

Kwai Summary Attention V2 Technical Report

OneRec Team

Summary-token architectures such as Kwai Summary Attention (KSA) compress distant context into learnable Summary token per chunk, and hybrid variants interleave these layers with full attention to retain exact access to the original text. In such hybrid KSA-Full architecture, the interleaving inserted summary tokens are served as a streaming flow role across different layers. Although effective, however, it also introduce some bottlenecks: (1) the different layers summary compression chunk must be same, therefore hard to reach a flexible KV Cache compression, e.g., different layer set different summary compression chunk window. (2) in the full-attention layers, the summary token introduced additional computation cost, but such layer ignore them and still pay quadratic prefilling, therefore unfriendly to the computation side. Motivated by them, we developed the KSA V2, which not only achieves a more flexible KV cache memory compression but also reduces the computation cost in Full attention layer. Specifically, we found that the summary tokens can serve two roles: the KV memory and sparse indexer. As KV memory, they can be further compressed without coarsening the representation passed between layers: Compressed KSA (C-KSA) pools groups of Summary KV into layer-local Meta entries, raising a larger chunk distant history compression. As sparse index, the summary token could conduct exact-access layers where to look: Sparse KSA (S-KSA) builds block-level routing keys from the Summaries and attends only to selected original-text blocks. We conduct experiments based on the 1.9B (MHA) and 4B (GQA) Hybrid KSA backbones, and we mostly adopt them by training the new compressed and sparse modules. In 1.9B (MHA) and 128K sequence scale, it reduces resident cache size by 40.4% and achieves $1.51\times$ prefill and $1.27\times$ decode throughput.

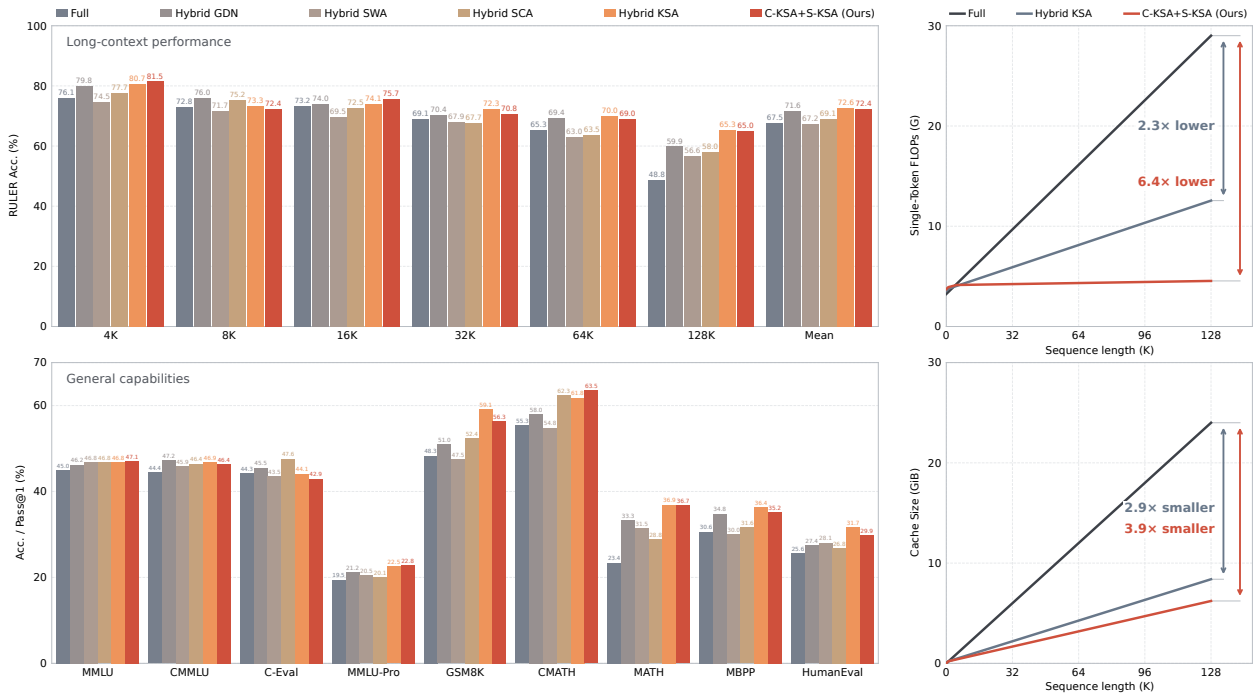


Figure 1 | Left: long-context and general capabilities of 1.9B C-KSA+S-KSA and the reference models reported in KSA. Right: per token inference FLOPs and cache size (1GiB = 2^{30} bytes).

1. Introduction

Serving long contexts incurs substantial storage and computation costs. With full attention, the key-value (KV) cache grows linearly with context length, prefill attention is quadratic, and each decoding query reads the full history. Per-token KV compression (Ainslie et al., 2023; DeepSeek-AI et al., 2024) reduces storage per position but retains an entry for every token. Sparse attention (DeepSeek-AI et al., 2025; Lu et al., 2025; Yuan et al., 2025) reduces each query’s read set while retaining token-level KV. Sliding-window and recurrent linear attention (Jiang et al., 2023; Yang et al., 2025) reduce both storage and computation, but do not preserve direct access to distant tokens. Hybrid architectures balance these trade-offs by interleaving compact-memory layers with full-attention layers that retain direct access to the original text.

Summary-token hybrids are a strong instance of this design. Kwai Summary Attention (KSA) (Chu et al., 2026) appends a learned Summary token after every k_s text tokens; in KSA layers, text queries read nearby chunks at full resolution and distant chunks through their Summaries. Hybrid KSA interleaves three such layers with one full-attention layer (Figure 2) and retrieves long-range information better than a same-size full-attention model.

In these hybrids, however, the Summaries are underused. They are computed and refined at every layer, yet serve only as low-resolution memory for KSA layers, leaving an inefficiency on each side. On the efficient side, compression is tied to chunk size: a KSA layer stores one entry per distant chunk, so compressing further requires larger chunks, which also coarsen the Summaries that every later layer reads and refines. The KSA report’s 4B ablation shows that coarser Summary chunks can degrade retrieval performance. On the exact side, full-attention layers read every token, although most barely matter. At 128K, oracle block selection restricts the main-attention read set to about 4% of text positions, yet retains about 82% of decode attention mass and near-dense retrieval performance in a frozen Hybrid KSA (Figure 12; Table 10). What is missing is a cheap way to know where to look, and the Summaries, which describe every chunk, are never consulted.

We argue that a single summary hierarchy can play both roles. As memory, Summaries can be compressed a second time when stored, without coarsening the representation that is propagated; as an index, they can direct exact-access layers to the few original-text blocks worth reading. *Efficient layers store coarse, exact layers read fine, and the Summaries connect the two.*

We instantiate this idea with two drop-in layer types, which together form **KSA V2**. **Compressed KSA (C-KSA)** pools every k_m consecutive Summary KV entries into one layer-local Meta entry, using learned dimension-wise weights as in DeepSeek-V4 (DeepSeek-AI et al., 2026). Meta entries are never fed forward, so the Summary sequence keeps its granularity while distant-history storage shrinks by a factor of $k_s k_m$ ($1024\times$ with $k_s = 8$ and $k_m = 128$). **Sparse KSA (S-KSA)** replaces full attention: each Summary pools the tokens of each b -token block in its chunk into a routing key, a Lightning Indexer (DeepSeek-AI et al., 2025) selects the top blocks, and the main attention reads their original KV with attention sinks and a local window. Routing needs $b\times$ fewer candidates than token-level indexing in DSA, and unlike CSA (DeepSeek-AI et al., 2026), the main attention never reads compressed text.

Because both layer types reuse existing Summaries, neither adds tokens nor changes the Summary layout, so both can be retrofitted into a pretrained hybrid. The adapted 1.9B model largely preserves Hybrid KSA’s retrieval and general capabilities while outperforming Full on retrieval. Ablations show that C-KSA alone matches its parent and that sparse adaptation incurs a smaller retrieval loss on the compressed backbone than on the original (Section 3.3). At 128K, the model reduces resident cache size by 40.4% and achieves $1.51\times$ prefill and $1.27\times$ decode throughput relative to Hybrid KSA. Compared with Full, it reduces cache size by 73.7% and achieves $2.16\times$ prefill throughput (Section 3.6).

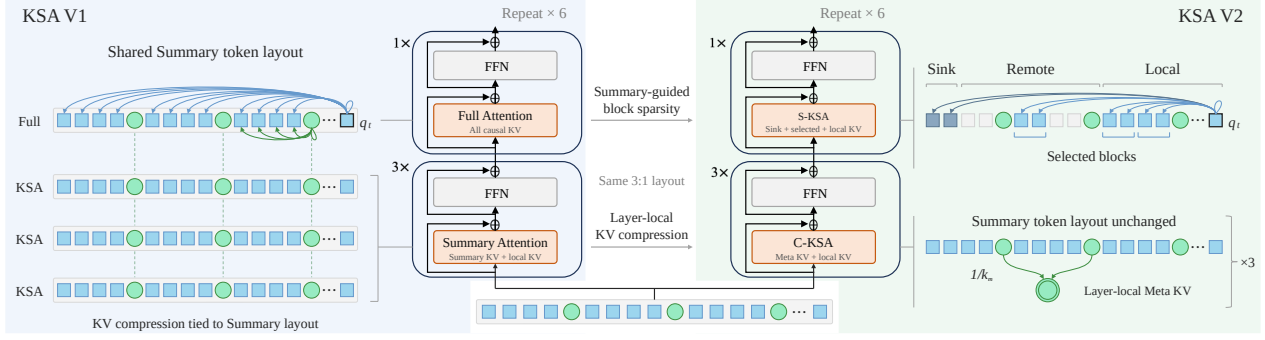


Figure 2 | Architecture overview: from Hybrid KSA to KSA V2.

The remaining cache is mostly original-text KV in the exact-access layers, and aggregation-heavy tasks such as frequent-word extraction degrade under block selection (Table 7).

The approach also retains near-parent retrieval performance on a 4B GQA backbone, supporting transfer beyond the 1.9B MHA setting.

Our contributions are threefold:

- We propose C-KSA, which compresses Summary KV at storage time into layer-local Meta entries, so that further compression of distant history no longer requires coarser Summaries.
- We propose S-KSA, which turns existing Summaries into block-level routing keys for sparse attention over selected original-text blocks.
- We provide a retrofit recipe and GPU kernels that convert a pretrained Hybrid KSA model with RULER performance comparable to its parent, and we report where the remaining cost lies and where the approach fails.

2. Methodology

KSA V2 uses the Summaries in two roles: C-KSA compresses them further as long-range memory, and S-KSA uses them as an index to select original-text blocks for sparse attention. As shown in Figure 2, we retain Hybrid KSA’s 3:1 layout, repeating three C-KSA layers followed by one S-KSA layer throughout the backbone.

We denote text tokens by T_i , chunk Summaries by S_c , and Meta KV entries by M_g . The chunk size k_s specifies the number of text tokens per Summary, k_m the number of Summaries per Meta entry, and b the number of text tokens per routing block. For a text query at position t , q_t and O_t denote the attention query and output, respectively.

2.1. Compressed KSA (C-KSA)

C-KSA adds a second level of compression to KSA: KSA inserts one Summary token after every k_s text tokens, and C-KSA aggregates every k_m consecutive Summary KV entries into one Meta KV entry. This increases the compression factor for distant history from k_s to $k_s k_m$, while keeping recent text directly accessible (Figure 3).

Text-to-Summary aggregation. As in KSA, we partition the text sequence into chunks of k_s tokens and append a Summary token S_c to each chunk c . All Summary tokens share a learned initial

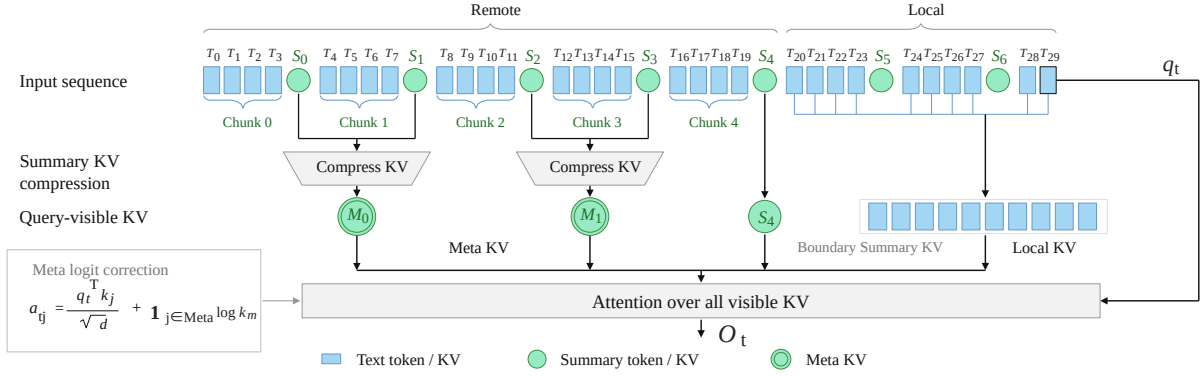


Figure 3 | Compressed KSA with $k_s = 4$ and $k_m = 2$. Complete remote Summary groups are compressed into Meta KV, while boundary Summary KV and local text KV remain directly accessible.

embedding and use the same attention and feed-forward parameters as text tokens. At each layer, S_c attends to its chunk’s text tokens and itself to aggregate chunk-level information. Text tokens retain their original positional indices, and S_c uses the position of its chunk’s last text token.

Summary-to-Meta aggregation. In each C-KSA layer, we divide the Summary sequence into non-overlapping groups of k_m consecutive Summaries. Let d denote the hidden size and d_{KV} the total key/value width across heads. For a group g , let $h_r \in \mathbb{R}^d$ denote the normalized layer input of its r -th Summary, where $0 \leq r < k_m$. We compute its key, value, and aggregation logits as

$$\begin{aligned} k_r &= \text{RMSNorm}(W_K h_r), & v_r &= W_V h_r, \\ z_r &= A h_r. \end{aligned} \quad (1)$$

Here, $W_K, W_V, A \in \mathbb{R}^{d_{KV} \times d}$; W_K and W_V are the backbone’s existing projections, and A is an additional learned projection. RMSNorm is applied independently to each attention head. We add a learned within-group position bias $B_r \in \mathbb{R}^{d_{KV}}$ and normalize the logits across the k_m Summaries to obtain weight vectors $\alpha_r \in \mathbb{R}^{d_{KV}}$:

$$[\alpha_0, \dots, \alpha_{k_m-1}] = \text{softmax}_{\text{group}}([z_0 + B_0, \dots, z_{k_m-1} + B_{k_m-1}]). \quad (2)$$

The softmax operates independently for each feature coordinate. The same weights aggregate the keys and values:

$$\bar{k} = \sum_{r=0}^{k_m-1} \alpha_r \odot k_r, \quad \bar{v} = \sum_{r=0}^{k_m-1} \alpha_r \odot v_r, \quad (3)$$

where $\odot$ denotes element-wise multiplication. Finally, we apply rotary position embedding (RoPE) (Su et al., 2024) to the pooled key at position p_g of the group’s last Summary, producing the Meta KV entry

$$M_g = (\text{RoPE}(\bar{k}, p_g), \bar{v}). \quad (4)$$

The aggregation parameters A and $\{B_r\}_{r=0}^{k_m-1}$ are specific to each layer and shared across groups. Meta entries are used only within the layer that constructs them, while the original Summary states continue to propagate across layers. The group size k_m can therefore vary across C-KSA layers without changing the shared Summary token layout. We omit the layer index for clarity.

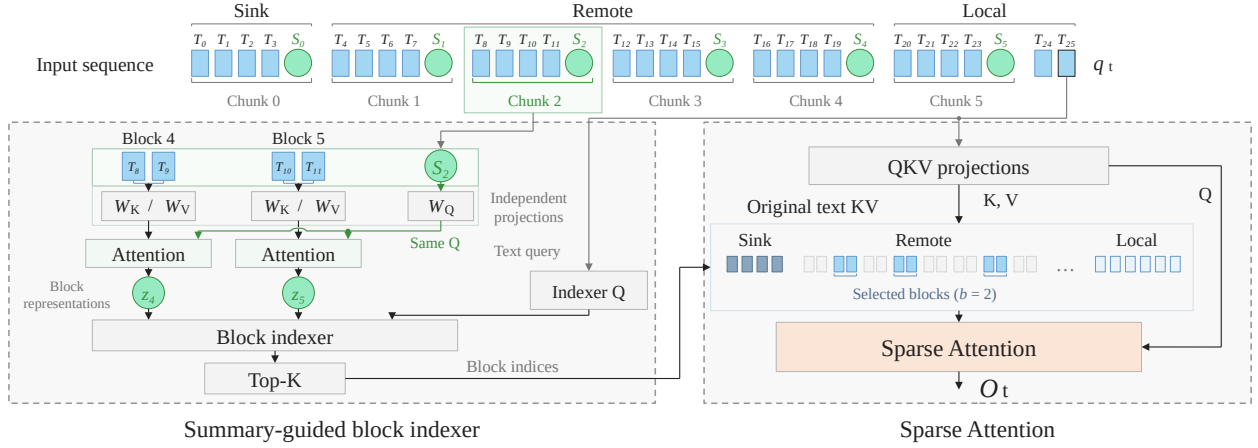


Figure 4 | Sparse KSA with $k_s = 4$ and $b = 2$.

Visible KV entries. For each text query, C-KSA attends to three types of KV entries: local text, boundary Summaries, and remote Meta entries (Figure 3). The local window contains the causal prefix of the current chunk and up to W preceding complete chunks. Outside this window, each complete group of k_m Summaries is represented by one Meta KV entry. The remaining fewer than k_m Summaries immediately preceding the window remain individually accessible until their entire group lies outside the window. All three types of entries participate in a single attention softmax.

Attention-mass correction. Compressing remote Summary entries can shift attention toward the local window by reducing the number of terms in the softmax denominator, even if local logits remain unchanged. We therefore add $\log k_m$ to each Meta logit s before normalization. Since $\exp(s + \log k_m) = k_m \exp(s)$, this helps preserve the balance between local and remote attention.

2.2. Sparse KSA (S-KSA)

S-KSA replaces full attention with Summary-guided block selection and sparse reads of original text (Figure 4). It uses the Summary tokens already present in KSA to construct one index key for each text block. For each query, a Lightning Indexer selects relevant remote blocks, and the main attention reads their original KV together with local context. This design reduces the routing overhead of token-level selection and avoids dense attention over the entire text history, while preserving direct access to the selected blocks' original tokens.

Summary-guided block aggregation. We divide each chunk into blocks of b consecutive text tokens, where b divides k_s . For a block u , let $h_1, \dots, h_b \in \mathbb{R}^d$ denote its normalized layer inputs and $h_s \in \mathbb{R}^d$ the normalized input of its chunk's Summary. For one auxiliary attention head of width d_A , omitting the head index, we compute

$$\begin{aligned} q_s &= \text{RoPE}\left(\text{RMSNorm}(W_Q^A h_s), p_s\right), \\ k_i &= \text{RoPE}\left(\text{RMSNorm}(W_K^A h_i), p_i\right), \\ v_i &= W_V^A h_i. \end{aligned} \tag{5}$$

Here, $W_Q^A, W_K^A, W_V^A \in \mathbb{R}^{d_A \times d}$ are auxiliary projections specific to each S-KSA layer and independent of the backbone's attention projections. The positions p_s and p_i follow the original sequence positions,

with the Summary sharing its chunk’s last text position.

Writing $K^A = [k_1, \dots, k_b]$ and $V^A = [v_1, \dots, v_b]$, the head’s block representation is

$$\widehat{z}_u = V^A \text{softmax}\left(\frac{(K^A)^\top q_s}{\sqrt{d_A}}\right). \quad (6)$$

The softmax operates over the block’s b text tokens; the Summary supplies only the query. Concatenating the outputs of the H_A auxiliary heads gives $z_u \in \mathbb{R}^{H_A d_A}$. Blocks in a chunk share the Summary query but aggregate their own text tokens. This resembles the landmark pooling of HiLS-Attention (Hu et al., 2026), but reuses the existing Summary as the query rather than inserting additional landmark tokens.

Block indexing. We adapt the Lightning Indexer (DeepSeek-AI et al., 2025) to score text blocks using their Summary-guided representations. For a text query at position t , the score for block u is

$$\rho_u = \sum_{h=1}^{H_I} w_h \text{ReLU}\left(\frac{(q_h^I)^\top k_u^I}{\sqrt{d_I}}\right). \quad (7)$$

The Indexer uses H_I query heads of dimension d_I . For each text query, its normalized layer input h_t is mapped to a query vector $q_h^I \in \mathbb{R}^{d_I}$ and a scalar weight w_h for each head h . Each block representation z_u is mapped to a single index key $k_u^I \in \mathbb{R}^{d_I}$, shared across all heads. The weights w_h , scaled by $1/\sqrt{H_I}$, combine the per-head relevance scores into the block score ρ_u . For the 4B model, we additionally apply RMSNorm to the projected index queries and keys before RoPE to improve adaptation convergence. Section 3.4 discusses this choice. Before scoring, partial RoPE encodes queries at position t and keys at their blocks’ last text positions.

Sparse access to original text. Given the block scores $\{\rho_u\}$, S-KSA selects up to K/b remote blocks, where K denotes the remote budget measured in original-text tokens. Candidates must belong to completed chunks and lie entirely before the causal local window. Eligible sink blocks at the beginning of the sequence are retained within this budget, and the remaining slots are filled by the highest-scoring candidates. Let $\mathcal{L}_t$ and $\mathcal{R}_t$ denote local and selected remote positions, respectively. The main attention computes

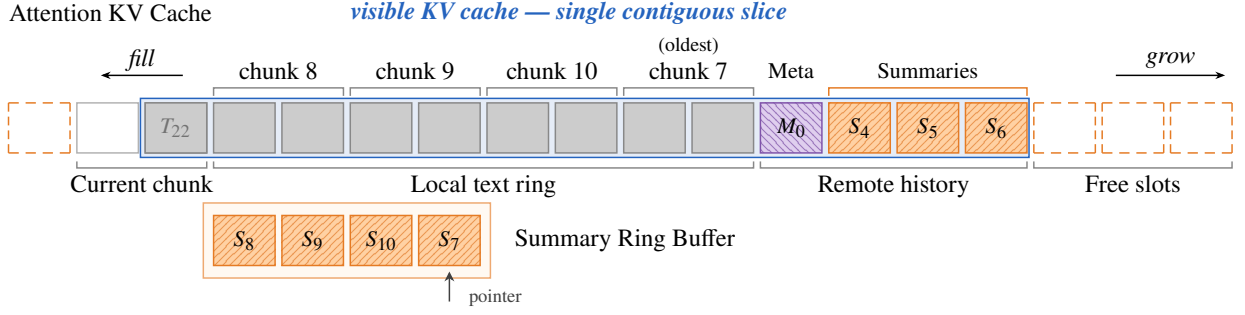
$$O_t = \text{Attn}(q_t, [K_{\mathcal{L}_t}, K_{\mathcal{R}_t}], [V_{\mathcal{L}_t}, V_{\mathcal{R}_t}]), \quad (8)$$

where brackets denote concatenation along the sequence dimension. The attention uses the backbone’s original-text Q/K/V, with a single softmax over local and remote tokens in each head.

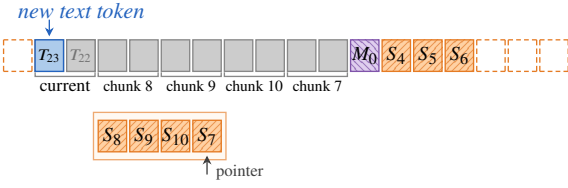
For N text tokens, S-KSA constructs approximately N/b index keys, reducing candidate comparisons by a factor of b relative to token-level indexing. Its routing cost scales as $O(N^2/b)$ during prefill, while the main attention reads a budgeted set of original-text blocks for each query.

2.3. KV Cache for C-KSA

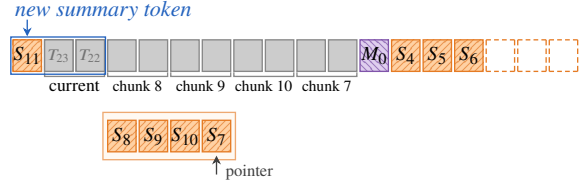
The C-KSA KV cache combines a contiguous attention region with a Summary ring buffer to support incremental decoding and second-level compression. We first describe its layout and update rules, then analyze its memory cost and how sequence compression combines with GQA and MLA.



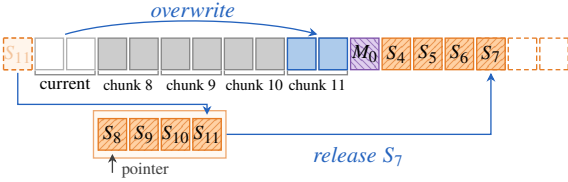
(a) Cache layout. Text queries read one contiguous KV slice (blue outline). The Summary Ring Buffer stores local Summary KV and compression state.



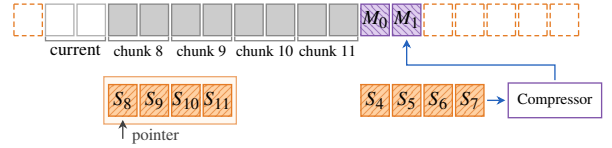
(b) Text insertion. Writing T_{23} immediately left of T_{22} completes chunk 11.



(c) Summary insertion. S_{11} attends only to T_{22} , T_{23} , and its own KV.



(d) Ring update. Chunk 11 replaces chunk 7, moving S_7 to remote history. S_{11} takes its slot, and the pointer advances.



(e) Meta compression. S_4 – S_7 become M_1 , which overwrites the group's first slot and frees the other three.

Figure 5 | C-KSA cache layout and decoding updates with $k_s = 2$, $W = 4$, and $k_m = 4$. Gray (T), orange (S), and purple (M) denote text, Summary, and Meta KV. Blue marks indicate reads or writes.

2.3.1. Cache Structure and Updates

To support efficient decoding, we design a compact KV cache structure for C-KSA (Figure 5). This structure allows each text query to read all visible KV entries through a single contiguous slice, avoiding per-step concatenation of cache segments. Each C-KSA layer maintains two components: an *Attention KV Cache* and a *Summary Ring Buffer*. The illustration uses $k_s = 2$, $W = 4$, and $k_m = 4$, with T_i , S_c , and M_g denoting text, Summary, and Meta KV entries, respectively.

Cache layout. The Attention KV Cache stores keys and values in separate tensors with the same layout: the current chunk, a local text ring, and remote history, followed by free slots (Figure 5a). The current chunk reserves k_s text slots, and the local ring reserves $k_s W$ slots for W preceding complete chunks. Remote history contains Meta entries followed by fewer than k_m Summaries awaiting compression. The visible KV entries are contiguous along the sequence dimension, allowing each text query to read them with a single slicing operation. The Summary Ring Buffer stores the local chunks' Summary KV and auxiliary compression state. While a chunk remains in the local window, text queries attend to its original-text KV rather than its Summary KV. When the chunk leaves the window, its Summary KV enters remote history. A shared pointer identifies the next chunk slot to replace in both rings.

Table 1 | KV-cache cost comparison for a sequence of N text tokens. Example settings use $k_s = 8$, $k_m = 128$, $H = 128$, $d_h = 128$, $H_{KV} = 8$, $d_c = 512$, and $d_r = 64$. Compression rates are relative to MHA.

Mechanism	Effective Context	KV Cache Size ($N \rightarrow \infty$)	Compression Rate
MHA	N (exactly)	$2 \cdot N \cdot H \cdot d_h$	—
GQA	N (exactly)	$2 \cdot N \cdot H_{KV} \cdot d_h$	$\frac{H_{KV}}{H} \approx 6.25\%$
MLA	N (exactly)	$N \cdot (d_c + d_r)$	$\frac{d_c + d_r}{2Hd_h} \approx 1.76\%$
GDN	N (ambiguous)	$H \cdot d_h^2$	$\frac{d_h}{2N} \approx 0\%$
SWA	w (exactly)	$2 \cdot w \cdot H_{KV} \cdot d_h$	$\frac{w}{N} \cdot \frac{H_{KV}}{H} \approx 0\%$
KSA	N (summarized)	$2 \cdot \frac{N}{k_s} \cdot H \cdot d_h$	$\frac{1}{k_s} \approx 12.5\%$
KSA with GQA	N (summarized)	$2 \cdot \frac{N}{k_s} \cdot H_{KV} \cdot d_h$	$\frac{1}{k_s} \cdot \frac{H_{KV}}{H} \approx 0.78\%$
KSA with MLA	N (summarized)	$\frac{N}{k_s} \cdot (d_c + d_r)$	$\frac{1}{k_s} \cdot \frac{d_c + d_r}{2Hd_h} \approx 0.22\%$
C-KSA	N (summarized)	$2 \cdot \frac{N}{k_s k_m} \cdot H \cdot d_h$	$\frac{1}{k_s k_m} \approx 0.098\%$
C-KSA with GQA	N (summarized)	$2 \cdot \frac{N}{k_s k_m} \cdot H_{KV} \cdot d_h$	$\frac{1}{k_s k_m} \cdot \frac{H_{KV}}{H} \approx 0.0061\%$
C-KSA with MLA	N (summarized)	$\frac{N}{k_s k_m} \cdot (d_c + d_r)$	$\frac{1}{k_s k_m} \cdot \frac{d_c + d_r}{2Hd_h} \approx 0.0017\%$

Text and Summary token insertion. New text KV entries are written from right to left in the current-chunk region, keeping the filled portion adjacent to the local text ring. Each text query attends to the visible KV, including its own entry. At a chunk boundary, the Summary query attends only to the completed chunk’s text KV and its own KV. In the example, T_{23} completes chunk 11 (Figure 5b), and S_{11} attends to T_{22} , T_{23} , and itself (Figure 5c).

Updating the text and Summary rings. Once text and Summary attention finish, the completed chunk’s text KV is copied into the local text ring, and its Summary KV and auxiliary compression state are stored in the Summary Ring Buffer. If the rings are not yet full, the new entries occupy unused slots. Otherwise, they replace the oldest local chunk and its Summary record; the old Summary KV moves into remote history, with its auxiliary state retained for compression. The shared pointer advances to the next chunk slot. In Figure 5d, chunk 11 replaces chunk 7: S_7 enters remote history, S_{11} takes its Summary ring slot, and the pointer advances to the slot holding S_8 .

Compressing remote Summaries. When k_m uncompressed Summaries accumulate in remote history, they are compressed into a single Meta KV entry. This entry overwrites the group’s first slot, leaving the remaining $k_m - 1$ slots available for reuse. In Figure 5e, S_4, S_5, S_6, S_7 are compressed into M_1 , stored immediately after M_0 . The visible KV entries remain contiguous, allowing each text query to read them through a single slice.

2.3.2. KV Cache Memory Analysis

We extend the KV-cache analysis of KSA with C-KSA’s second compression level. Table 1 retains the original mechanisms for comparison. Table 1 counts stored scalars per layer and request, excluding allocation headroom and auxiliary buffers.

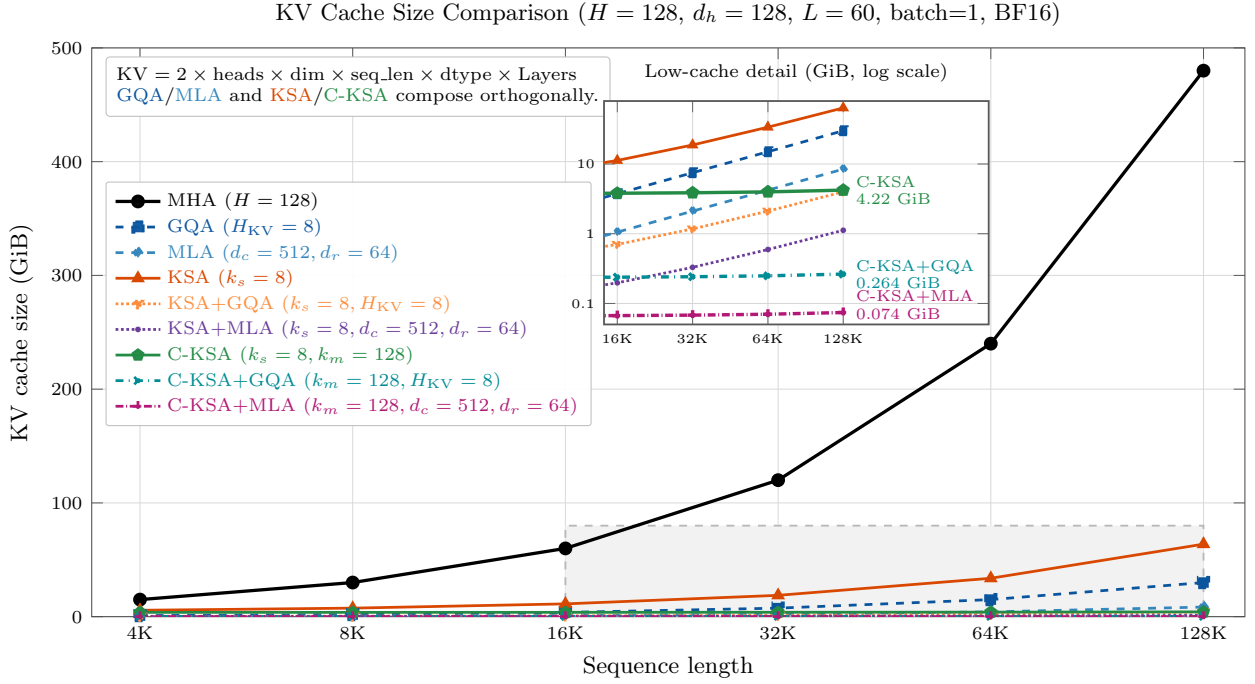


Figure 6 | KV cache size comparison across different mechanisms as sequence length grows. The inset uses a logarithmic vertical axis to separate the low-cache curves and labels C-KSA values at 128K.

KV Cache Cost. Let N denote text length, H and H_{KV} the query and KV head counts, and d_h the head width. In Table 1, d_c and d_r are MLA’s latent and rotary-key widths, and w is the local text window.

KSA’s leading cache term is $2NH_{KV}d_h/k_s$. Under the compression rule in Section 2.1, C-KSA replaces it with

$$M_{\text{C-KSA}} = 2H_{KV}d_h \left[\frac{N}{k_s k_m} + O(k_s W + W + k_m) \right]. \quad (9)$$

The bounded term accounts for local text, recent Summaries, and unfinished compression groups. Thus C-KSA reduces the growing history term by a further factor of k_m while preserving the local window. With $k_s = 8$ and $k_m = 128$, the growing history term is $1/1024$ of uncompressed attention with the same KV width. This nominal $1024\times$ compression applies to distant history; the local and auxiliary terms remain. For fixed compression factors, cache growth remains linear in N .

Orthogonality of KV Cache Compression. C-KSA preserves KSA’s orthogonality to per-position compression: the additional k_m factor multiplies the sequence compression without changing the KV head count or representation width. The resulting GQA and MLA combinations are listed in Table 1.

Figure 6 uses the settings in Table 1 with 60 layers, batch size 1, BF16, and $w = k_s W = 1024$ local tokens. The KSA and C-KSA curves use $N/k_s + w$ and $N/(k_s k_m) + w$ entries, respectively, omitting other bounded auxiliary buffers and allocation headroom.

Hybrid KSA and C-KSA. With matched layer counts and KV widths and a fraction f of Full layers, the leading cache ratio is

$$\frac{M_{\text{C-KSA+Full}}}{M_{\text{Hybrid KSA}}} \simeq \frac{f + (1-f)/(k_s k_m)}{f + (1-f)/k_s}. \quad (10)$$

With $f = 1/4$ in the 3:1 layout, the Full layers limit the whole-model savings.

2.4. Kernel Design

C-KSA and S-KSA reduce attention computation and KV reads through compression and sparse access. Realizing these savings on GPUs requires efficient KV addressing, block selection, and attention execution. Figure 7 summarizes the method-specific addressing and routing alongside the tiled prefill pattern used by both variants.

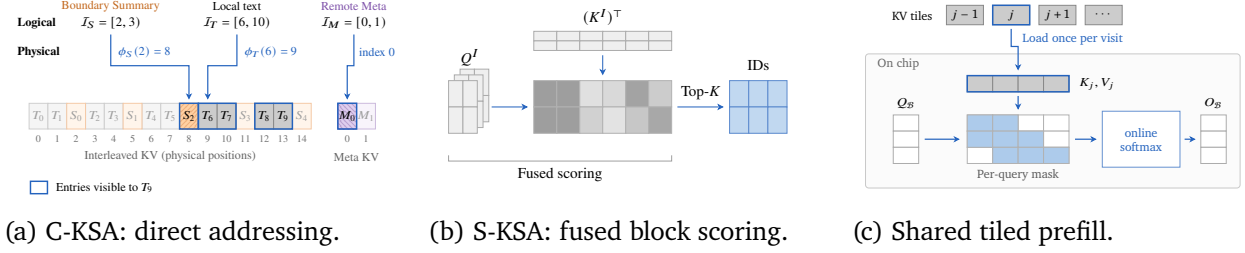


Figure 7 | Kernel design for C-KSA and S-KSA: (a) direct KV addressing, (b) fused block scoring and selection under a K -token budget, and (c) shared tiled prefill with KV reuse.

Shared prefill execution. Both C-KSA and S-KSA follow FlashAttention’s tiled execution and online softmax (Dao et al., 2022), as illustrated in Figure 7c. Let $\mathcal{B}$ denote a group of text queries arranged into one or more query tiles, with query and output matrices $Q_{\mathcal{B}}$ and $O_{\mathcal{B}}$. Each box in these matrices represents one token vector; blue cells in the query–key grid indicate visible pairs, while white cells are masked. Each loaded KV tile (K_j, V_j) is reused on chip within the group, while a per-query mask enforces each query’s visible set. Building on this shared execution pattern, C-KSA and S-KSA specialize the KV addressing and tile schedule described below.

2.4.1. C-KSA

Logical intervals and direct addressing. C-KSA’s regular layout allows the kernel to compute the KV locations needed by each query using simple arithmetic. During prefill, text and Summary Q/K/V are interleaved, while precomputed Meta K/V are stored in a separate buffer. We treat text, Summary, and Meta entries as three logical segments, each with its own zero-based index space.

Figure 7a illustrates this addressing process with $k_s = 2$, $k_m = 2$, and $W = 1$. Blue outlines mark entries visible to T_9 . For a text query at position t , let $a = \max(0, \lfloor t/k_s \rfloor - W)$ denote the first chunk in its local window and $g = \lfloor a/k_m \rfloor$ the number of complete Meta groups preceding that window. We first determine the visible logical indices in each segment:

$$I_T(t) = [k_s a, t + 1), \quad I_S(t) = [k_m g, a), \quad I_M(t) = [0, g). \quad (11)$$

These intervals contain the causal local text, boundary Summaries, and complete remote Meta groups, respectively.

We then map these logical indices to physical positions in the interleaved storage. For text index i and Summary index j , the mappings are

$$\phi_T(i) = i + \left\lfloor \frac{i}{k_s} \right\rfloor, \quad \phi_S(j) = j(k_s + 1) + k_s. \quad (12)$$

The kernel applies these mappings when loading Q/K/V. Meta indices directly address entries in the separate Meta buffer. For each query tile, the logical intervals determine the KV tile ranges to visit,

and a position-based mask excludes entries outside each query’s visible set. The ranges and masks are computed within the kernel, with no dense attention mask or block-index table stored in memory.

Prefill integration. The text-query kernel carries each query’s online-softmax state across the text, Summary, and Meta segments, applying the $\log k_m$ correction only to Meta logits. A separate local kernel computes each Summary query over its own text chunk and self-KV. Both kernels write outputs directly to their interleaved positions.

Decoding. The KV cache in Section 2.3.1 stores the visible KV entries in a contiguous region, allowing attention to be computed directly with scaled dot-product attention (SDPA).

2.4.2. S-KSA

S-KSA uses fused block scoring and a sparse KV-tile schedule to support query-dependent access to original text. The scoring optimization also applies during decoding.

Fused block scoring. Using separate kernels for dot products and head aggregation requires storing intermediate scores in GPU memory. To reduce this memory traffic, we compute block scores with a fused kernel that performs dot products and combines weighted head scores on chip. In Figure 7b, Q^I denotes the stacked query matrices of the indexer heads and K^I their shared block keys. The superscript I denotes the indexer. Each Q^I matrix has queries along rows and features along columns. The central grayscale matrix has queries along rows and candidate blocks along columns, with darker cells indicating higher scores. A separate selection step then chooses remote blocks independently for each query under a K -token budget (up to K/b blocks); each row of the blue IDs matrix on the right lists the block indices selected for one query.

KV-tile scheduling. For each query group, we map its selected remote blocks and local windows to KV tiles and construct a conservative schedule of tiles to visit. The schedule skips inactive KV tiles, while per-query masks enforce block selection, local visibility, and causality within the scheduled tiles. We supply this schedule and the masks to the FlashAttention forward kernel for tiled prefill execution.

Decoding. Gathering selected KV entries into temporary buffers incurs extra memory reads and writes before attention computation. To avoid this overhead during decoding, we use a custom sparse kernel that reads directly from the KV cache using the selected block indices.

3. Experiments

Our experiments examine whether continued pretraining can adapt Hybrid KSA to more aggressive KV compression and sparse original-text access while retaining its capabilities. We first evaluate C-KSA with Full attention, then add S-KSA to the compressed backbone, assessing long-context retrieval, general capabilities, and inference efficiency. We further analyze the effects of compression granularity, block size, teacher pooling, and routing on model performance. Unless otherwise stated, all experiments and analyses use the 1.9B model.

3.1. Experimental Setup

Pretrained model and variants. We start from the 128K checkpoints of the 1.9B and 4B Hybrid KSA models in the KSA report (Chu et al., 2026). The 1.9B backbone has 24 multi-head attention (MHA) layers; the 4B backbone has 36 grouped-query attention (GQA) layers with 32 query heads and eight KV heads. Both interleave three KSA layers with one Full layer and insert one Summary after every $k_s = 8$ text tokens. We evaluate two successive variants: **C-KSA+Full** compresses remote Summary KV in KSA layers; **C-KSA+S-KSA** further replaces the Full layers with S-KSA. Although C-KSA supports layer-specific compression, each evaluated configuration uses a common k_m across its C-KSA layers. Unless otherwise specified, all experiments use the backbone and module configurations in Table 2.

Baselines and training data. We reuse the KSA report’s from-scratch results at 1.9B and continual-pretraining results at 4B. The baselines include Full, Hybrid SWA, Hybrid SCA, Pure KSA, and Hybrid KSA at both scales, plus Hybrid GDN at 1.9B and Hybrid Linear at 4B. Hybrid KSA serves as the parent-model reference for capability retention, while Full provides a dense-attention reference. We report module-only and full-parameter training budgets separately. A continuation study found no consistent RULER improvement for 1.9B Hybrid KSA under the tested recipe and budgets (Table 11), supporting our use of its original checkpoint as the capability reference. Adaptation reuses KSA’s training corpus and data mixture, with data sources and mixture proportions detailed in the original report. We use a fixed context length of 128K.

Evaluation. Following the KSA report, we assess long-context modeling and general capabilities. Long-context modeling is evaluated using RULER (Hsieh et al., 2024) at sequence lengths from 4K to 128K. General capabilities cover knowledge (MMLU, CMMLU, C-Eval, and MMLU-Pro (Hendrycks et al., 2021a; Huang et al., 2023; Li et al., 2024a; Wang et al., 2024)), mathematics (GSM8K, CMATH, and MATH (Cobbe et al., 2021; Hendrycks et al., 2021b; Wei et al., 2023)), and code generation (MBPP and HumanEval (Austin et al., 2021; Chen et al., 2021)).

3.2. Adaptation Protocol

Figure 8 summarizes our sequential adaptation route at 1.9B: we first study compression granularity in C-KSA+Full, then compare sparse-attention schemes on the selected $k_m = 128$ checkpoint. Module-only adaptation trains only newly introduced parameters while freezing all inherited parameters; full-parameter adaptation updates the entire model. C-KSA uses module-only adaptation followed by full-parameter adaptation; the subsequent S-KSA stage uses module-only adaptation.

C-KSA adaptation. During module-only adaptation, we train the aggregation projection A and position biases $\{B_r\}$ while freezing all inherited parameters. The new parameters are initialized to zero, yielding uniform aggregation weights. We align compressed attention outputs with those of uncompressed KSA. Let $O_{l,t,d}^C$ and $O_{l,t,d}^K$ denote the d -th coordinate of the attention outputs from C-KSA and uncompressed KSA, respectively, at layer l and text position t . Both outputs are computed from the same layer input, before the output projection. The alignment loss is

$$\mathcal{L}_{\text{align}} = \frac{1}{|\mathcal{I}_C| |\mathcal{T}| D} \sum_{l \in \mathcal{I}_C} \sum_{t \in \mathcal{T}} \sum_{d=1}^D \min \left\{ \left(O_{l,t,d}^C - \text{sg}(O_{l,t,d}^K) \right)^2, 0.1 \right\}, \quad (13)$$

where $\mathcal{I}_C$ is the set of C-KSA layers, $\mathcal{T}$ contains the text positions in a minibatch, D is the concatenated attention width, and $\text{sg}(\cdot)$ denotes stop-gradient. Each squared coordinate error is clipped before

Table 2 | Configurations of the selected 1.9B and 4B models. Module settings apply where the corresponding modules are used.

Configuration	1.9B	4B
Shared backbone		
Number of layers	24	36
Hidden size	2048	2560
Intermediate size	6144	9728
Attention heads (Q/KV)	16/16	32/8
Head dimension	128	128
Hybrid architecture ratio	3:1	3:1
Summary chunk size k_s	8	8
Sliding chunk number (KSA/C-KSA)	128	128
Tied embeddings	False	True
C-KSA		
Summary-to-Meta group size k_m	128	32
KV width d_{KV}	2048	1024
Aggregation projection A shape	2048×2048	1024×2560
Position-bias shape $k_m \times d_{KV}$	128×2048	32×1024
Additional parameters	80.2M	71.7M
S-KSA		
Block size b	4	4
Remote token budget K	4096	4096
Sink tokens, included in K	8	8
Local window length (text tokens)	1024	1024
Aggregation heads (Q/KV)	16/16	32/8
Aggregation head dimension d_A	128	128
Indexer heads	8	8
Indexer head dimension	128	128
Query low-rank dimension	512	512
Final Indexer Q/K RMSNorm	No	Yes
Additional parameters	86.6M	163.0M

averaging. We then transition from module-only to full-parameter adaptation, using the objectives

$$\mathcal{L}_{\text{comp}} = \begin{cases} \mathcal{L}_{\text{CE}} + \lambda_{\text{align}} \mathcal{L}_{\text{align}}, & \text{module-only adaptation,} \\ \mathcal{L}_{\text{CE}}, & \text{full-parameter adaptation,} \end{cases} \quad (14)$$

where $\mathcal{L}_{\text{CE}}$ is the text-token cross-entropy loss and $\lambda_{\text{align}} = 5$.



Figure 8 | Sequential adaptation and parameter selection for KSA V2 at 1.9B.

Table 3 | Adaptation hyperparameters and token budgets for the selected models.

Configuration	C-KSA		S-KSA
	Module-only	Full-parameter	Module-only
Trainable parameters	Aggregation modules	All	Aggregation and Indexer
Training tokens (B), 1.9B	6.7	3.4	18.5
Training tokens (B), 4B	6.7	15.9	4.2
Target learning rate	10^{-4}	10^{-4}	10^{-3}
LR warmup (updates)	200	400	200
Sequence length		128K	
Global batch size		32	
Initial learning rate		10^{-6}	
LR schedule		Warmup then constant	
Optimizer		AdamW	
AdamW (β_1, β_2)		(0.9, 0.95)	
Weight decay		0.01	
Gradient clipping		1.0	

S-KSA adaptation. Module-only adaptation trains the new aggregation modules and Indexer while freezing all inherited parameters, including the C-KSA aggregation parameters. During training, we retain dense attention in Full layers and use its probabilities to supervise block selection. For query t , let $\mathcal{U}_t$ contain all eligible remote blocks, $\mathcal{B}_u$ the text positions in block u , and $a_{t,i}$ the dense causal attention probability averaged across backbone heads. Suppressing the layer index, we define the block-level teacher distribution p_t , Indexer distribution π_t , and training objective as

$$\begin{aligned}
 p_{t,u} &= \text{sg} \left(\frac{\max_{i \in \mathcal{B}_u} a_{t,i}}{\sum_{v \in \mathcal{U}_t} \max_{j \in \mathcal{B}_v} a_{t,j}} \right), \\
 \pi_t &= \text{softmax}(\rho_t), \\
 \mathcal{L}_{\text{S-KSA}} &= \mathbb{E}_{l,t} [D_{\text{KL}}(p_t \parallel \pi_t)],
 \end{aligned} \tag{15}$$

where $\text{sg}(\cdot)$ denotes stop-gradient and ρ_t contains the Indexer scores defined in Equation 7 for all blocks in $\mathcal{U}_t$. The loss is averaged over valid text queries within each adapted layer, then across layers. Following QSA (Qiu et al., 2026), the teacher takes the blockwise maximum, after averaging over heads; we compare max and sum pooling in Section 3.7. At inference, the trained Indexer selects remote blocks for sparse attention.

Training configuration. Table 3 summarizes our adaptation hyperparameters. We reinitialize the optimizer and learning-rate schedule at each stage transition.

3.3. Model Quality

Table 4 compares C-KSA+Full and C-KSA+S-KSA with the 1.9B baselines from the KSA report.

Long-context retrieval. C-KSA+S-KSA achieves a six-length RULER mean of 72.39% and a 128K score of 65.01%, compared with 72.61% and 65.35% for Hybrid KSA and 67.55% and 48.75% for Full.

Table 4 | Long-context retrieval and general capabilities at 1.9B (%; pass@1 for code). Bold and underlined values mark the best and second-best scores, respectively, here and in subsequent tables.

Benchmark	Baselines						Ours	
	Full	Hybrid SWA	Hybrid SCA	Hybrid GDN	Pure KSA	Hybrid KSA	C-KSA+Full	C-KSA+S-KSA
Long-context retrieval								
RULER-4K	76.08	74.54	77.72	79.83	70.44	80.65	81.52	<u>81.48</u>
RULER-8K	72.85	71.69	<u>75.22</u>	76.01	65.91	73.35	72.24	72.42
RULER-16K	73.24	69.54	72.55	74.04	66.74	74.07	<u>74.44</u>	75.67
RULER-32K	69.06	67.86	67.74	70.41	62.54	72.30	<u>71.93</u>	70.78
RULER-64K	65.32	63.03	63.54	69.39	57.13	69.95	<u>69.61</u>	68.99
RULER-128K	48.75	56.64	58.01	59.87	39.29	<u>65.35</u>	66.31	65.01
RULER mean	67.55	67.22	69.13	71.59	60.34	<u>72.61</u>	72.68	72.39
General knowledge								
MMLU	44.99	46.84	46.77	46.23	46.83	46.83	<u>46.97</u>	47.09
CMMLU	44.41	45.89	46.42	47.19	45.59	46.88	46.49	46.39
C-Eval	44.28	43.54	47.62	45.54	45.69	44.13	43.83	42.94
MMLU-Pro	19.48	20.46	20.10	21.22	21.72	22.52	<u>22.71</u>	22.83
Mathematics								
GSM8K	48.29	47.46	52.39	50.95	54.81	59.14	55.95	<u>56.33</u>
CMATH	55.33	54.83	62.33	58.00	58.50	61.83	64.50	<u>63.50</u>
MATH	23.38	31.46	28.82	33.30	30.04	36.92	36.60	<u>36.72</u>
Code								
MBPP	30.60	30.00	31.60	34.80	<u>35.80</u>	36.40	35.60	35.20
HumanEval	25.61	28.05	26.83	27.44	<u>29.88</u>	31.71	28.05	<u>29.88</u>

The combined model thus recovers near-parent retrieval performance while retaining the advantage of Hybrid KSA over Full.

Compression and sparsity compose. At $k_m = 128$, C-KSA+Full increases the distant-history compression factor within KSA layers from 8 to 1024, with mean and 128K RULER scores of 72.68% and 66.31%. Adding S-KSA to this backbone lowers the 128K score to 65.01%, whereas the same sparse adaptation applied to Hybrid KSA lowers it from 65.35% to 61.06% (Table 9(c)).

General capabilities. C-KSA+S-KSA remains broadly comparable to Hybrid KSA across the nine general-capability benchmarks, with gains on MMLU, MMLU-Pro, and CMATH and lower scores on the remaining tasks. The largest decrease is 2.81 percentage points on GSM8K, while MBPP and HumanEval decrease by 1.20 and 1.83 points, respectively. Compared with C-KSA+Full, adding S-KSA changes each score by at most 1.83 points, with both gains and losses, indicating no uniform decline from sparse adaptation. The combined model exceeds Full on eight of the nine benchmarks, with C-Eval as the exception.

Adaptation dynamics. Figure 9 shows how retrieval performance recovers during adaptation. For C-KSA, retrieval performance improves during module-only adaptation and approaches Hybrid KSA after full-parameter adaptation. On the selected C-KSA+Full backbone, S-KSA with $b = 4$ approaches Hybrid KSA’s performance by training only its newly introduced aggregation modules and Indexer.

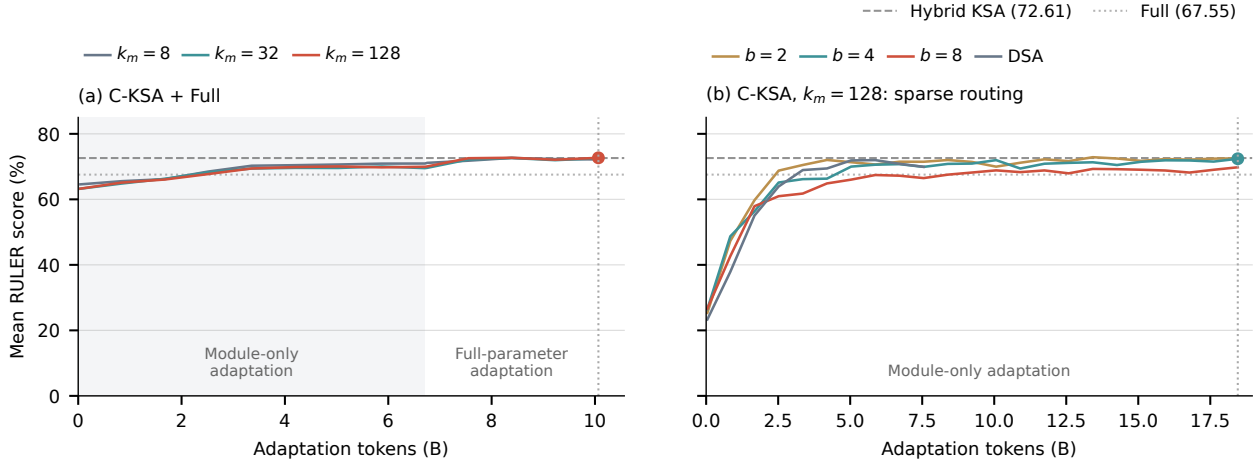


Figure 9 | Mean RULER scores during compression and sparse adaptation. Circles mark the checkpoints reported in Table 4.

3.4. Scaling to a 4B GQA Model

To test transfer beyond the 1.9B MHA backbone, we adapt the 4B GQA model with $k_m = 32$ and $b = 4$. With $k_s = 8$, C-KSA compresses distant-history KV by $256\times$ along the sequence dimension while preserving GQA’s shared KV heads. The C-KSA+Full checkpoint uses 6.7B tokens of module-only adaptation followed by 15.9B tokens of full-parameter adaptation. Starting from this checkpoint, S-KSA uses a further 4.2B tokens of module-only adaptation, leaving all inherited parameters frozen. The remote budget, local window, and sink settings remain unchanged from the 1.9B experiments; the 4B Indexer additionally normalizes its projected queries and keys as described in Section 2.2.

Table 5 compares the adapted models with the six 4B baselines reported in KSA. C-KSA+S-KSA achieves a six-length RULER mean of 84.15%, compared with 84.45% for Hybrid KSA and 83.72% for its C-KSA+Full starting point. At 128K, it scores 74.41%, compared with 71.67% for Hybrid KSA and 65.86% for Full. On general-capability benchmarks, C-KSA+S-KSA stays within 0.40 percentage points of C-KSA+Full and 0.50–2.58 points below Hybrid KSA. Overall, C-KSA+S-KSA retains retrieval performance close to Hybrid KSA, with modest declines in general capabilities. These results support its effectiveness at a larger scale and compatibility with GQA.

Indexer Q/K normalization. Without Q/K normalization, the 4B Indexer loss remains high, and RULER performance stays below its C-KSA+Full parent even after 21.8B adaptation tokens. Under the same adaptation setup, normalizing the projected queries and keys restores near-parent performance with only 4.2B tokens. A possible explanation is that differences in feature scale between the pretrained backbones affect index-logit magnitudes and optimization; RMSNorm reduces this scale sensitivity. We retain the original 1.9B setup for consistency and have not repeated its full experimental suite with this change. Whether normalization also reduces the adaptation data required at 1.9B remains an open question for future work.

Compression granularity. Table 6 compares compression granularities on the 4B backbone, using 6.7B tokens of module-only adaptation followed by 15.9B tokens of full-parameter adaptation for each configuration. Compared with $k_m = 8$, $k_m = 32$ reduces the number of remote Meta entries by a further factor of four while retaining similar RULER performance. We select $k_m = 32$ for the 4B sparse-adaptation experiments.

Table 5 | Long-context retrieval and general capabilities at 4B (%; pass@1 for code). Baselines are from the KSA report (Chu et al., 2026).

Benchmark	Baselines						Ours	
	Full	Hybrid SWA	Hybrid SCA	Hybrid Linear	Pure KSA	Hybrid KSA	C-KSA+Full	C-KSA+S-KSA
Long-context retrieval								
RULER-4K	92.88	91.30	86.02	86.39	91.55	<u>92.97</u>	92.77	93.38
RULER-8K	91.38	88.03	84.28	83.86	86.78	<u>90.53</u>	88.12	87.78
RULER-16K	89.12	82.87	80.67	78.06	84.78	<u>88.86</u>	86.27	85.45
RULER-32K	<u>84.74</u>	78.94	76.89	76.48	80.30	86.65	82.68	83.60
RULER-64K	<u>78.16</u>	73.88	68.88	73.50	76.09	76.04	<u>78.58</u>	80.28
RULER-128K	65.86	66.27	60.94	67.98	66.81	71.67	<u>73.92</u>	74.41
RULER mean	83.69	80.22	76.28	77.71	81.05	84.45	83.72	<u>84.15</u>
General knowledge								
MMLU	71.83	70.57	69.83	64.33	<u>70.73</u>	70.50	68.79	68.79
CMMLU	75.00	<u>73.69</u>	72.59	68.41	<u>73.29</u>	72.63	70.88	71.06
C-Eval	73.66	<u>72.36</u>	71.66	67.42	72.14	<u>72.66</u>	70.36	70.58
MMLU-Pro	46.36	45.23	45.11	38.83	<u>45.70</u>	45.39	43.66	43.58
Mathematics								
GSM8K	82.75	<u>81.92</u>	80.10	72.44	81.09	79.50	79.00	79.00
CMATH	83.41	84.84	83.16	79.09	<u>84.58</u>	84.25	81.50	81.67
MATH	47.48	48.24	47.45	42.57	<u>48.15</u>	47.56	46.10	46.40
Code								
MBPP	61.30	<u>61.70</u>	59.60	55.30	61.50	62.20	60.40	60.00
HumanEval	58.54	<u>61.89</u>	<u>61.89</u>	54.58	60.97	62.50	60.98	60.98

Table 6 | RULER scores (%) of 4B C-KSA+Full with different compression granularities.

k_m	4K	8K	16K	32K	64K	128K	Mean
8	91.13	88.22	85.71	82.62	81.35	74.58	83.94
16	91.45	<u>89.51</u>	88.29	83.06	75.47	69.56	82.89
32	92.77	88.12	86.27	<u>82.68</u>	78.58	<u>73.92</u>	<u>83.72</u>
64	89.83	87.17	85.62	<u>81.67</u>	<u>79.56</u>	<u>70.93</u>	<u>82.46</u>
128	<u>92.23</u>	89.83	<u>86.36</u>	81.25	<u>74.38</u>	71.90	82.66

3.5. Long-Context Retrieval Analysis

We evaluate whether C-KSA+S-KSA retains the parent model’s ability to retrieve information from different positions in a long context. In the Needle-in-a-Haystack test (Kamradt, 2023), a target item is inserted into the context and retrieved through a targeted question. Figure 10 compares the models on identical inputs across six context lengths from 4K to 128K and eleven insertion depths from 0% to 100%. Table 7 complements the grid with a paired comparison of RULER subtasks at 128K.

Table 7 | RULER-128K subtask scores (%). **C-KSA+S-KSA** exceeds Hybrid KSA on Single, Multiquery, and SQuAD.

Subtask	Hybrid KSA	C-KSA+S-KSA
NIAH-Single	93.33	98.33
NIAH-Multikey	85.00	83.33
NIAH-Multivalue	91.25	88.75
NIAH-Multiquery	91.25	96.25
VT	25.00	23.00
FWE	45.00	26.67
SQuAD	30.00	35.00

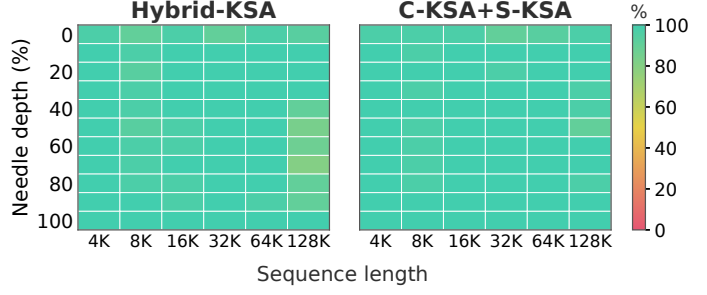


Figure 10 | Single-needle retrieval across context lengths and needle depths. **C-KSA+S-KSA** maintains high accuracy across the grid, averaging **97.88%** over all depths at 128K.

3.6. Inference Efficiency

C-KSA reduces inference costs by compressing remote Summary KV, while S-KSA replaces dense original-text attention with block-sparse access. Figure 1 summarizes the resulting compute and cache costs for C-KSA+S-KSA with $k_m = 128$ and $b = 4$. At 128K, its estimated matrix-FLOP count per text token is lower by a factor of 6.38 relative to Full and 2.76 relative to Hybrid KSA. Its BF16 KV and index-key cache is smaller by factors of 3.86 and 1.35, respectively.

Figure 11 compares the resident cache size, prefill throughput, and decode throughput of Full, Hybrid KSA, and C-KSA+S-KSA from 16K to 128K using the 1.9B models with BF16 and batch size one on a single NVIDIA H800 GPU. C-KSA+S-KSA uses $k_m = 128$ and $b = 4$.

Cache. At 128K, C-KSA+S-KSA’s resident cache occupies **6.31 GiB**, compared with 10.59 GiB for Hybrid KSA and 24.00 GiB for Full, reducing cache size by **40.4%** and **73.7%**, respectively. Measurements include indexer, auxiliary state, KV storage, and resident workspaces.

Prefill. At 128K, C-KSA+S-KSA reaches **44,971 tokens/s**, compared with 29,837 for Hybrid KSA and 20,830 for Full, corresponding to **2.16×** of Full attention. Its gain over Hybrid KSA increases from 1.23× at 16K to 1.51× at 128K.

Decode. C-KSA+S-KSA achieves **215.50 tokens/s** at 16K and **198.85 tokens/s** at 128K, compared with 162.87 and 157.02 for Hybrid KSA, respectively. Across the four sequence lengths, this gives a **1.23–1.32×** speedup.

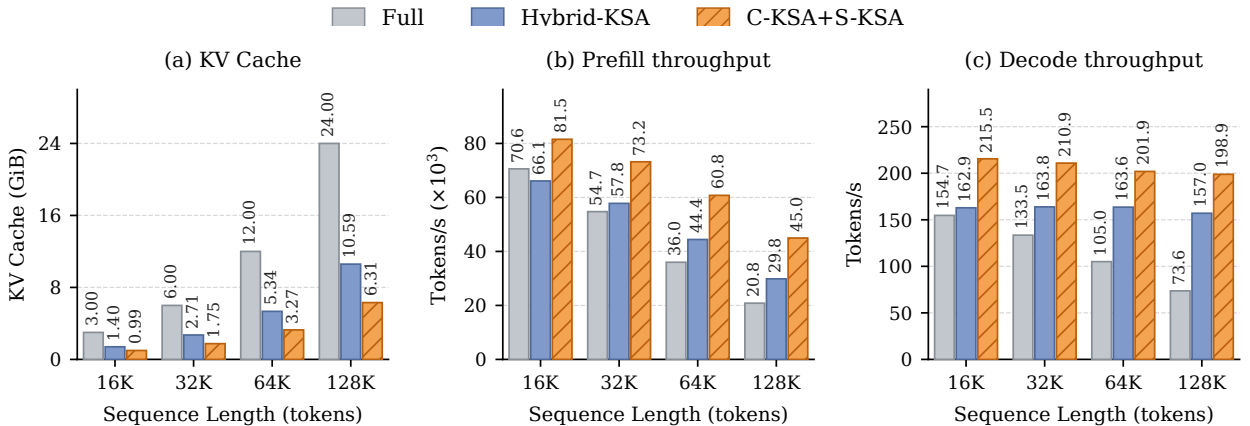


Figure 11 | Inference cache size and throughput of 1.9B Full, Hybrid KSA, and C-KSA+S-KSA ($k_m = 128$, $b = 4$).

3.7. Analysis and Ablation Studies

Direct versus hierarchical compression. An alternative way to obtain $1024\times$ compression of distant-history KV is to increase the Summary chunk size directly to $k_s = 1024$. Table 8 compares this approach with C-KSA+Full, using the original $k_s = 8$ Hybrid KSA as a reference. All three configurations retain Full-attention layers. The large-chunk model is trained from scratch, while C-KSA builds on small-chunk Hybrid KSA.

Table 8 | RULER performance of direct and hierarchical compression with Full attention retained.

Configuration	4K	8K	16K	32K	64K	128K	Mean
Hybrid KSA, $k_s = 8$	<u>80.65</u>	73.35	<u>74.07</u>	72.30	69.95	65.35	<u>72.61</u>
Hybrid KSA, $k_s = 1024$	75.65	72.14	<u>70.44</u>	70.81	69.11	<u>65.62</u>	<u>70.63</u>
C-KSA+Full, $k_s = 8$, $k_m = 128$	81.52	<u>72.24</u>	74.44	<u>71.93</u>	<u>69.61</u>	66.31	72.68

Direct compression achieves a mean RULER score of 70.63, compared with 72.61 for the small-chunk reference and 72.68 for C-KSA+Full. Hierarchical compression thus matches the reference performance and exceeds direct compression by 2.05 percentage points. This finding is consistent with the 4B-model ablation in the KSA report (Chu et al., 2026, Table 7), where increasing k_s from 8 to 64 reduces the mean RULER score over 4K–64K from 82.97 to 77.79. This advantage may stem from the **additional computation devoted to fine-grained Summary processing in C-KSA**, which may help preserve retrieval-relevant information at the same remote-memory compression ratio. Actually, the additional computation as in two aspects: (1) the C-KSA takes 8 as the summary chunk, which will add more tokens at input sequence level; (2) the compressed KV memory is further computed by an additional module, which can be seen as a two-step summarization.

Compression granularity with Full attention. Table 9(a) compares $k_m \in \{4, 8, 16, 32, 64, 128\}$ in C-KSA+Full. The six-length RULER mean ranges from 71.92 to 72.74, with no systematic decline as compression increases. The Full layers may help preserve retrieval quality by providing access to original-text details lost during compression.

At 128K, $k_m = 128$ leaves about 128 Meta entries per C-KSA layer, so larger k_m would save little further cache; we therefore select $k_m = 128$ and leave higher factors to million-token contexts.

Block size and routing methods. Table 9(b) compares sparse-attention schemes on the same $k_m = 128$ C-KSA+Full backbone, including S-KSA with $b \in \{2, 4, 8\}$, token-level DSA (DeepSeek-AI et al., 2025), and Qwen Sparse Attention (QSA) (Qiu et al., 2026). QSA provides a simple alternative to Summary-guided aggregation: it averages the projected token-level index keys within each block. We follow Qwen’s QSA indexer design, using $b = 4$ and eight indexer heads to match our comparison settings. All methods are compared under the same experimental setup. We report only RULER in this panel, since most general-capability sequences fit within the combined remote and local attention budget and offer limited discrimination of routing quality.

S-KSA with $b = 4$ achieves a mean RULER score comparable to those of $b = 2$ and token-level DSA, with only half and one quarter as many routing candidates, respectively. It also outperforms QSA at the same block size. Increasing to $b = 8$ reduces the candidate count further but degrades retrieval quality, particularly at 128K. We therefore select $b = 4$ as a balance between retrieval quality and routing cost.

Table 9 | Ablations of compression and sparse attention. Dashes indicate unreported scores since the sequence length is not enough to activate sparse mechanism.

Configuration	Long-context retrieval (RULER)							Knowledge			Mathematics				Code	
	4K	8K	16K	32K	64K	128K	Mean	MMLU	CMMLU	C-Eval	MMLU Pro	GSM8K	CMATH	MATH	MBPP	Human Eval
<i>(a) C-KSA + Full: compression granularity</i>																
$k_m = 4$	79.54	72.37	74.22	71.37	68.21	65.78	71.92	46.77	46.27	43.76	23.06	57.39	64.00	36.80	36.40	28.05
$k_m = 8$	79.42	72.56	74.35	71.69	68.95	66.15	72.19	46.78	46.41	43.76	23.12	57.16	64.50	36.46	36.60	29.27
$k_m = 16$	79.95	73.31	74.92	72.31	68.46	67.46	72.74	46.95	46.30	43.39	22.61	56.71	63.17	36.56	37.40	31.10
$k_m = 32$	80.69	72.73	74.04	71.86	69.15	66.90	72.56	47.04	46.30	43.46	22.83	56.33	62.67	37.06	36.20	28.66
$k_m = 64$	80.94	72.00	73.68	72.30	68.41	67.04	72.40	47.12	46.19	43.46	22.92	56.25	63.50	36.70	36.00	27.44
$k_m = 128$ (selected)	81.52	72.24	74.44	71.93	69.61	66.31	72.68	46.97	46.49	43.83	22.71	55.95	64.50	36.60	35.60	28.05
<i>(b) C-KSA, $k_m = 128$: sparse attention and teacher pooling</i>																
DSA (token routing)	81.38	72.22	75.12	71.22	69.37	65.37	72.45	—	—	—	—	—	—	—	—	—
QSA, $b = 4$, max	81.48	72.19	75.15	70.26	67.57	61.39	71.34	—	—	—	—	—	—	—	—	—
S-KSA, $b = 2$, max	81.08	72.67	75.30	71.58	69.21	65.23	72.51	—	—	—	—	—	—	—	—	—
S-KSA, $b = 4$, max (selected)	81.48	72.42	75.67	70.78	68.99	65.01	72.39	—	—	—	—	—	—	—	—	—
S-KSA, $b = 8$, max	81.38	72.47	74.15	68.29	64.83	57.62	69.79	—	—	—	—	—	—	—	—	—
S-KSA, $b = 4$, sum	81.48	72.15	74.81	70.28	68.51	61.68	71.49	—	—	—	—	—	—	—	—	—
<i>(c) Backbone for sparse adaptation: S-KSA, $b = 4$, max teacher</i>																
KSA + S-KSA	80.71	73.82	74.76	72.12	68.57	61.06	71.84	46.60	46.99	43.83	22.62	58.83	61.67	37.40	36.80	31.71
C-KSA + S-KSA, $k_m = 128$	81.48	72.42	75.67	70.78	68.99	65.01	72.39	47.09	46.39	42.94	22.83	56.33	63.50	36.72	35.20	29.88

Max versus sum teacher pooling. At $k_m = 128$ and $b = 4$, we compare the max-pooled teacher in Equation 15, which follows QSA (Qiu et al., 2026), with a sum-pooled teacher that uses $\sum_{i \in \mathcal{B}_u} a_{t,i}$ as each block’s unnormalized weight. Both runs use the same training setup, differing only in teacher pooling. Max pooling improves the mean RULER score from 71.49% to 72.39% and the 128K score from 61.68% to 65.01% (Table 9(b)). Retrieval likely depends on a few salient tokens, whose signal sum pooling dilutes with diffuse attention. These results confirm QSA’s choice on the KSA backbone.

Failure mode. The largest 128K subtask decrease is frequent-word extraction, from 45.00% to 26.67% (Table 7), an aggregation task.

Backbone for sparse adaptation. Table 9(c) compares S-KSA adapted on Hybrid KSA and C-KSA+Full with $k_m = 128$. Both use $b = 4$, a max-pooled teacher, and 18.5B tokens of module-only sparse adaptation with the same remote budget, local window, and sink settings. C-KSA+S-KSA achieves a mean RULER score of 72.39 and a 128K score of 65.01, compared with 71.84 and 61.06 for KSA+S-KSA. Relative to their respective dense references, the 128K reduction is 1.30 points for C-KSA versus 4.29 for Hybrid KSA, indicating better retention of the backbone’s retrieval performance in this comparison.

Figure 12(c) provides a complementary observation: at the same token budget, oracle block selection retains slightly more decode attention mass on C-KSA than on Hybrid KSA, including local context and sinks. This is consistent with the smaller retrieval loss after sparse adaptation and suggests that the compressed backbone remains well suited to sparse original-text access. Together, these results support the compatibility of hierarchical compression and sparse attention.

Sparse selection using Full attention scores. To test whether sparse attention requires backbone adaptation, we freeze all model parameters and select remote tokens or blocks directly using Full attention probabilities averaged across heads. Token selection ranks individual probabilities; block selection ranks the maximum probability within each block of $b = 4$ tokens, matching the Indexer’s teacher pooling rule. We call this *oracle* selection and compare it with the same backbone’s unchanged

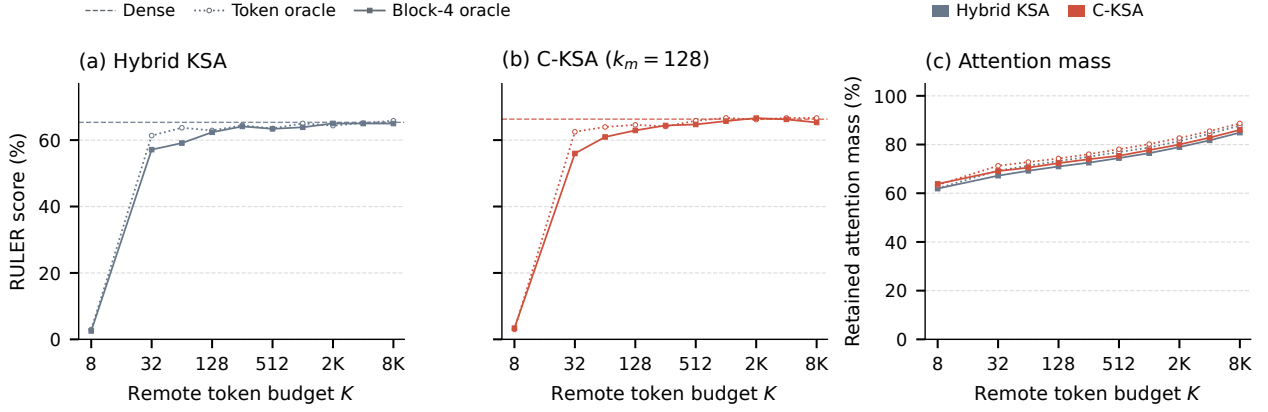


Figure 12 | Oracle selection at 128K with frozen Hybrid KSA (blue) and C-KSA+Full ($k_m = 128$, red). (a,b) RULER scores. (c) Decode attention mass retained by token and block selection ($b = 4$), including local context and sinks before sparse renormalization.

Table 10 | Dense and oracle RULER results with frozen backbones. Oracle selection uses $K = 4096$; Mean averages all six context lengths (4K–128K).

Backbone	Dense		Token oracle		Block-4 oracle	
	Mean	128K	Mean	128K	Mean	128K
Hybrid KSA	72.61	65.35	73.12	65.13	73.05	65.00
C-KSA+Full, $k_m = 16$	72.74	67.46	73.24	68.94	73.17	69.41
C-KSA+Full, $k_m = 32$	72.56	66.90	72.69	66.93	72.81	67.22
C-KSA+Full, $k_m = 64$	72.40	67.04	72.87	66.56	72.63	66.89
C-KSA+Full, $k_m = 128$	72.68	66.31	72.84	66.67	72.66	66.24

Full layers. The remote budget K includes eight sink tokens; a 1024-token local window is retained separately.

Figure 12 shows rapid recovery toward dense performance as the remote budget increases on both Hybrid KSA and C-KSA with $k_m = 128$. On C-KSA, block-level selection reaches 64.71 at $K = 512$. At $K = 4096$, the selected remote tokens and the 1024-token local window together cover only about 3.9% of a 128K context, yet retain 82.83% of decode attention mass. The corresponding RULER score is 66.24, compared with 66.31 for dense attention. This concentration of attention on a small subset of tokens helps explain why sparse attention can preserve retrieval performance when relevant tokens are selected. At the same budget, Hybrid KSA retains 81.71% of decode attention mass. Local context and sinks alone already retain substantial attention mass but yield poor retrieval performance, highlighting the importance of selecting relevant remote content.

Table 10 extends the comparison to Hybrid KSA and four C-KSA+Full backbones at $K = 4096$, reporting mean RULER performance across all six context lengths and the 128K score. Both selection methods remain within 0.51 percentage points of the dense references in mean RULER score across all five backbones. Accurate selection therefore recovers near-dense RULER performance without backbone updates in these settings, including on models already adapted to C-KSA.

Full-parameter sparse continuation also showed no clear RULER improvement under the training recipe and budget we tested. Together with the recovery observed after module-only training in Section 3.3, these results support focusing sparse adaptation on the aggregation module and Indexer rather than requiring backbone updates.

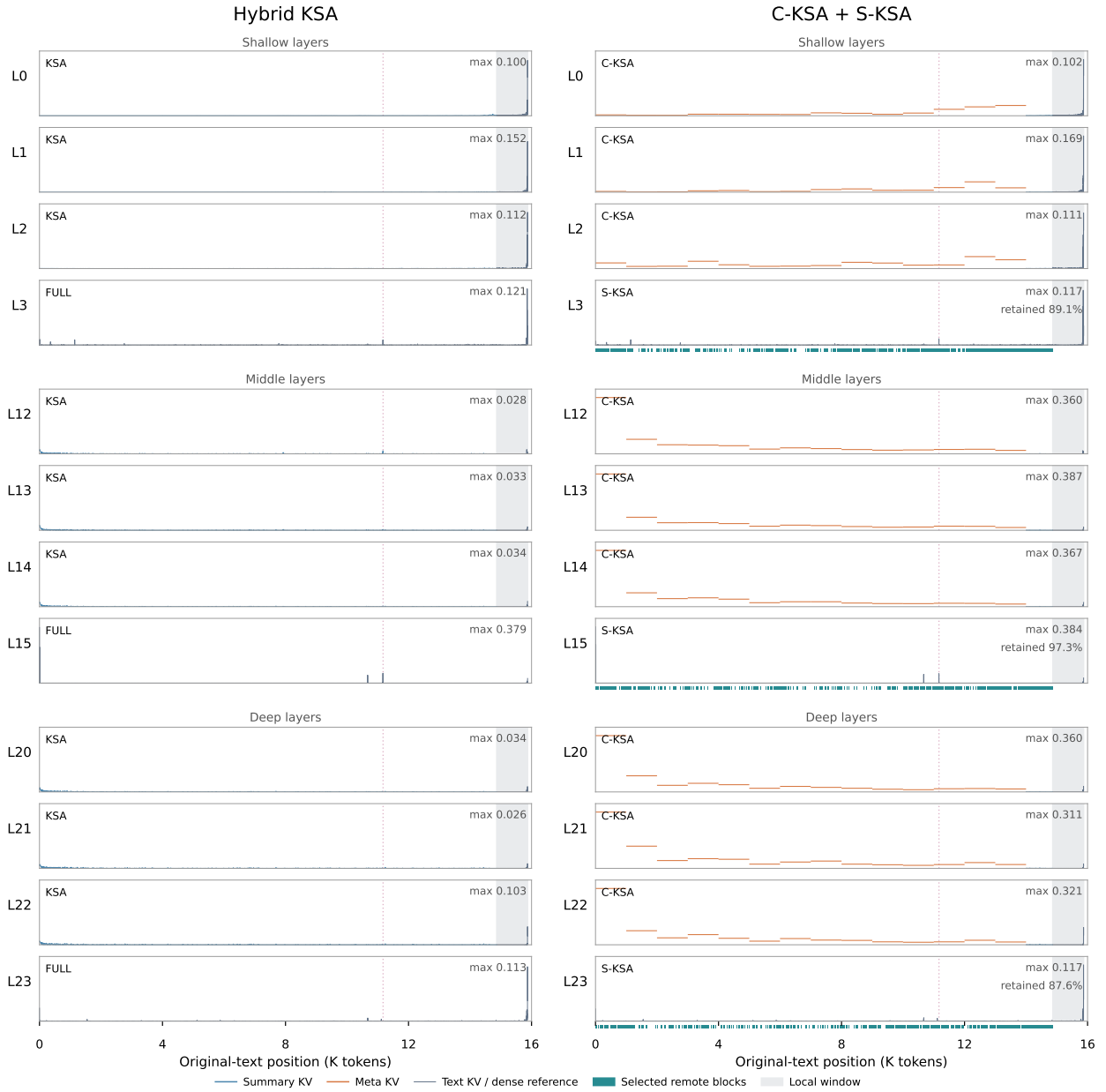


Figure 13 | Per-layer attention and block selection on a shared 16K NIAH example for Hybrid KSA (left) and C-KSA+S-KSA (right). Teal strips mark selected remote blocks, dashed lines mark the answer, and gray shading denotes the local window.

Attention patterns and block selection. Figure 13 compares layer-wise attention in Hybrid KSA and our main C-KSA+S-KSA model on a shared 16K NIAH example, using $k_m = 128$, $b = 4$, and $K = 4096$ for the latter. Both models correctly retrieve the answer from outside the local window. We visualize attention when predicting the first answer digit. For C-KSA, the plotted attention probabilities include the $\log k_m$ correction to Meta logits before softmax. For S-KSA, we plot dense attention probabilities and mark the text blocks selected by the Indexer.

For the visualized query, Meta entries receive 7.7–13.7% of C-KSA’s attention mass in the displayed shallow layers, compared with 73.1–87.8% in the middle and deep layers, showing substantial use of compressed remote memory. In all three displayed S-KSA layers, the selected blocks cover the answer span. Together with local context, these blocks retain 87.6–97.3% of the dense attention mass while accessing only 31.5% of the original-text positions.

Continued pretraining of Hybrid KSA. Because our adapted models receive additional training while the reported baselines do not, we examine whether continued pretraining of the unchanged Hybrid KSA backbone would provide a stronger quality reference. Table 11 reports a full-parameter continuation experiment from the same 128K checkpoint, using a global batch size of 256, a peak learning rate of 4×10^{-4} , and a warmup–stable–decay schedule. Across five evaluated checkpoints covering 5.7–51.3B additional tokens, the six-length RULER mean remains below the original checkpoint’s 72.60. The 128K score improves at 28.5B tokens but falls back at later checkpoints. Under the tested recipe and data budgets, continued pretraining does not yield a consistent improvement in Hybrid KSA’s RULER performance. This supports using the original checkpoint as the quality reference for assessing whether architectural adaptation can preserve its capabilities at lower inference cost.

Table 11 | RULER scores after continued pretraining of Hybrid KSA.

Added tokens (B)	Mean	128K
0 (original)	72.60	<u>65.27</u>
5.7	70.84	63.26
17.1	70.77	63.72
28.5	71.86	66.76
39.9	69.74	63.17
51.3	<u>71.95</u>	63.62

4. Related Work

Multi-head attention (MHA) is a fundamental component of Transformer language models, with separate query, key, and value projections for each head (Vaswani et al., 2017). Full attention provides direct access to preceding tokens, but its computation during training and prefill grows quadratically with sequence length, while its KV cache grows linearly during autoregressive decoding. FlashAttention reduces memory traffic for exact attention (Dao et al., 2022), while PagedAttention improves KV-cache allocation and sharing (Kwon et al., 2023). These system-level optimizations complement research on memory compression, sparse attention, and hybrid architectures.

4.1. Memory Compression

One line of memory compression reduces storage at each token position. Multi-query attention (MQA) shares a single KV head across all query heads (Shazeer, 2019), while grouped-query attention (GQA)

shares KV heads within groups of query heads (Ainslie et al., 2023). Multi-head latent attention (MLA) stores compact per-token latent states (DeepSeek-AI et al., 2024), and quantization methods such as KIVI reduce the precision of cached entries (Liu et al., 2024). These approaches reduce storage per position without removing historical positions. Other methods retain only a subset of token-level KV entries: H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024b) use attention statistics to identify important positions, while StreamingLLM (Xiao et al., 2024) retains initial attention sinks together with a recent window.

Another direction compresses historical context into fixed-size recurrent states rather than retaining token-level KV entries. Linear Transformers (Katharopoulos et al., 2020) achieve this through kernel feature maps and recurrent accumulation of key-value statistics. Subsequent designs introduce decay in RetNet (Sun et al., 2023), data-dependent gating in GLA (Yang et al., 2024), and gated delta-rule updates in Gated DeltaNet (Yang et al., 2025). Kimi Delta Attention (KDA) (Kimi Team et al., 2025) further extends Gated DeltaNet with finer-grained gating for recurrent memory updates. Related approaches include RWKV (Peng et al., 2024), with matrix-valued recurrent states, and selective state-space models such as Mamba (Gu and Dao, 2024) and Mamba-2 (Dao and Gu, 2024). In their recurrent forms, these models maintain a state size independent of context length, but access historical information through aggregated states rather than separately retained memory entries.

Sequence-level aggregation represents multiple context positions with fewer memory entries that remain accessible to attention. Compressive Transformer (Rae et al., 2020) compresses older hidden-state memories, while Dynamic Memory Compression (Nawrot et al., 2024) learns adaptive KV-cache compression. Learned summary and memory representations are explored in Gist (Mu et al., 2023), ICAE (Ge et al., 2024), and UniGist (Deng et al., 2025a). AutoCompressors (Chevalier et al., 2023) uses summary vectors as soft prompts, whereas Activation Beacon (Zhang et al., 2025b) retains compressed KV activations at each layer. GMSA (Tang et al., 2026) combines group merging and layer semantic alignment in an encoder-decoder context-compression framework. Cartridges (Eyuboglu et al., 2026) instead optimizes corpus-specific KV representations offline for reuse across subsequent queries.

KSA (Chu et al., 2026) interleaves Summary tokens with fixed-size text chunks. In KSA layers, text queries directly access nearby text and retrieve distant context through Summary KV, while the Summary hidden states continue to be updated across layers. A systematic study of gist-based compression identifies task-dependent limitations, particularly in synthetic recall (Deng et al., 2025b). These findings underscore the importance of evaluating retained capabilities rather than compression ratio alone.

Other designs organize compressed memory at multiple resolutions. MELODI (Chen et al., 2025) combines recurrent short-term memory with further-compressed long-term memory constructed at an intermediate layer. AdmTree (Li et al., 2025) organizes gist representations into adaptive semantic trees for hierarchical context compression. H-SSA (Mao et al., 2026) inserts meta-gist tokens to summarize groups of gist tokens and support coarse-to-fine retrieval. DeepSeek-V4 (DeepSeek-AI et al., 2026) applies learned dimension-wise aggregation to text-token KV, using different compression granularities in Compressed Sparse Attention (CSA) and Heavily Compressed Attention (HCA). C-KSA instead applies dimension-wise aggregation to Summary KV, forming Meta entries used only within the layer that constructs them, rather than introducing additional tokens into the hidden-state sequence. This further reduces distant-history storage without coarsening the Summary sequence passed between layers, and permits each layer to choose its own KV compression ratio.

4.2. Sparse Attention

Sparse attention reduces computation by restricting the positions read by each query. Sparse Transformer (Child et al., 2019) uses structured sparse factorizations, Longformer (Beltagy et al., 2020) combines local windows with global attention, and BigBird (Zaheer et al., 2020) additionally incorporates random connections. Sliding-window attention, used in Mistral (Jiang et al., 2023), restricts each layer’s direct access to nearby tokens. Another line exploits attention sparsity in pretrained models without additional training. QUEST (Tang et al., 2024) estimates the relevance of KV-cache pages to each query and reads only the selected pages. MInference (Jiang et al., 2024) combines head-specific pattern templates with input-dependent sparse indices to accelerate prefill, while SpargeAttention (Zhang et al., 2025a) and XAttention (Xu et al., 2025) estimate attention importance to skip low-contribution computations. These methods sparsify attention through prescribed patterns or inference-time selection, without training an auxiliary routing module.

Content-dependent sparse architectures integrate routing into the attention mechanism. SeerAttention (Gao et al., 2025) learns block-level gates by self-distilling dense attention, while DeepSeek Sparse Attention (DSA) (DeepSeek-AI et al., 2025) uses a Lightning Indexer to select individual tokens. At block granularity, MoBA (Lu et al., 2025) scores mean-pooled token keys, whereas Qwen Sparse Attention (QSA) (Qiu et al., 2026) mean-pools auxiliary index keys within micro-blocks; both attend to the selected blocks’ original-token KV. Native Sparse Attention (NSA) (Yuan et al., 2025) uses learned block compression to guide fine-grained selection and combines compressed, selected, and local attention branches. In contrast, DeepSeek-V4’s CSA (DeepSeek-AI et al., 2026) selects compressed KV entries for its remote attention. S-KSA also reduces routing to block granularity, but reuses existing KSA Summaries as aggregation queries to construct routing representations and retains original-token KV for the main attention.

Learned memory tokens can also provide an interface for block retrieval. Landmark Attention (Mothashami and Jaggi, 2023) represents each context block with a landmark token and learns to retrieve blocks through attention to these landmarks. Simplified Sparse Attention (SSA) (Mao et al., 2026) scores gist keys and unfolds selected chunks to recover their original tokens. HiLS-Attention (Hu et al., 2026) uses landmark queries to pool token keys, then combines within-chunk attention outputs using entropy-corrected chunk weights. This allows the language-modeling loss to train routing; SAS (Li et al., 2026) similarly learns context ranking through differentiable gates inside attention normalization. In S-KSA, Summaries guide auxiliary aggregation rather than serving directly as routing keys. We train the aggregation modules and Indexer by attention distillation and use the resulting scores only for block selection. The main attention applies a single softmax over selected original tokens and local context, without using routing scores as additional attention weights.

4.3. Hybrid Architectures

Hybrid architectures seek to retain the fine-grained retrieval of attention while reducing the cost of processing long contexts. A common strategy interleaves several efficient layers with a smaller number of full-attention layers. Hybrid H3 (Fu et al., 2023), Mamba-2-Hybrid (Waleffe et al., 2024), and Jamba (Lenz et al., 2025) combine state-space layers with attention, while Kimi Linear (Kimi Team et al., 2025) combines KDA with MLA. Local attention provides another complementary component: RecurrentGemma (Botev et al., 2024) and Samba (Ren et al., 2025) pair it with recurrent or state-space layers, whereas Gemma 3 (Gemma Team et al., 2025) and MiMo-V2-Flash (Xiaomi LLM-Core Team et al., 2026) interleave it with global attention.

Beyond layer-wise interleaving, other designs exploit sharing or finer-grained mixing. Zamba (Glorioso et al., 2024) reuses a shared attention module at multiple depths of a Mamba backbone.

YOCO (Sun et al., 2024) stacks a cross-decoder on an efficient self-decoder and shares a global KV cache across cross-decoder layers. HySparse (Gao et al., 2026) reuses token selections and global KV from full-attention layers in subsequent sparse layers, while DeepSeek-V4.1 (DeepSeek-AI, 2026) introduces cross-layer KV and index reuse within compressed sparse attention. At a finer granularity, Hymba (Dong et al., 2025) runs attention and state-space heads in parallel within the same layer.

Hybrid KSA (Chu et al., 2026) combines Summary-based KSA layers with full-attention layers in a 3:1 layout. KSA V2 retains this layer-wise structure while improving both components: C-KSA further compresses the Summary KV used for distant-history access, and S-KSA replaces dense attention over original text with Summary-guided block-sparse attention.

5. Conclusion and Future Work

We showed that the Summaries of a summary-token hybrid can serve both as compressible memory and as a routing index. C-KSA compresses Summary KV within each layer without coarsening the Summaries passed between layers, and S-KSA replaces full attention with Summary-guided block-sparse attention over original text. Retrofitted into a pretrained 1.9B Hybrid KSA model, KSA V2 retains its retrieval performance and largely preserves general capabilities, while reducing cache size and improving inference throughput. Additional experiments on a 4B GQA backbone retain near-parent RULER performance, supporting transfer to a larger model with shared KV heads. The remaining cost lies mainly in the original-text KV stored by the sparse layers, a natural target for future work.

Our evaluation covers 1.9B MHA and 4B GQA backbones with contexts up to 128K. General-capability evaluations and efficiency measurements are limited to 1.9B; the 4B validation covers RULER retrieval. Future work will explore longer contexts and larger models, including whether additional compression levels can provide further savings while preserving useful information. We will also explore how to allocate compression factors across layers. Evaluating long-document understanding and multi-step reasoning will also help establish how well the observed retrieval performance transfers to broader long-context tasks.

6. Author List and Acknowledgement

Core Contributors Chenglong Chu, Guohong Mu, Guowang Zhang, Hao Peng, Hui Wang, Jian Liang, Jiangxia Cao[†], Jiao Ou, Jiaxin Deng, Lu Ren^{*}, Muhao Wei, Ruiming Tang^{*}, Tianle Xu, Wenwu Ou, Xingmei Wang, Zhixin Ling, Zhiyuan Liang^{*}, Ziqi Wang^{*}

Contributors Chaoyi Ma, Haixiao Gao, Kuo Cai, Renpu Liu, Ruochen Yang, Wenlong Xu, Xinqi Jin, Yan Sun, Yifan Wang, Yingzhi He, Yiyu Wang, Yusen Huo, Zhaojie Liu, Zihan Zhou

All the authors listed alphabetically by first name. ^{*}individuals who have departed from our team.

[†]Project Leader: caojiangxia@kuaishou.com

References

- Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30.
- Child, R., S. Gray, A. Radford, and I. Sutskever (2019). “Generating Long Sequences with Sparse Transformers”. In: *arXiv preprint arXiv:1904.10509*.
- Shazeer, N. (2019). “Fast Transformer Decoding: One Write-Head is All You Need”. In: *arXiv preprint arXiv:1911.02150*.
- Beltagy, I., M. E. Peters, and A. Cohan (2020). “Longformer: The Long-Document Transformer”. In: *arXiv preprint arXiv:2004.05150*.
- Katharopoulos, A., A. Vyas, N. Pappas, and F. Fleuret (2020). “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”. In: *International Conference on Machine Learning*. PMLR, pp. 5156–5165.
- Rae, J. W., A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap (2020). “Compressive Transformers for Long-Range Sequence Modelling”. In: *International Conference on Learning Representations*.
- Zaheer, M., G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed (2020). “Big Bird: Transformers for Longer Sequences”. In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 17283–17297.
- Austin, J., A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton (2021). “Program Synthesis with Large Language Models”. In: *arXiv preprint arXiv:2108.07732*.
- Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, et al. (2021). “Evaluating Large Language Models Trained on Code”. In: *arXiv preprint arXiv:2107.03374*.
- Cobbe, K., V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman (2021). “Training Verifiers to Solve Math Word Problems”. In: *arXiv preprint arXiv:2110.14168*.
- Hendrycks, D., C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt (2021a). “Measuring Massive Multitask Language Understanding”. In: *International Conference on Learning Representations*.
- Hendrycks, D., C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt (2021b). “Measuring Mathematical Problem Solving With the MATH Dataset”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Vol. 1.
- Dao, T., D. Fu, S. Ermon, A. Rudra, and C. Ré (2022). “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 16344–16359.
- Ainslie, J., J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai (2023). “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901.
- Chevalier, A., A. Wettig, A. Ajith, and D. Chen (2023). “Adapting Language Models to Compress Contexts”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846.
- Fu, D. Y., T. Dao, K. K. Saab, A. W. Thomas, A. Rudra, and C. Ré (2023). “Hungry Hungry Hippos: Towards Language Modeling with State Space Models”. In: *International Conference on Learning Representations*.
- Huang, Y., Y. Bai, Z. Zhu, J. Zhang, J. Zhang, T. Su, J. Liu, C. Lv, Y. Zhang, j. lei jiayi, Y. Fu, M. Sun, and J. He (2023). “C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models”. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 62991–63010.

- Jiang, A. Q., A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed (2023). “Mistral 7B”. In: *arXiv preprint arXiv:2310.06825*.
- Kamradt, G. (2023). *Needle In A Haystack - Pressure Testing LLMs*. Official website / repository. Evaluation repository, accessed September 17, 2026.
- Kwon, W., Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica (2023). “Efficient Memory Management for Large Language Model Serving with PagedAttention”. In: *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626.
- Mohtashami, A. and M. Jaggi (2023). “Random-Access Infinite Context Length for Transformers”. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 54567–54585.
- Mu, J., X. Li, and N. Goodman (2023). “Learning to Compress Prompts with Gist Tokens”. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 19327–19352.
- Sun, Y., L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, and F. Wei (2023). “Retentive Network: A Successor to Transformer for Large Language Models”. In: *arXiv preprint arXiv:2307.08621*.
- Wei, T., J. Luan, W. Liu, S. Dong, and B. Wang (2023). “CMATH: Can Your Language Model Pass Chinese Elementary School Math Test?” In: *arXiv preprint arXiv:2306.16636*.
- Zhang, Z., Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen (2023). “H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models”. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 34661–34710.
- Botev, A., S. De, S. L. Smith, A. Fernando, G.-C. Muraru, R. Haroun, L. Berrada, R. Pascanu, P. G. Sessa, et al. (2024). “RecurrentGemma: Moving Past Transformers for Efficient Open Language Models”. In: *arXiv preprint arXiv:2404.07839*.
- Dao, T. and A. Gu (2024). “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality”. In: *International Conference on Machine Learning*. PMLR, pp. 10041–10071.
- DeepSeek-AI, A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, et al. (2024). “DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model”. In: *arXiv preprint arXiv:2405.04434*.
- Ge, T., H. Jing, L. Wang, X. Wang, S.-Q. Chen, and F. Wei (2024). “In-context Autoencoder for Context Compression in a Large Language Model”. In: *International Conference on Learning Representations*, pp. 2591–2607.
- Glorioso, P., Q. Anthony, Y. Tokpanov, J. Whittington, J. Pilault, A. Ibrahim, and B. Millidge (2024). “Zamba: A Compact 7B SSM Hybrid Model”. In: *arXiv preprint arXiv:2405.16712*.
- Gu, A. and T. Dao (2024). “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *Conference on Language Modeling*.
- Hsieh, C.-P., S. Sun, S. Krizan, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg (2024). “RULER: What’s the Real Context Size of Your Long-Context Language Models?” In: *Conference on Language Modeling*.
- Jiang, H., Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu (2024). “MInference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention”. In: *Advances in Neural Information Processing Systems*. Vol. 37, pp. 52481–52515.
- Li, H., Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan, and T. Baldwin (2024a). “CMMLU: Measuring massive multitask language understanding in Chinese”. In: *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11260–11285.
- Li, Y., Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen (2024b). “SnapKV: LLM Knows What You are Looking for Before Generation”. In: *Advances in Neural Information Processing Systems*. Vol. 37, pp. 22947–22970.

- Liu, Z., J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu (2024). “KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache”. In: *International Conference on Machine Learning*. PMLR, pp. 32332–32344.
- Nawrot, P., A. Łańcucki, M. Chochowski, D. Tarjan, and E. Ponti (2024). “Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference”. In: *International Conference on Machine Learning*. PMLR, pp. 37396–37412.
- Peng, B., D. Goldstein, Q. G. Anthony, A. Albalak, E. Alcaide, S. Biderman, E. Cheah, T. Ferdinan, K. K. GV, H. Hou, S. Krishna, R. McClelland Jr., N. Muennighoff, F. Obeid, A. Saito, G. Song, H. Tu, R. Zhang, B. Zhao, Q. Zhao, J. Zhu, and R.-J. Zhu (2024). “Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence”. In: *Conference on Language Modeling*.
- Su, J., M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu (2024). “RoFormer: Enhanced transformer with Rotary Position Embedding”. In: *Neurocomputing* 568, p. 127063.
- Sun, Y., L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei (2024). “You Only Cache Once: Decoder-Decoder Architectures for Language Models”. In: *Advances in Neural Information Processing Systems*. Vol. 37, pp. 7339–7361.
- Tang, J., Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han (2024). “QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference”. In: *International Conference on Machine Learning*. PMLR, pp. 47901–47911.
- Waleffe, R., W. Byeon, D. Riach, B. Norick, V. Korthikanti, T. Dao, A. Gu, A. Hatamizadeh, S. Singh, D. Narayanan, G. Kulshreshtha, V. Singh, J. Casper, J. Kautz, M. Shoeybi, and B. Catanzaro (2024). “An Empirical Study of Mamba-based Language Models”. In: *arXiv preprint arXiv:2406.07887*.
- Wang, Y., X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen (2024). “MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark”. In: *Advances in Neural Information Processing Systems*. Vol. 37, pp. 95266–95290.
- Xiao, G., Y. Tian, B. Chen, S. Han, and M. Lewis (2024). “Efficient Streaming Language Models with Attention Sinks”. In: *International Conference on Learning Representations*, pp. 21875–21895.
- Yang, S., B. Wang, Y. Shen, R. Panda, and Y. Kim (2024). “Gated Linear Attention Transformers with Hardware-Efficient Training”. In: *International Conference on Machine Learning*. PMLR, pp. 56501–56523.
- Chen, Y., D. Hutchins, A. Jansen, A. Zhmoginov, D. Racz, and J. Andersen (2025). “MELODI: Exploring Memory Compression for Long Contexts”. In: *International Conference on Learning Representations*, pp. 67589–67611.
- DeepSeek-AI, A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, et al. (2025). “DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models”. In: *arXiv preprint arXiv:2512.02556*.
- Deng, C., Z. Zhang, K. Mao, S. Li, T. Fang, H. Zhang, H. Mi, D. Yu, and Z. Dou (2025a). “UniGist: Towards General and Hardware-aligned Sequence-level Long Context Compression”. In: *Advances in Neural Information Processing Systems*. Vol. 38, pp. 19775–19797.
- Deng, C., Z. Zhang, K. Mao, S. Li, X. Huang, D. Yu, and Z. Dou (2025b). “A Silver Bullet or a Compromise for Full Attention? A Comprehensive Study of Gist Token-based Context Compression”. In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4861–4879.
- Dong, X., Y. Fu, S. Diao, W. Byeon, Z. Chen, A. Mahabaleshwarkar, S.-Y. Liu, M. Van keirsbilck, M.-H. Chen, Y. Suhara, Y. C. Lin, J. Kautz, and P. Molchanov (2025). “Hymba: A Hybrid-head Architecture for Small Language Models”. In: *International Conference on Learning Representations*, pp. 97729–97755.
- Gao, Y., Z. Zeng, D. Du, S. Cao, P. Zhou, J. Qi, J. Lai, H. So, T. Cao, F. Yang, and M. Yang (2025). “SeerAttention: Self-distilled Attention Gating for Efficient Long-context Prefilling”. In: *Advances in Neural Information Processing Systems*. Vol. 38, pp. 55846–55869.

- Gemma Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, et al. (2025). “Gemma 3 Technical Report”. In: *arXiv preprint arXiv:2503.19786*.
- Kimi Team, Y. Zhang, Z. Lin, X. Yao, J. Hu, F. Meng, C. Liu, X. Men, S. Yang, et al. (2025). “Kimi Linear: An Expressive, Efficient Attention Architecture”. In: *arXiv preprint arXiv:2510.26692*.
- Lenz, B., O. Lieber, A. Arazi, A. Bergman, A. Manevich, B. Peleg, B. Aviram, C. Almagor, C. Fridman, et al. (2025). “Jamba: Hybrid Transformer-Mamba Language Models”. In: *International Conference on Learning Representations*, pp. 67959–67984.
- Li, Y., S. Chen, Y. Li, Y. Chen, H.-T. Zheng, H. Wang, W. Jiang, and P. S. Yu (2025). “AdmTree: Compressing Lengthy Context with Adaptive Semantic Trees”. In: *Advances in Neural Information Processing Systems*. Vol. 38, pp. 40389–40415.
- Lu, E., Z. Jiang, J. Liu, Y. Du, T. Jiang, C. Hong, S. Liu, W. He, E. Yuan, Y. Wang, Z. Huang, H. Yuan, S. Xu, X. Xu, G. Lai, Y. Chen, H. Zheng, J. Yan, J. Su, Y. Wu, Y. Zhang, Z. Yang, X. Zhou, M. Zhang, and J. Qiu (2025). “MoBA: Mixture of Block Attention for Long-Context LLMs”. In: *Advances in Neural Information Processing Systems*. Vol. 38, pp. 17790–17815.
- Ren, L., Y. Liu, Y. Lu, y. shen yelong, C. Liang, and W. Chen (2025). “Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling”. In: *International Conference on Learning Representations*, pp. 53551–53575.
- Xu, R., G. Xiao, H. Huang, J. Guo, and S. Han (2025). “XAttention: Block Sparse Attention with Antidiagonal Scoring”. In: *International Conference on Machine Learning*. PMLR, pp. 69819–69831.
- Yang, S., J. Kautz, and A. Hatamizadeh (2025). “Gated Delta Networks: Improving Mamba2 with Delta Rule”. In: *International Conference on Learning Representations*, pp. 29687–29707.
- Yuan, J., H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, and W. Zeng (2025). “Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention”. In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097.
- Zhang, J., C. Xiang, H. Huang, J. Wei, H. Xi, J. Zhu, and J. Chen (2025a). “SpargeAttention: Accurate and Training-free Sparse Attention Accelerating Any Model Inference”. In: *International Conference on Machine Learning*. PMLR, pp. 76397–76413.
- Zhang, P., Z. Liu, S. Xiao, N. Shao, Q. Ye, and Z. Dou (2025b). “Long Context Compression with Activation Beacon”. In: *International Conference on Learning Representations*, pp. 101932–101948.
- Chu, C., G. Zhou, G. Zhang, H. Li, H. Peng, H. Cheng, H. Wang, J. Liang, J. Cao, et al. (2026). “Kwai Summary Attention Technical Report”. In: *arXiv preprint arXiv:2604.24432*.
- DeepSeek-AI (2026). “DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression”. In: *arXiv preprint arXiv:2609.19969*.
- DeepSeek-AI, A. Xu, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, et al. (2026). “DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence”. In: *arXiv preprint arXiv:2606.19348*.
- Eyuboglu, S., R. Ehrlich, S. Arora, N. Guha, D. Zinsley, E. Liu, A. Rudra, J. Y. Zou, A. Mirhoseini, and C. Re (2026). “Cartridges: Lightweight and general-purpose long context representations via self-study”. In: *International Conference on Learning Representations*, pp. 42642–42687.
- Gao, Y., J. Wei, Q. Zhang, Y. Cheng, S. Chen, Z. Tang, Z. Jiang, Y. Song, H. Zhang, L. Zhao, B. Yang, G. Wang, S. Cao, and F. Luo (2026). “HySparse: A Hybrid Sparse Attention Architecture with Oracle Token Selection and KV Cache Sharing”. In: *arXiv preprint arXiv:2602.03560*.
- Hu, X., X. Wei, H. Gu, M. Zhang, T. Liang, H. Li, L. Zhu, Y. Wang, S. Han, Y. Bai, K. Tu, H. Mi, and L. Liang (2026). “Hierarchical Sparse Attention Done Right: Toward Infinite Context Modeling”. In: *arXiv preprint arXiv:2607.02980*.
- Li, Z., L. Zhu, H. Gu, X. Hu, Y. Wang, H. Mi, S. Han, L. Liang, and Z. Guo (2026). “SAS: Simple Attention Sparsification via End-to-End Optimization of Context Ranking”. In: *arXiv preprint arXiv:2609.13141*.

- Mao, Y., M. Y. Li, and E. B. Fox (2026). “Simplified Sparse Attention via Gist Tokens”. In: *arXiv preprint arXiv:2604.20920*.
- Qiu, Z., Z. Wang, X. Li, Y. Li, Y. Xu, Y. Wang, H. Zhang, R. Men, B. Mao, et al. (2026). “On the Design of Qwen3.8-Next Architecture: Evaluation, Efficiency, and Training Stability”. In: *arXiv preprint arXiv:2608.30320*.
- Tang, J., Z. Zhang, S. Wu, J. Ye, L. Bai, Z. Wang, T. Lu, L. Hai, Y. Zhao, H.-T. Zheng, and H.-G. Kim (2026). “GMSA: Enhancing Context Compression via Group Merging and Layer Semantic Alignment”. In: *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28690–28704.
- Xiaomi LLM-Core Team, B. Xiao, B. Xia, B. Yang, B. Gao, B. Shen, C. Zhang, C. He, C. Lou, et al. (2026). “MiMo-V2-Flash Technical Report”. In: *arXiv preprint arXiv:2601.02780*.