrogate slot can represent a longer contiguous run, so its coverage remains visible even when the stored slot band is narrow. As the budget grows, raw coverage increases and the average span represented by each surrogate contracts, leaving more exact KV capacity for sparse evidence.

Table 1 gives the workload-level details behind these averages at $B_{KV} = 64$ and 512, using each dataset’s official metric. At $B_{KV} = 512$, gains concentrate on QA and code. SurrogateKV-Snap

Method	LongBench score	
	$B_{KV} = 128$	$B_{KV} = 512$
CaM	34.78	37.04
D2O	34.79	39.50
SurrogateKV-Snap	37.99	40.88

Table 2: LongBench comparison with merging baselines. CaM and SurrogateKV use fixed slot budgets; D2O uses its released retention-ratio settings targeting approximately 128/512 states. Blue marks SurrogateKV-Snap.

improves over SnapKV on single-document QA, multi-document QA, and code, while SurrogateKV-Ada gives its largest gain on code; the other categories show smaller or budget-dependent changes.

5.3 Comparison with Merging-Based Compression

Table 2 compares SurrogateKV-Snap with CaM (Zhang et al., 2024) and D2O (Wan et al., 2025) at two cache-size settings. CaM and SurrogateKV use fixed slot budgets, whereas D2O uses its released retention-ratio settings targeting approximately 128/512 states. At $B_{KV} = 128/512$, SurrogateKV-Snap exceeds the stronger merging baseline by 3.20/1.38 points. Thus, assigning part of a fixed slot budget to independently addressable regional surrogates can outperform the evaluated merging baselines. Section 2.2 and Appendix C provide design and workload details.

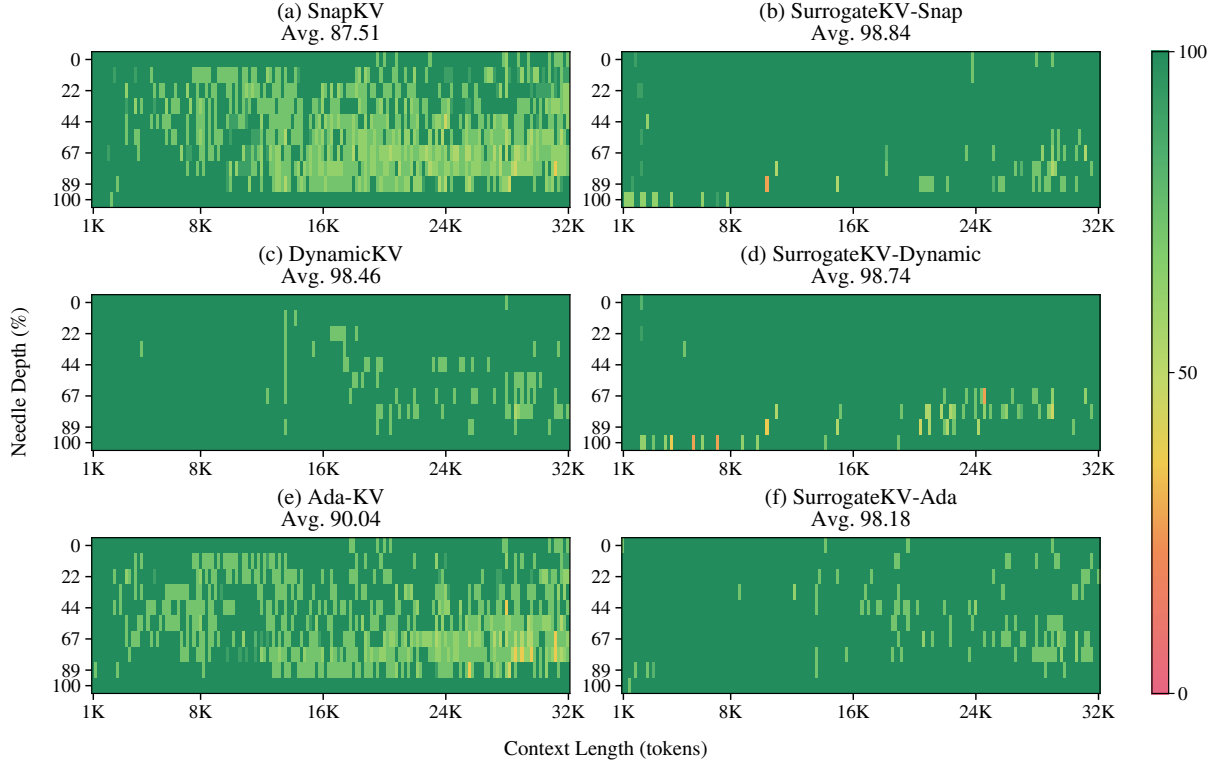


Figure 6: Needle-in-a-Haystack retrieval heatmaps at a fixed per-layer KV cache budget of $B_{KV} = 128$ slots/layer. The x-axis denotes context length, the y-axis denotes needle depth, and scores are reported on a 0–100 scale.

5.4 Needle-in-a-Haystack Retrieval

NIAH isolates remote-evidence retrieval. At $B_{KV} = 128$, all three variants improve over their matched baselines; SurrogateKV-Snap and SurrogateKV-Ada gain 11.33 and 8.14 points, respectively. For SurrogateKV-Ada, accuracy remains high at interior depths (11–89%) and at the 0/100% boundaries (97.84/99.56, respectively). Together, the budget sweep and NIAH results are consistent with raw entries preserving sparse evidence and surrogates restoring coverage over evicted context.

5.5 Attention-Shift Diagnostics

We also measure how compression changes the concentration of the first decoded token’s attention. For each matched example, layer, and head, we compute the normalized entropy shift from FullKV,

$$\Delta_H = \left| \frac{H(a_{\text{method}})}{\log |a_{\text{method}}|} - \frac{H(a_{\text{FullKV}})}{\log |a_{\text{FullKV}}|} \right|.$$

Lower values indicate that the compressed cache preserves the concentration of the FullKV attention pattern more closely. Figure 7 focuses on the head-aware setting, where SurrogateKV-Ada reduces the mean layer-head shift from Ada-KV. Appendix E reports the full diagnostics across

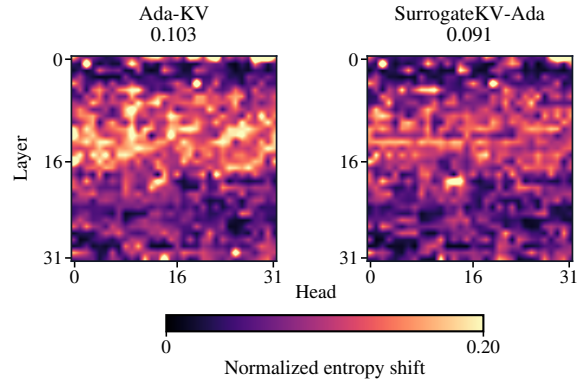


Figure 7: Attention-shift diagnostics for the head-aware allocation group under a per-layer budget of $B_{KV} = 64$ slots/layer. Lower normalized entropy shifts indicate attention concentration closer to FullKV.

all allocation groups; SurrogateKV-Snap and SurrogateKV-Ada reduce the mean shift, while SurrogateKV-Dynamic remains comparable to DynamicKV.

5.6 Ablation Study

Candidate selection and allocation. On 20 examples per workload at $B_{KV} = 512$, removing SURROGATE costs 1.77 points, while fixed quotas of 253–757 surrogates score 40.33–40.51. Thus, gains come from region selection and retained raw KV, not a preset surrogate count.

Ablation	Setting	LB Avg.↑	# Sur.
Full method	SurrogateKV-Snap	42.21	397
No surrogate	Raw/drop only	40.44	0
Fixed quota	$\rho = 0.25$	40.51	253
	$\rho = 0.50$	40.33	505
	$\rho = 0.75$	40.40	757
Constructor	Mean only	42.17	397
	Attention-weighted mean	42.08	397
	Salience-pivot	42.17	397
	Cosine medoid	42.17	397
Atom size	$c = 2$	42.38	551
	$c = 8$	42.07	339
	$c = 16$	41.89	290

Table 3: LongBench ablations on LLaMA-3-8B-Instruct at $B_{KV} = 512$ (16 workloads, 20 examples each). The blue full method uses adaptive admission, mean+norm, and $c = 4$; controls vary quota, constructor, or atom size. # Sur. is the mean admitted surrogate count.

Surrogate construction. With candidate selection and $c = 4$ fixed, alternative constructors remain within 0.14 points of mean+norm, and the four lightweight choices tie at $B_{KV} = 128$. The small spread indicates that constructor choice contributes less than selective admission.

Atom granularity. Finer $c = 2$ atoms add 0.17 points at $B_{KV} = 512$ but raise compression-side time by 69%; $c = 4$ also gives the best method-averaged result at $B_{KV} = 128$ and is the default used throughout the evaluation.

5.7 Serving Efficiency

Table 4 reports 32K-input/8K-output generation on LLaMA-3-8B-Instruct at $B_{KV} = 128$. TTFT includes the one-time construction cost; the resident-KV columns exclude temporary construction buffers and remain fixed because surrogate entries replace, rather than supplement, baseline slots. The implementation scores candidates before materializing prototypes, constructs prototypes only for admitted regions, and directly packs the accepted raw and surrogate entries. Relative to the matched baselines, SurrogateKV adds 0.29–0.30s to TTFT, changes decode throughput by at most 0.25 tokens/s, retains the same resident KV footprint, and changes end-to-end latency by -0.07% to $+0.86\%$. For the merging baselines in Table 2, CaM’s construction-inclusive TTFT is 23.48/23.24 s at $B_{KV} = 128/512$, compared with 0.89/0.95 s for SurrogateKV-Snap; under matched input/output settings, D2O’s generation time per output token is $4.88\times$ that of SurrogateKV-Snap.

Method	Runtime		Resident KV		
	TTFT ↓ (s)	Decode ↑ (tok/s)	Lat. ↓ (s)	Prefill ↓ (GB)	Decode ↓ (GB)
FullKV	4.07	29.31	283.57	4.000	5.000
SnapKV	4.13	32.66	254.91	0.016	1.016
SurrogateKV-Snap	4.42	32.70	254.93	0.016	1.016
DynamicKV	4.15	32.63	255.20	0.016	1.016
SurrogateKV-Dynamic	4.45	32.69	255.02	0.016	1.016
Ada-KV	4.15	33.00	252.36	0.016	1.016
SurrogateKV-Ada	4.44	32.75	254.54	0.016	1.016

Table 4: 32K/8K generation efficiency on LLaMA-3-8B-Instruct at $B_{KV} = 128$. Blue rows mark SurrogateKV variants; Decode is tokens/s and KV columns are resident GQA-compact tensors under the matched slot limit.

Appendix F reports multi-token and context-scaling measurements.

6 Conclusion

We presented SurrogateKV, a training-free KV cache compression method that treats compression as a representation-allocation problem. Under a fixed cache budget, SurrogateKV keeps high-salience regions as raw KV, summarizes selected lower-salience contiguous regions as surrogate KV, and drops the rest. The surrogate representation addresses a limitation observed in prior KV-compression studies, where removing important KV states or losing information from dropped tokens degrades long-context benchmark performance. Across shared-token, layer-adaptive, and head-aware variants, every packed entry remains an ordinary KV pair and decoding uses standard attention.

SurrogateKV-Ada improves over Ada-KV by 0.49 LongBench points at $B_{KV} = 512$ while preserving 98.4% of FullKV performance, and SurrogateKV-Snap improves NIAH retrieval from 87.51 to 98.84 at $B_{KV} = 128$. These quality gains incur a small TTFT overhead from one-time surrogate construction after prefill. Across the sweep, SurrogateKV-Snap improves at all six budgets and SurrogateKV-Ada at five; ablations identify selective admission as the main source of gain. This extends fixed-budget KV compression from token selection to representation allocation.

Limitations

Allocation balance remains important. SurrogateKV relies on the budget supplied by the underlying compressor, and its gains are strongest when that budget leaves room for both raw evidence and surrogate coverage. The layer-adaptive results illustrate this dependence: very small or uneven per-layer budgets can leave too little exact raw KV for sparse evidence. Better joint optimization of layer/head budgets and surrogate selection remains future work.

Surrogate construction is deliberately lightweight. SurrogateKV uses deterministic mean prototypes with norm calibration and constructs them once after prefill. This keeps the method training-free and compatible with standard attention, but heterogeneous regions or sparse query-critical evidence can be blurred by pooling. The scoring, allocation, and construction steps also add a measured one-time prefill-side cost.

The evaluated integrations construct the packed cache once after prefill. H₂O instead updates token-level heavy-hitter scores during decoding, so combining it with regional surrogates requires defining how a variable-length region is scored and updated online. A query-aware selector such as QUEST or SparQ could operate over SurrogateKV’s packed raw and surrogate entries, but this combination is not evaluated here.

Evaluation scope. The main LongBench and NIAH evaluations use LLaMA-3-8B-Instruct and Mistral-7B-Instruct-v0.2, respectively. Matched Qwen2.5-14B/72B LongBench tasks and context-scaling measurements extend the evaluation. More model families and longer real end-to-end workloads would further test generality.

Ethics and Reproducibility Statements

Ethics. SurrogateKV is an inference-time KV-cache compression method. It does not train on additional data, collect user data, or modify model parameters. The main risk is indirect: compression can change which contextual information remains available during decoding, which may affect factuality, retrieval accuracy, or robustness on sensitive inputs. For deployment, cache-compression budgets and settings should therefore be reported together with task quality, and applications with high factual or safety requirements should validate compressed inference under their target workloads.

Reproducibility. Code and artifacts are available at <https://github.com/kimjunewon/SurrogateKV>. The artifact includes the SurrogateKV runtime, evaluation entry points, cache-compression configurations, exported experiment data, NIAH heatmaps, and the CSV files used to reproduce the main tables and figures. Efficiency measurements use Hugging Face generation on NVIDIA A100 80GB PCIe GPUs. We report the public model checkpoints, benchmark sources, budgets, and method variants used in the experiments; model weights and benchmark datasets are not redistributed.

Acknowledgments

This work was supported by the Korea government (MSIT) through the National Research Foundation of Korea (NRF) (RS-2024-00416666 and RS-2025-00564249) and the Institute of Information Communications Technology Planning Evaluation (IITP) (IITP-RS-2026 25547954).

References

- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. [LongBench: A bilingual, multi-task benchmark for long context understanding](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.
- Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. [PyramidKV: Dynamic KV cache compression based on pyramidal information funneling](#). In *Second Conference on Language Modeling*.
- Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. [Extending context window of large language models via positional interpolation](#). *arXiv preprint arXiv:2306.15595*.
- Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. [LongLoRA: Efficient fine-tuning of long-context large](#)

- language models. In *International Conference on Learning Representations*.
- Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. [A dataset of information-seeking questions and answers anchored in research papers](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610.
- Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. [Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084.
- Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025. [Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference](#). In *Advances in Neural Information Processing Systems*, volume 38, pages 113152–113188. Curran Associates, Inc.
- Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. [SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization](#). In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79.
- Aaron Grattafiori et al. 2024. [The Llama 3 herd of models](#). *arXiv preprint arXiv:2407.21783*.
- Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. [LongCoder: A long-range pre-trained language model for code completion](#). In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 12098–12107. PMLR.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. [Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps](#). In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. [KVQuant: Towards 10 million context length LLM inference with KV cache quantization](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 1270–1303.
- Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. [Efficient attentions for long document summarization](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7B](#). *arXiv preprint arXiv:2310.06825*.
- Bo Jiang and Sian Jin. 2026. [KVSculpt: KV cache compression as distillation](#). *arXiv preprint arXiv:2603.27819*.
- Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. [TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1601–1611.
- Greg Kamradt. 2023. LLMTest needle in a haystack: Pressure testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack. GitHub repository.
- Tom    Ko  isk  y, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, G  bor Melis, and Edward Grefenstette. 2018. [The NarrativeQA reading comprehension challenge](#). *Transactions of the Association for Computational Linguistics*, 6:317–328.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with PagedAttention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.
- Xin Li and Dan Roth. 2002. [Learning question classifiers](#). In *COLING 2002: The 19th International Conference on Computational Linguistics*.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. [SnapKV: LLM knows what you are looking for before generation](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 22947–22970.
- Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. [RepoBench: Benchmarking repository-level code auto-completion systems](#). In *International Conference on Learning Representations*.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. [Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time](#). In *Advances in Neural Information Processing Systems*, volume 36.
- Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. [YaRN: Efficient context window extension of large language models](#). In *International Conference on Learning Representations*.

- Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. [SparQ attention: Bandwidth-efficient LLM inference](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 42558–42583. PMLR.
- Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2025. [RazorAttention: Efficient KV cache compression through retrieval heads](#). In *International Conference on Learning Representations*.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. [QUEST: Query-aware sparsity for efficient long-context LLM inference](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 47901–47911. PMLR.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. [MuSiQue: Multi-hop questions via single-hop question composition](#). *Transactions of the Association for Computational Linguistics*, 10:539–554.
- Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. 2025. [D2O: Dynamic discriminative operations for efficient long-context inference of large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. [Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks](#). *arXiv preprint arXiv:2407.08454*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. [Efficient streaming language models with attention sinks](#). In *International Conference on Learning Representations*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. [Qwen2.5 technical report](#). *arXiv preprint arXiv:2412.15115*.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. [HotpotQA: A dataset for diverse, explainable multi-hop question answering](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.
- Jian Yuan, Ziwei He, Haoli Bai, Jingwen Leng, and Bo Jiang. 2025. [WeightedKV: Attention scores weighted key-value cache merging for large language models](#). In *ICASSP 2025 – 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.
- Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. [CaM: Cache merging for memory-efficient LLMs inference](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 58840–58850. PMLR.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. [H₂O: Heavy-hitter oracle for efficient generative inference of large language models](#). In *Advances in Neural Information Processing Systems*, volume 36.
- Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. [QMSum: A new benchmark for query-based multi-domain meeting summarization](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. [DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving](#). In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210. USENIX Association.
- Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. 2025. [DynamicKV: Task-aware adaptive KV cache compression for long context LLMs](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 8042–8057, Suzhou, China. Association for Computational Linguistics.

A Experimental Details

A.1 Model, Decoding, and Budgets

LongBench experiments use LLaMA-3-8B-Instruct (Grattafiori et al., 2024). Needle-in-a-Haystack (NIAH) experiments use Mistral-7B-Instruct-v0.2 (Jiang et al., 2023). Benchmark evaluations use deterministic decoding and the official prompt and maximum-output settings of each benchmark. We sweep per-layer KV cache budgets $B_{KV} \in \{64, 128, 256, 512, 1024, 2048\}$, where one retained raw KV pair and one surrogate KV pair each consume one slot. Generated-token KV is appended identically for all methods. For SurrogateKV, recent-query salience uses local max pooling with window $\omega = 7$, and

allocation uses atom size $c = 4$. The recent suffix contains 32 tokens for the shared-token and head-aware variants and 8 for the layer-adaptive variant, following the corresponding base profiles. Runtime measurements use NVIDIA A100 80GB PCIe GPUs with PyTorch 2.11.0 and Transformers 4.44.2. Serving measurements use batch size 1 on a single GPU without tensor parallelism, FP16 weights, FlashAttention 2, fixed-length greedy decoding, and no EOS early termination. TTFT runs from generation-call start through prefill, cache construction, and first-token selection; decode throughput is $(N_{\text{out}} - 1)/(\text{latency} - \text{TTFT})$, and latency covers the complete generation call for the stated input/output lengths.

A.2 Benchmarks and Metrics

Table 5 summarizes the workloads and metrics used in our evaluation. LongBench scores follow the official evaluation protocol (Bai et al., 2024) and are averaged across datasets. NIAH is reported separately as a retrieval diagnostic and is not included in the LongBench average.

B Detailed Compression Procedure

Algorithm 2 expands the compression procedure in Section 4. It first forms a raw/drop frontier and then admits feasible candidate runs with positive gain under Eq. 8. The action costs are $|C_i|$, 1, and 0 slots for RAW, SURROGATE, and DROP, respectively. Because packed entries are ordinary KV pairs, variable compressed lengths can be handled with standard padding or paged-cache metadata.

C Additional LongBench Results

Table 6 gives the full LongBench table across all evaluated budgets. Table 7 gives the per-workload CaM, D2O, and SurrogateKV-Snap comparison at $B_{KV} = 128$ and 512. For CaM, construction-inclusive TTFT is 23.48/23.24 s at $B_{KV} = 128/512$, compared with 0.89/0.95 s for SurrogateKV-Snap. Under the matched input/output protocol, D2O’s generation time per output token is $4.88\times$ that of SurrogateKV-Snap. Figures 8–10 provide the corresponding workload-level score curves and allocation composition for each allocation group. Each allocation panel averages source-position shares within its workload group, so the shares vary with input length and admitted region span even at the same slot budget.

The surrogate-slot band is enlarged downward for visibility; the underlying rates are unchanged.

D Additional Needle-in-a-Haystack Results

Figure 11 reports the same NIAH diagnostic as Section 5.4, but under the more aggressive budget $B_{KV} = 64$. At this budget, SurrogateKV-Snap improves SnapKV from 66.08 to 85.17 average retrieval accuracy, and SurrogateKV-Ada improves Ada-KV from 72.13 to 97.60. SurrogateKV-Dynamic is lower than DynamicKV at this tight budget, consistent with its LongBench budget trend.

E Additional Attention-Shift Diagnostics

Figure 12 reports layer-head and workload-level attention shifts across the three allocation groups.

F Model, Context, and Layer-Wise Scaling

Table 8 reports matched Qwen2.5-14B and Qwen2.5-72B LongBench task results (Yang et al., 2024). At $B_{KV} = 512$, SurrogateKV-Snap improves SnapKV by 2.72 and 2.16 points; at $B_{KV} = 128$, the changes are +0.55 and -0.43 . At $B_{KV} = 512$, both larger models show positive gains; the mixed results at $B_{KV} = 128$ indicate that model size alone does not remove sensitivity to cache budget.

In a separate one-token context-scaling run at $B_{KV} = 128$, TTFT overhead is 0.29/0.57s at 8K/32K on LLaMA-3-8B and 0.28/0.80s on Qwen2.5-14B; model loading and tokenization are excluded. As baseline TTFT grows with context length, the relative overhead falls from 39.1% to 14.5% and from 20.2% to 10.7%, respectively. From $B_{KV} = 128$ to 512, SurrogateKV-Dynamic moves from -0.60 to $+0.20$ relative to DynamicKV. Over the same range, source tokens represented per surrogate fall from 972 to 290 as raw allocation grows, matching the recovery in the LongBench sweep.

Table 9 reports generation efficiency at 8K/2K and 16K/4K input/output lengths. Across these settings, the SurrogateKV variants add 0.13–0.20s to TTFT, keep the same resident KV footprints, and change decode throughput by at most 0.69 tokens/s relative to their matched baselines.

Benchmark	Category	Workloads	Language	Metric
LongBench	Single-doc QA	NarrativeQA (Kočíský et al., 2018), Qasper (Dasigi et al., 2021), MultiFieldQA-en (Bai et al., 2024)	English	F1
	Multi-doc QA	HotpotQA (Yang et al., 2018), 2WikiMQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022)	English	F1
	Summarization	GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019)	English	ROUGE-L
	Few-shot learning	TREC (Li and Roth, 2002), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019)	English	Acc., F1, ROUGE-L
	Synthetic retrieval	PassageCount, PassageRetrieval-en (Bai et al., 2024)	English	Accuracy
	Code completion	LCC (Guo et al., 2023), RepoBench-P (Liu et al., 2024)	Python, Java, C#	Edit similarity
NIAH	Retrieval diagnostic	Synthetic needle retrieval over varying context lengths and needle depths	English	Retrieval accuracy

Table 5: Workload categories, languages, and metrics for LongBench (Bai et al., 2024) and Needle-in-a-Haystack (NIAH) (Kamradt, 2023).

Algorithm 2 Detailed SurrogateKV compression procedure

Require: Prefill states Q, K, V , cache budget B , recent size n_r , atom size c , pooling window ω

Ensure: Packed compressed cache $\widetilde{KV}$

- 1: Split cache positions into non-recent prefix $U = \{1, \dots, T - n_r\}$ and recent suffix $R = \{T - n_r + 1, \dots, T\}$
- 2: Keep all tokens in R as RAW; set remaining budget $B' \leftarrow B - |R|$

Step 1: Saliency and atoms

- 3: Compute recent-query saliency $\{s_t\}_{t \in U}$ using Eq. 3
- 4: Partition U into contiguous atoms $\{A_j\}_{j=1}^J$ of size c
- 5: For every A_j , set $m_j \leftarrow \text{mean}_{t \in A_j}(s_t)$ and $p_j \leftarrow \max_{t \in A_j} s_t$
- 6: Set every ϕ_j from the rank-normalized m_j and p_j using Eq. 5

Step 2: Raw/drop frontier

- 7: Initialize every atom in U as DROP; set used budget $b \leftarrow 0$
- 8: **for** each atom A_j in descending order of ϕ_j **do**
- 9: **if** $b + |A_j| \leq B'$ **then**
- 10: Mark A_j as RAW; update $b \leftarrow b + |A_j|$
- 11: **end if**
- 12: **end for**

Step 3: Surrogate selection

- 13: Seed candidates with maximal contiguous DROP runs bounded by RAW
- 14: Add beneficial combinations of neighboring candidates across intervening RAW atoms
- 15: **for** each candidate D_k , in descending initial gain order **do**
- 16: Recompute $G_B(D_k)$ and the lowest-value outside RAW payer set under the current allocation
- 17: **if** D_k is feasible and $G_B(D_k) > 0$ **then**
- 18: Mark raw atoms absorbed by D_k and its payer set as DROP
- 19: Mark D_k as SURROGATE
- 20: Refill whole-atom slack with the highest-value dropped atoms that fit
- 21: Recompute b from the RAW/SURROGATE/DROP action costs
- 22: **end if**
- 23: **end for**

Step 4: Region formation and packing

- 24: Merge adjacent atoms with the same action into maximal regions $\{(C_i, z_i)\}_{i=1}^M$
- 25: Initialize packed cache $\widetilde{KV} \leftarrow \emptyset$
- 26: **for** $i = 1, \dots, M$ **do**
- 27: **if** $z_i = \text{RAW}$ **then**
- 28: Append original token-level entries KV_{C_i}
- 29: **else if** $z_i = \text{SURROGATE}$ **then**
- 30: Compute mean prototypes μ_i^K, μ_i^V and calibrated surrogate pair $(\hat{k}_i, \hat{v}_i)$ using Eqs. 10–11
- 31: Append surrogate pair $\{(\hat{k}_i, \hat{v}_i)\}$
- 32: **end if**
- 33: **end for**
- 34: Append recent suffix KV_R to $\widetilde{KV}$
- 35: **return** $\widetilde{KV}$

			Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		
Budget	Allocation	Method	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	MuSiQue	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	LCC	RB-P	Avg.
Full cache			25.56	32.15	39.71	43.56	35.72	21.18	28.76	23.21	26.62	74.00	90.48	42.59	4.71	69.25	59.19	54.05	41.92
$B_{KV} = 64$	Shared token	H2O	19.31	11.33	20.54	34.00	29.85	18.48	19.54	21.92	22.12	40.00	84.30	37.07	5.30	69.25	54.08	49.32	33.53
		SnapKV	19.77	9.02	27.51	35.97	27.85	18.03	16.14	20.71	16.32	38.50	86.40	36.34	4.78	68.83	53.84	51.83	33.24
		SurrogateKV-Snap	21.13	12.15	33.13	37.02	28.92	17.21	15.88	21.38	15.31	51.00	87.64	33.18	5.38	69.33	54.08	51.88	34.66
	Layer adaptive	PyramidKV	21.24	13.93	29.56	34.45	23.49	16.03	18.74	21.83	19.14	58.50	88.31	37.01	4.79	69.50	54.08	49.48	35.01
		DynamicKV	21.82	11.94	30.08	33.93	24.98	17.42	18.54	22.13	18.99	53.50	87.12	35.45	4.98	69.50	54.17	50.44	34.69
		SurrogateKV-Dynamic	20.08	10.59	34.68	37.74	28.15	18.13	16.81	21.08	16.23	47.00	85.77	32.48	5.50	68.45	54.48	52.39	34.35
Head adaptive	Ada-KV	20.69	9.67	30.89	40.46	27.77	18.03	16.40	20.83	17.85	39.50	87.78	37.11	4.80	68.33	56.39	54.78	34.45	
	SurrogateKV-Ada	21.63	9.98	31.52	39.99	30.61	18.59	16.54	21.44	17.81	40.50	88.62	36.55	5.08	69.33	56.94	55.22	35.02	
$B_{KV} = 128$	Shared token	H2O	23.45	14.55	25.29	35.68	29.09	18.05	21.54	21.59	23.75	46.50	88.23	38.46	5.50	68.75	56.56	52.00	35.56
		SnapKV	20.98	13.85	31.80	36.61	29.85	19.18	19.19	21.54	20.55	45.00	87.80	37.78	4.63	68.85	57.70	55.33	35.66
		SurrogateKV-Snap	22.98	14.14	35.40	43.71	32.52	20.95	18.70	21.49	19.22	66.00	89.73	36.31	5.25	69.50	56.78	55.14	37.99
	Layer adaptive	PyramidKV	21.86	17.12	31.90	38.59	28.82	18.67	19.98	22.35	20.72	67.00	89.35	38.37	5.42	69.00	57.63	52.04	37.43
		DynamicKV	22.23	14.85	34.30	41.23	28.93	19.87	20.37	22.22	21.11	66.00	89.34	38.70	4.22	69.50	57.64	54.34	37.80
		SurrogateKV-Dynamic	22.38	16.97	35.01	40.60	31.66	19.65	18.92	21.46	20.23	60.00	88.69	32.47	5.26	69.50	57.86	54.59	37.20
Head adaptive	Ada-KV	23.16	17.15	33.99	42.28	35.29	19.34	19.76	22.19	21.04	63.50	90.72	38.71	5.54	69.50	58.50	56.85	38.59	
	SurrogateKV-Ada	20.93	18.29	34.57	41.04	35.19	20.54	19.70	21.79	20.98	61.50	90.81	37.52	5.85	69.25	60.19	56.62	38.42	
$B_{KV} = 256$	Shared token	H2O	23.41	18.22	28.60	38.20	33.24	17.38	22.94	21.98	24.83	59.50	88.46	40.06	5.02	68.83	59.98	54.28	37.81
		SnapKV	23.97	18.24	34.43	42.07	34.02	20.96	20.67	22.38	22.35	62.50	90.14	39.44	5.25	69.50	59.66	56.63	38.89
		SurrogateKV-Snap	23.81	19.72	35.67	44.44	33.44	21.03	20.03	22.49	21.95	69.50	90.44	39.59	5.25	69.00	60.47	56.17	39.56
	Layer adaptive	PyramidKV	23.97	20.24	35.41	42.47	31.44	20.12	21.27	22.74	22.60	71.50	90.48	40.00	5.33	69.50	58.38	53.88	39.33
		DynamicKV	23.64	19.57	36.59	43.29	33.46	20.13	21.66	22.52	22.89	71.00	90.02	39.80	4.97	69.50	59.51	55.78	39.70
		SurrogateKV-Dynamic	23.30	22.77	36.99	43.14	32.32	20.05	20.36	22.60	22.37	69.50	90.64	35.96	5.05	69.00	60.45	56.42	39.38
Head adaptive	Ada-KV	24.52	22.42	37.61	43.90	33.01	20.62	21.48	21.90	22.84	70.00	90.32	39.58	5.37	69.50	60.48	56.34	39.99	
	SurrogateKV-Ada	24.37	22.74	38.26	43.27	33.43	20.74	21.22	22.08	22.77	69.50	90.66	39.38	5.87	69.50	60.93	58.38	40.19	
$B_{KV} = 512$	Shared token	H2O	23.39	23.16	31.76	41.11	31.63	18.42	24.68	22.36	25.57	69.00	90.37	40.87	5.38	69.50	60.01	57.67	39.68
		SnapKV	24.43	23.42	37.40	42.75	34.44	20.36	22.31	22.61	24.08	70.50	90.52	40.45	5.31	69.50	59.94	56.14	40.26
		SurrogateKV-Snap	23.14	25.95	39.47	44.22	33.89	21.36	21.89	22.96	23.72	72.00	90.52	40.73	5.42	69.50	61.90	57.44	40.88
	Layer adaptive	PyramidKV	24.71	23.60	35.19	43.26	32.12	20.43	23.34	22.97	24.27	72.00	90.61	40.80	5.71	69.50	59.22	55.10	40.18
		DynamicKV	24.79	24.70	36.63	43.60	32.51	20.13	22.98	22.81	24.57	72.00	90.39	40.79	5.61	69.50	61.33	57.34	40.60
		SurrogateKV-Dynamic	24.35	25.86	37.57	43.54	35.49	21.00	22.20	22.25	23.88	72.00	90.61	40.19	5.25	69.00	62.26	57.49	40.81
Head adaptive	Ada-KV	25.09	24.57	38.08	43.28	35.02	20.64	22.83	22.62	24.25	73.00	90.61	40.72	5.22	69.25	60.38	56.83	40.77	
	SurrogateKV-Ada	25.28	25.72	38.53	44.02	35.38	20.26	22.61	23.12	24.27	73.50	90.48	40.49	5.49	69.25	62.48	59.30	41.26	
$B_{KV} = 1024$	Shared token	H2O	24.90	27.20	35.04	41.80	31.72	18.54	25.97	22.57	26.16	73.00	91.32	41.18	5.32	69.25	59.38	57.47	40.68
		SnapKV	24.91	25.92	37.67	43.84	35.07	20.06	24.29	22.43	25.57	73.00	90.56	40.54	5.16	69.50	60.27	56.25	40.92
		SurrogateKV-Snap	25.67	29.93	38.35	44.36	34.57	21.10	24.25	22.88	25.18	72.50	90.56	41.56	5.14	69.00	60.69	57.50	41.45
	Layer adaptive	PyramidKV	25.38	26.37	37.83	43.93	34.37	21.69	24.63	22.56	25.54	72.50	90.56	41.68	5.38	69.25	59.65	55.37	41.04
		DynamicKV	25.13	26.99	39.03	43.90	34.22	20.13	24.27	22.79	25.40	72.50	90.56	41.65	5.31	69.50	60.79	56.83	41.19
		SurrogateKV-Dynamic	25.82	28.91	39.21	43.30	34.28	20.27	24.32	22.69	25.07	72.50	90.56	41.88	5.19	69.00	60.70	56.73	41.28
Head adaptive	Ada-KV	25.32	28.84	39.24	43.46	35.61	20.41	24.28	23.10	25.48	73.50	90.56	41.61	4.96	69.25	60.47	56.78	41.43	
	SurrogateKV-Ada	25.38	29.20	39.02	44.02	36.64	20.20	24.09	22.85	25.33	74.00	90.43	42.13	5.42	69.25	60.81	57.70	41.65	
$B_{KV} = 2048$	Shared token	H2O	26.09	28.86	37.18	43.29	34.09	19.98	27.53	23.03	26.60	73.50	90.93	41.80	5.38	69.50	59.30	57.16	41.51
		SnapKV	25.86	29.50	39.41	43.84	35.88	21.41	25.85	23.23	26.32	74.00	90.56	41.68	5.16	69.25	59.86	55.63	41.72
		SurrogateKV-Snap	26.29	30.73	39.15	43.89	35.35	21.65	26.29	23.23	26.22	73.00	90.56	42.24	5.14	68.75	59.66	56.37	41.78
	Layer adaptive	PyramidKV	25.40	29.69	38.49	43.69	35.86	21.53	27.09	23.29	26.14	73.50	90.56	42.17	4.93	69.25	59.90	55.47	41.69
		DynamicKV	25.44	28.70	39.47	43.53	34.99	21.42	26.06	23.33	26.33	73.00	90.56	42.49	5.23	69.25	59.52	56.18	41.59
		SurrogateKV-Dynamic	25.94	30.68	40.11	43.52	34.84	21.55	26.34	23.16	26.35	73.00	90.56	42.15	5.04	69.00	59.78	56.79	41.80
Head adaptive	Ada-KV	25.67	30.80	39.56	43.56	35.79	21.55	25.62	23.46	26.39	74.00	90.56	42.23	4.59	69.25	59.47	55.51	41.75	
	SurrogateKV-Ada	25.84	30.69	39.85	44.11	35.72	21.84	26.46	23.37	26.31	74.00	90.64	42.02	4.79	69.25	59.72	55.47	41.88	

Table 6: Full LongBench results; blue rows denote SurrogateKV variants and bold marks the best score within each allocation group.

		Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		
Budget	Method	NrrvQA	Qasper	MF-en	HotpotQA	2WikiMQA	MuSiQue	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	LCC	RB-P	Avg.
$B_{KV} = 128$	CaM	19.91	14.37	31.45	37.53	26.63	20.16	20.55	22.29	20.27	34.50	89.17	39.63	4.67	70.50	54.67	50.21	34.78
	D2O	23.64	17.34	26.35	41.45	32.99	17.97	22.78	18.32	18.65	54.00	89.77	33.33	4.80	69.12	40.05	46.04	34.79
	SurrogateKV-Snap	22.98	14.14	35.40	43.71	32.52	20.95	18.70	21.49	19.22	66.00	89.73	36.31	5.25	69.50	56.78	55.14	37.99
$B_{KV} = 512$	CaM	22.39	19.81	34.70	41.95	29.77	19.97	21.77	22.46	23.52	39.50	90.29	41.67	4.65	70.25	56.34	53.63	37.04
	D2O	24.14	20.18	31.89	42.60	37.26	19.64	23.68	22.16	23.61	69.00	91.36	40.44	5.79	69.00	55.78	55.45	39.50
	SurrogateKV-Snap	23.14	25.95	39.47	44.22	33.89	21.36	21.89	22.96	23.72	72.00	90.52	40.73	5.42	69.50	61.90	57.44	40.88

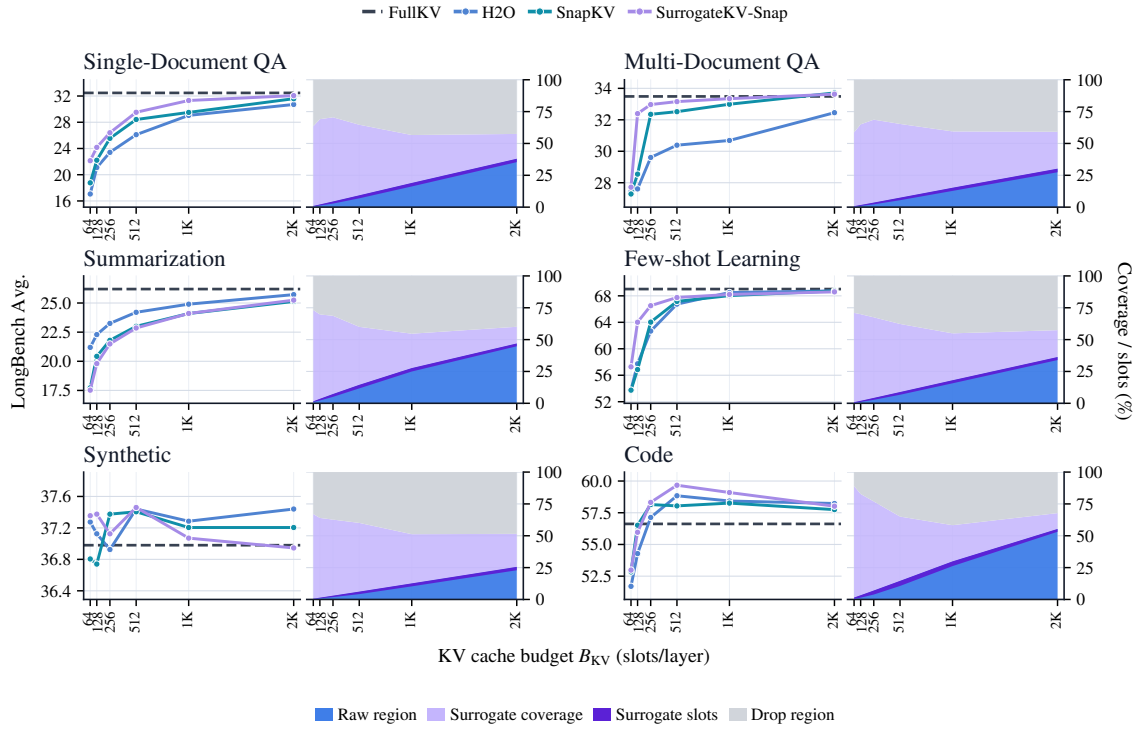


Figure 8: Workload-level LongBench score curves and source-position allocation for the shared-token budget group.

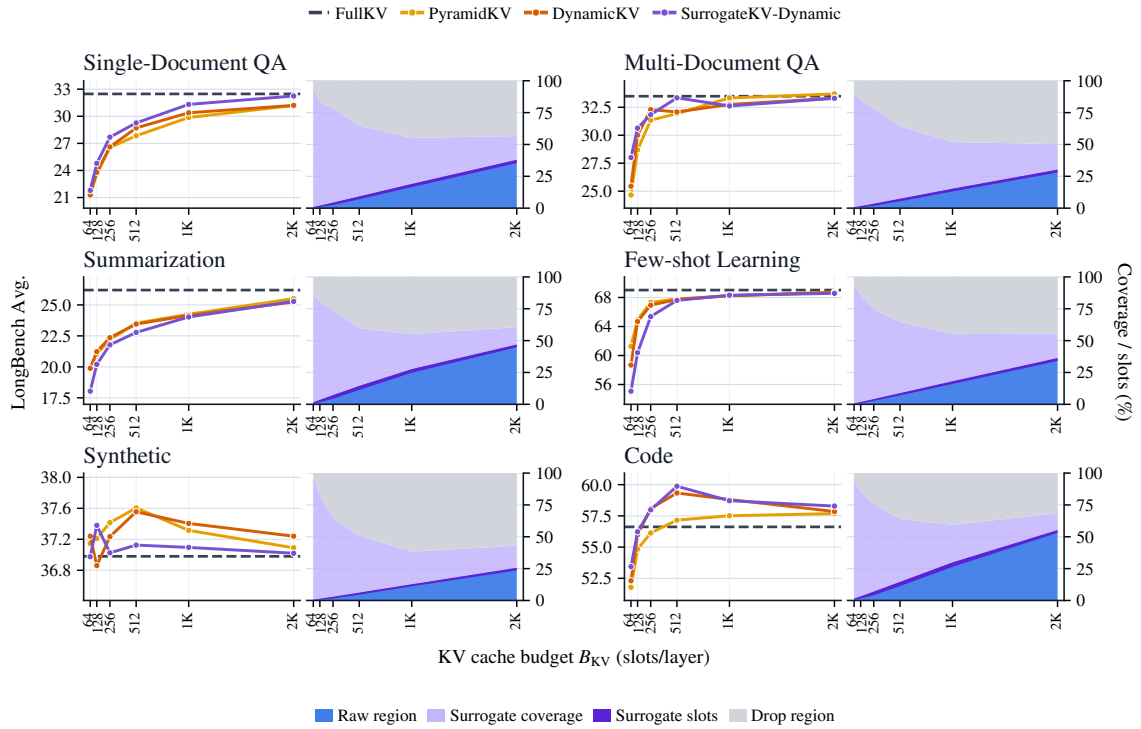


Figure 9: Workload-level LongBench score curves and source-position allocation for the layer-adaptive budget group.

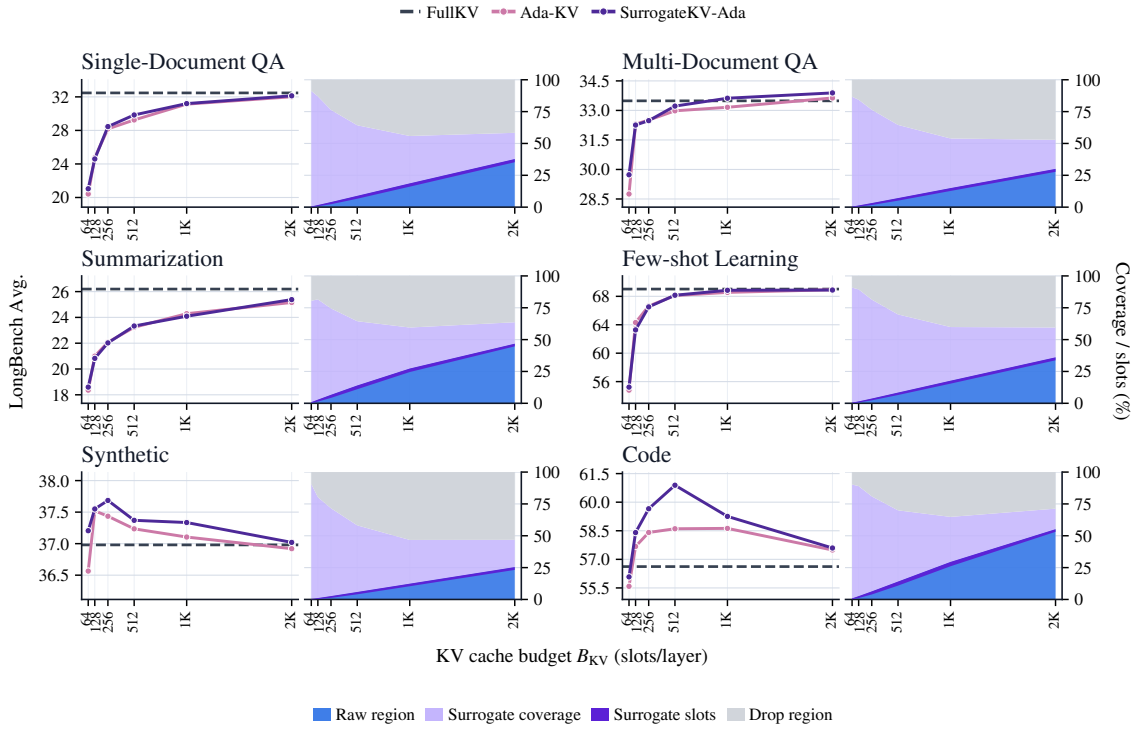


Figure 10: Workload-level LongBench score curves and source-position allocation for the head-aware budget group.

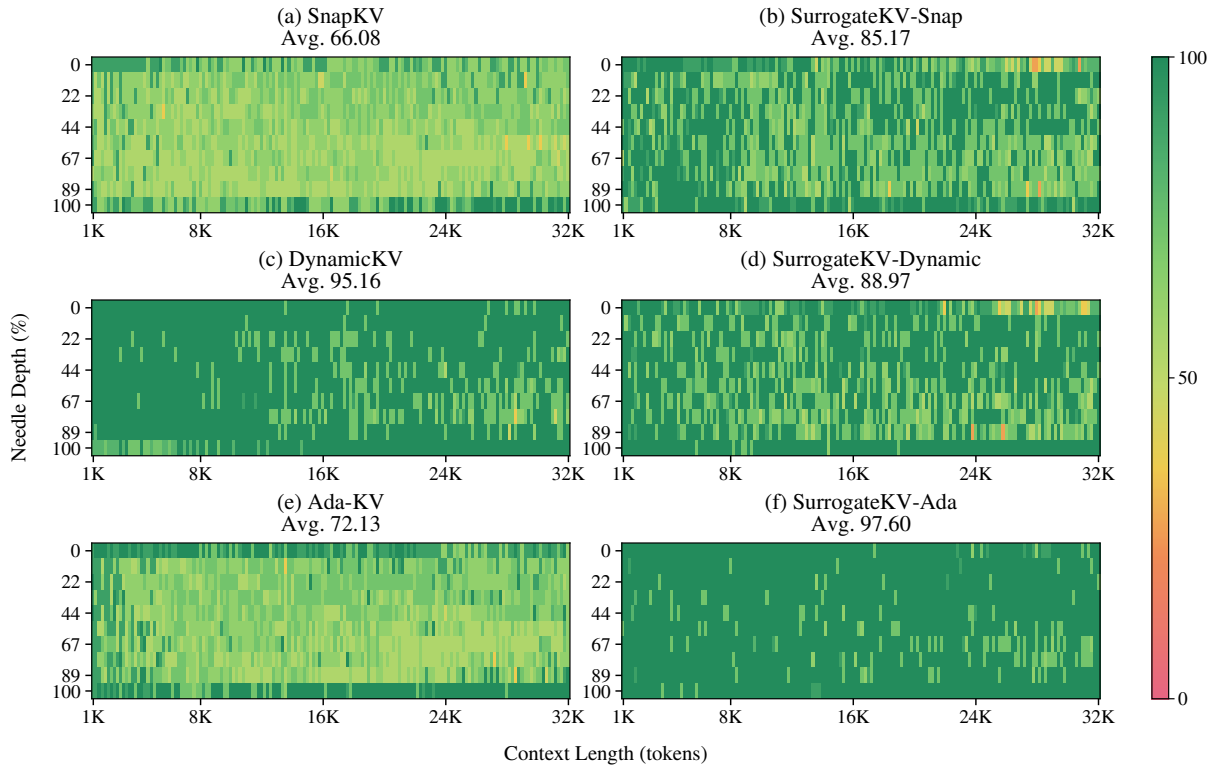


Figure 11: Needle-in-a-Haystack retrieval heatmaps at $B_{KV} = 64$ slots/layer. The x-axis denotes context length, the y-axis denotes needle depth, and scores are reported on a 0–100 scale. SurrogateKV-Snap and SurrogateKV-Ada recover more needle positions than their matched baselines at this aggressive budget (66.08 to 85.17 and 72.13 to 97.60, respectively), while SurrogateKV-Dynamic is less reliable.

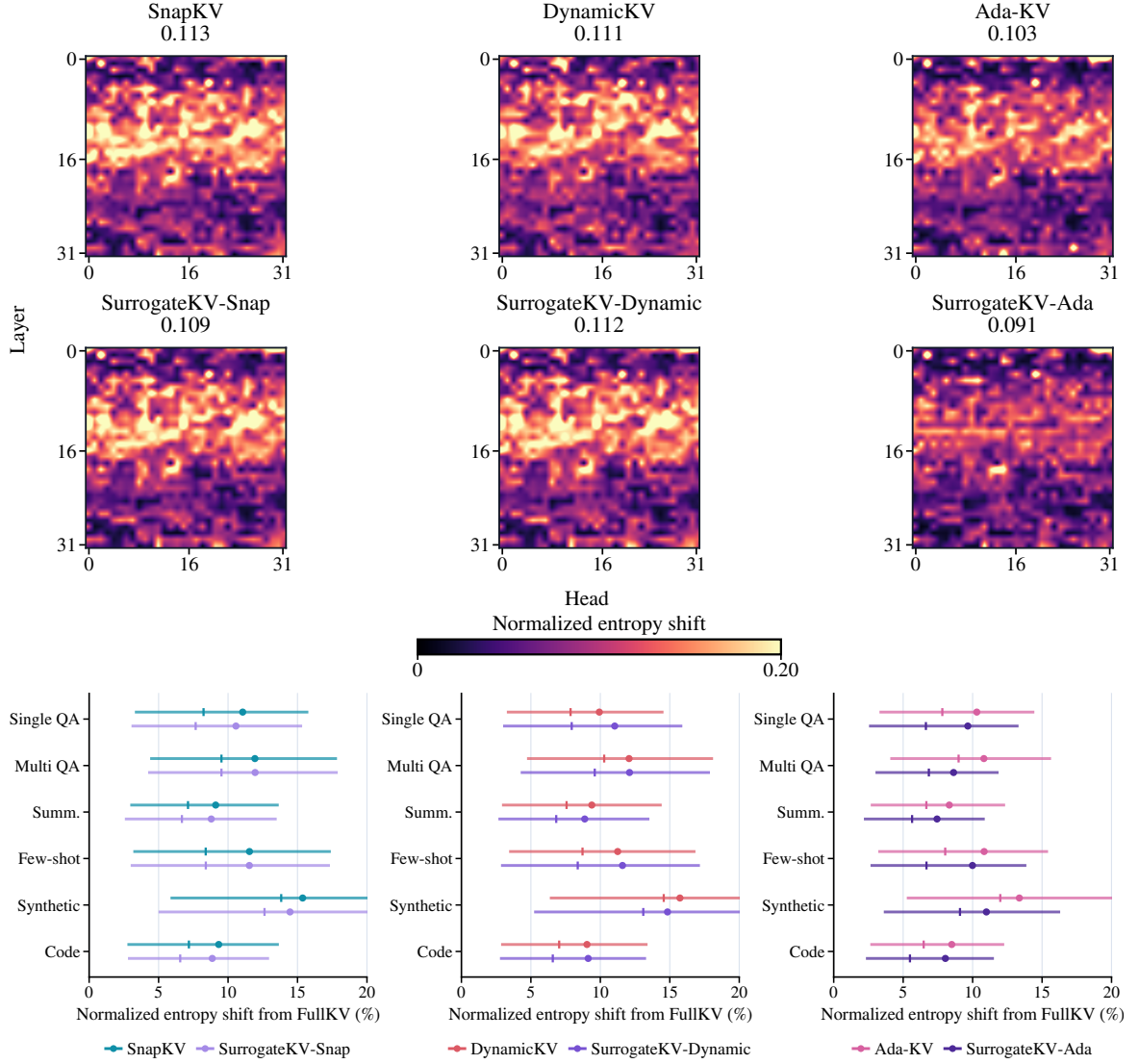


Figure 12: Attention-shift diagnostics at $B_{KV} = 64$ slots/layer. Top: layer-head shifts, with titles reporting means. Bottom: workload distributions; square markers, vertical ticks, and horizontal bars denote the mean, median, and interquartile range. Lower normalized entropy shifts indicate attention concentration closer to FullKV.

Input	Output	Method	TTFT ↓ (s)	Decode ↑ (tok/s)	Lat. ↓ (s)	Prefill KV ↓ (GB)	Decode KV ↓ (GB)
8K	2K	FullKV	0.80	32.67	63.45	1.000	1.250
8K	2K	SnapKV	0.76	32.86	63.06	0.016	0.266
8K	2K	SurrogateKV-Snap	0.94	32.99	63.00	0.016	0.266
8K	2K	DynamicKV	0.79	32.87	63.06	0.016	0.266
8K	2K	SurrogateKV-Dynamic	0.95	32.18	64.55	0.016	0.266
8K	2K	Ada-KV	0.78	32.83	63.14	0.016	0.266
8K	2K	SurrogateKV-Ada	0.91	32.78	63.35	0.016	0.266
16K	4K	FullKV	1.67	32.19	128.90	2.000	2.500
16K	4K	SnapKV	1.70	32.70	126.93	0.016	0.516
16K	4K	SurrogateKV-Snap	1.88	32.72	127.05	0.016	0.516
16K	4K	DynamicKV	1.72	32.61	127.31	0.016	0.516
16K	4K	SurrogateKV-Dynamic	1.89	32.76	126.90	0.016	0.516
16K	4K	Ada-KV	1.71	32.97	125.94	0.016	0.516
16K	4K	SurrogateKV-Ada	1.91	32.80	126.75	0.016	0.516

Table 9: Generation efficiency at 8K/2K and 16K/4K input/output lengths on LLaMA-3-8B-Instruct at $B_{KV} = 128$. Runtime reports TTFT, decode throughput, and end-to-end latency; KV columns report resident GQA-compact prefill and decode tensors.