SIMPLIFIED SPARSE ATTENTION VIA GIST TOKENS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention can reduce the cost of long-context inference, but most variants introduce new architectural components. We introduce Simplified Sparse Attention (SSA), a simpler approach to sparse attention that requires no architectural changes. Concretely, we first perform continued pretraining on sequences interleaved with gist tokens. We optimize the standard next-token loss as usual, but the gist tokens use an attention mask to restrict what parts of the context the language model can attend to; this teaches the model to pack each chunk's important information into the gist tokens. At inference time, SSA scores chunks via attention between the current query and the small set of gist tokens, *selectively unfolding* the top-k chunks by reintroducing their corresponding raw tokens. Since the query is scored only against the gist tokens, we avoid the memory-bandwidth cost associated with naive scoring against the full KV cache, without requiring the auxiliary KV cache approach used by sparse attention methods. On LongBench, SSA consistently outperforms compression and inference-time sparse-attention baselines under the same compression ratio. More strikingly, in retrieval-augmented generation, SSA can even outperform the continued-pretrained full-attention model by over 5.7 points. We attribute this to the ability of SSA's selective unfolding, which concentrates attention on the query-relevant chunks and effectively filters out noise. Beyond accuracy, SSA also improves efficiency: its decoding latency remains nearly flat as context length increases, yielding up to a 3.37$\times$ end-to-end decoding speedup over Flash-Decoding, while matching and eventually outperforming the FlashAttention baseline during prefill. SSA further extends to a hierarchical gist-of-gist variant (H-SSA) that achieves log-linear decoding complexity while maintaining or improving accuracy at high compression ratios up to 32$\times$.

1 INTRODUCTION

Long-context modeling has emerged as a critical capability for next-generation large language models (LLMs), driven by diverse real-world applications including in-depth reasoning (Zelikman et al., 2022; Guo et al., 2025), repository-level software engineering (Zhang et al., 2023a; Yang et al., 2024b), and multi-turn autonomous agent systems (AlphaEvolve team, 2025; Liu et al., 2026). However, as context lengths grows, the attention computation becomes increasingly dominant in the overall cost, presenting significant obstacles for both training and inference.

Sparse attention offers a promising way to mitigate this bottleneck at inference time. By restricting each query to attend to only a subset of keys, sparse methods can substantially reduce the cost of attention while preserving model capabilities. Existing approaches can be broadly categorized along two axes: methods that apply sparsity at inference time and methods that also introduce sparsity during training.

On the inference side, methods such as H_2O (Zhang et al., 2023b), StreamingLLM (Xiao et al., 2023), and Quest (Tang et al., 2024) apply sparsity during decoding by evicting or selecting KV-cache entries. While effective at reducing memory footprint, these approaches are applied post hoc to full-attention pretrained models, treating sparsity as an approximation rather than allowing the model to learn input-dependent sparse patterns during pretraining. On the training side, recent methods including Native Sparse Attention (NSA; Yuan et al., 2025), Deepseek Sparse Attention (DSA; Liu et al., 2025) and MoBA (Lu et al., 2025) have demonstrated that incorporating sparsity during pretraining can preserve or even improve model quality while achieving substantial speedups. This type of idea has also been explored in post-training in Neural Garbage Collection (Li et al., 2026) which

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

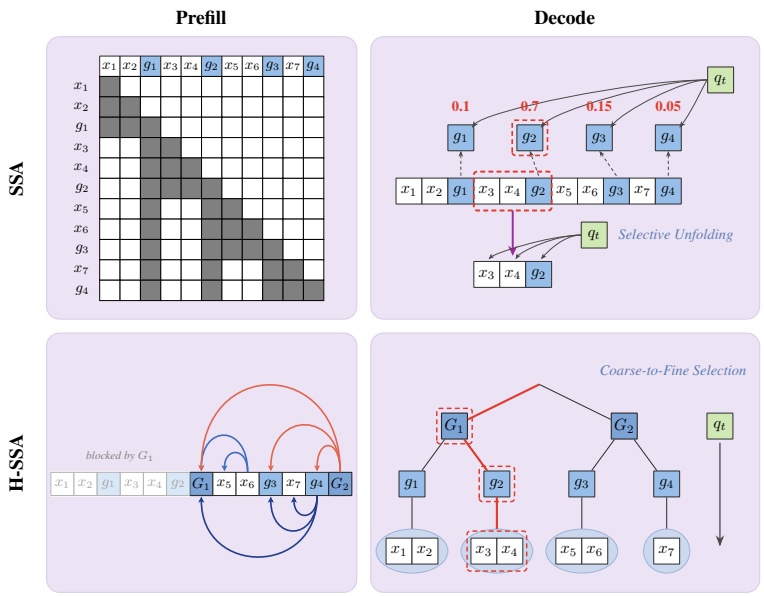


Figure 1: **SSA at a glance. Top row (SSA).** During prefill, interleaved gist tokens g_i compress each length-L chunk under a causal mask; during decode, the query q_t scores all gists and *selectively unfolds* the most relevant chunk (g_2) for sparse attention. **Bottom row (H-SSA).** We can also consider gists of gists: meta-gist tokens G_k summarize groups of gists, enabling coarse-to-fine selection at decode for log-linear cost.

trains a language model to evict from its own KV cache using reinforcement learning. However, existing training-based approaches often require additional architectural components or specialized mechanisms. This raises a natural question: can we obtain the benefits of training-based sparse attention without modifying the model architecture?

To design an architecture-free sparse attention method, we take inspiration from gist tokens (Mu et al., 2023; Chevalier et al., 2023; Yang et al., 2025; Zhang et al., 2024; Petrov et al., 2025; Deng et al., 2025). The idea is to insert a small number of special tokens—gist tokens—at fixed positions in the sequence, partitioning the context into chunks. Training uses the standard next-token prediction objective but with a change to the attention mask: tokens in a chunk can attend to earlier raw tokens only through the preceding chunk's gist tokens, not directly. Therefore, to minimize next-token loss under this constraint, the model must learn to pack the information needed for future predictions into the gists. Compression therefore emerges as a consequence of end-to-end optimization under constraints (Li et al., 2026), with no auxiliary loss, no new parameters, and no architectural additions.

To that end, we introduce **Simplified Sparse Attention (SSA)**, which interleaves gist tokens during continued pretraining and then uses these gist tokens to perform sparse attention during decoding. Concretely, after compressing the context into interleaved gist tokens, we perform Top-k selection over these gist tokens based on their attention scores with respect to the current query, and then *selectively unfold* the corresponding chunks by reintroducing their raw tokens into the context.

Remarkably, we find that an entirely pretraining-based intervention suffices to make this Top-k selection procedure highly effective. In particular, under a fixed token budget, this selective unfolding strategy significantly improves accuracy compared to standard compression baselines, including both append-only and interleaved gist methods, as well as inference-only sparse attention techniques. To explicitly teach the model to adapt to sparse attention, we can also incorporate the selective unfolding mechanism into the training process. Importantly, SSA is lightweight and simple to implement: it re-purposes a standard training procedure, continued pre-training, into a general purpose recipe for turning any full-attention architecture into a sparse attention method.

We additionally observe that gist tokens can also compress *other* gist tokens; in principle this procedure can continue *ad infinitum*. By hierarchically compressing gist tokens into higher-level summaries (meta-gist), we obtain a multi-resolution representation of the context. The selective

unfolding mechanism can then be applied in a coarse-to-fine manner: starting from the highest level of abstraction, the model progressively identifies the most relevant sub-contexts, unfolding from coarse summaries down to the original raw tokens. This hierarchical design captures information at multiple levels of granularity while maintaining computational efficiency that scales log-linearly with context length.

Empirically, SSA consistently outperforms gist-based compression and inference-time sparse attention baselines on LongBench across $8\times$–$32\times$ compression, and even surpasses full attention by up to 6.5 points in RAG. It also keeps decoding latency nearly flat as context grows, yielding up to a $3.37\times$ end-to-end decoding speedup over Flash-Decoding.

2 METHOD

We present Simplified Sparse Attention (SSA), a framework that bridges learnable gist-based context compression with sparse attention through a selective unfolding mechanism. The key idea is that interleaved gist tokens, trained to summarize local chunks, can simultaneously serve as *learned routing signals* for identifying the k most relevant sub-contexts. When a chunk is deemed relevant, its raw tokens are selectively reintroduced into the attention context, thereby recovering fine-grained detail precisely where needed. This results in a bounded context size of $|\mathcal{V}_t| = k \cdot (1 + L)$, where L is the chunk length and is independent of the full sequence length. The framework further supports two complementary stages: continued pretraining alone already enables effective selective unfolding at inference time, while optional selective finetuning with unfolded tokens further improves performance, making SSA both effective and flexible in practice.

Our framework is inspired by how humans process long contexts. When reading a novel or navigating a large codebase, readers do not retain every detail uniformly. Instead, they compress earlier content into high-level summaries—the gist—which is usually sufficient for ongoing comprehension. Only when a specific detail becomes necessary (e.g., a character's exact words or a function's signature) do they selectively revisit the relevant passage. SSA instantiates this strategy: gist tokens maintain compact, continuously updated summaries of prior context, while selective unfolding enables targeted recovery of fine-grained information precisely when needed.

2.1 GIST-BASED RELEVANCE SCORING

The central observation of this work is that interleaved gist tokens, beyond their role as compressed summaries, can naturally serve as *selection proxies* for their associated chunks. Because each gist token g_m encodes the semantic content of chunk C_m, the attention affinity between a query and a gist token provides a direct, learned estimate of the chunk's relevance.

Concretely, at each decoding step t, given the current query $\mathbf{q}_t$, we compute a relevance score for each chunk C_m via the dot product between $\mathbf{q}_t$ and the gist key $\mathbf{k}_{g_m}$:

$$s_{t,m} = \mathbf{q}_t^\top \mathbf{k}_{g_m}. \tag{1}$$

This scoring mechanism is conceptually related to the gating function in MoBA (Lu et al., 2025), which computes the affinity between a query and the *mean-pooled* keys of each block: $s_i = \mathbf{q}^\top \overline{\mathbf{K}}_{[I_i]}$. However, there is a crucial difference. Mean pooling is a fixed, non-parametric operation: the block representation is simply the arithmetic average of the raw keys, which may wash out semantically important signals and has no learnable parameters. In contrast, gist tokens are *learned* representations optimized through training. This enables two advantages: (i) the gist token can capture rich semantic content beyond what a simple average reflects, and (ii) gradients flow through the gist tokens, allowing the model to jointly optimize the *quality of compression* and the *accuracy of selection*.

This scoring is also related to NSA's compressed attention branch (Yuan et al., 2025), which generates coarse-grained summary tokens via a dedicated MLP and uses the resulting attention scores to identify important fine-grained chunks. Our approach achieves a similar effect without requiring additional architectural components: the gist tokens are a part of the standard input sequence and reuse the existing attention mechanism.

2.2 Selective Unfolding and Hybrid Attention

Given the relevance scores, we select the top-k chunks with the highest scores:

$$\mathcal{I}_t = \text{Top-k}\big(\{s_{t,m}\}_{m=1}^{M}\big). \tag{2}$$

For selected chunks $m \in \mathcal{I}_t$, we *unfold* the compressed representation by reintroducing both the gist KV pair $(\mathbf{k}_{g_m}, \mathbf{v}_{g_m})$ and the full raw KV pairs $(\mathbf{K}_{C_m}, \mathbf{V}_{C_m})$ into the attention context. Unselected chunks are *entirely excluded* from the decoding context. The hybrid key and value matrices are thus constructed as:

$$\mathbf{K}_{\text{hybrid}} = \bigcup_{m \in \mathcal{I}_t} \big(\mathbf{k}_{g_m} \cup \mathbf{K}_{C_m}\big), \quad \mathbf{V}_{\text{hybrid}} = \bigcup_{m \in \mathcal{I}_t} \big(\mathbf{v}_{g_m} \cup \mathbf{V}_{C_m}\big). \tag{3}$$

The output is then computed via standard attention:

$$\mathbf{h}_t = \text{Attention}(\mathbf{q}_t,\ \mathbf{K}_{\text{hybrid}},\ \mathbf{V}_{\text{hybrid}}). \tag{4}$$

It is worth noting that we skip selective unfolding in the first attention layer, as all gist tokens share identical input hidden embeddings at this stage, rendering top-k selection uninformative.

We experimented with several variants of the hybrid context, including (i) retaining gist tokens for all chunks (both selected and unselected) as compact surrogates, and (ii) using only the raw tokens from selected chunks without any gist tokens in the context. Empirically, we found that attending exclusively to the selected gist-chunk pairs yields the best performance (see Appendix §I). We attribute this to the fact that, under a fixed token budget, replacing unselected gist tokens with unfolded raw tokens from selected gists provides a more informative attention context. The gist tokens of selected chunks are still retained alongside their raw tokens, as they provide complementary compressed information that captures cross-token patterns within the chunk.

During the experiments, rather than using a fixed Top-k across all settings, we set the k adaptively based on the sequence length, compression ratio, and the group structure of grouped-query attention (GQA) (Ainslie et al., 2023), ensuring a consistent compression ratio across different configurations. Details of the adaptive top-k formula are provided in Appendix §C.

A practical benefit of routing context through gist tokens is that decoding only needs the gist KV cache to be accessed, rather than the full per-token KV cache of the preceding context. This differs from many sparse attention methods: although they attend to only a selected subset of tokens, they often still need access to the full KV cache to score or identify which tokens should be selected. As a result, these methods either load the full cache during decoding or introduce an auxiliary module for token scoring. DSA (Liu et al., 2025), for example, creates a separate auxiliary indexer for this purpose. By contrast, in SSA, gist tokens directly serve as the selection interface, allowing decoding to operate over the gist cache and thereby reducing KV-cache transfer overhead.

Grouped Unfolding for GQA. In modern architectures employing GQA, multiple query heads share the same key-value cache. Since different heads within the same KV group may prefer different chunks, we perform top-k selection separately for each query head and then take the union of the selected chunk indices within the group. Let $\mathcal{H}(u)$ denote the set of query heads associated with KV group u. For each head $h \in \mathcal{H}(u)$, we compute $\mathcal{I}_t^{(h)} = \text{Top-k}\big(\{s_{t,m}^{(h)}\}_{m=1}^{M}\big)$ and define the group-level unfolded set as: $\mathcal{I}_t^{(u)} = \bigcup_{h \in \mathcal{H}(u)} \mathcal{I}_t^{(h)}$.

All chunks in $\mathcal{I}_t^{(u)}$ are then unfolded into the shared KV-cache for group u. This design preserves head-specific routing while maintaining consistency with the GQA sharing pattern, ensuring that the unfolded KV-cache can be loaded once and reused across all heads in the group. This is analogous to NSA's group-centric kernel design (Yuan et al., 2025), which loads shared KV blocks once per GQA group to maximize memory reuse and data locality. Additionally, by batching queries within each group and reusing KV blocks, it increases arithmetic intensity and enables more efficient utilization of Tensor Core matrix operations.

2.3 TRAINING

SSA is trained with standard teacher forcing under the autoregressive language modeling objective, using the vanilla cross-entropy loss[1]. The training consists of two stages: continued pretraining, which is required, and selective finetuning, which is optional. Both stages share the same sequence structure: each training sequence is split into two contiguous regions—a *compressed context* (the prefix) and a *generation context* (the suffix). The loss is computed only over the suffix. Both stages are implemented through attention mask design, with no separate prefill or decoding phases.

Stage 1: Continued Pretraining (Required). In the continued pretraining stage, the model learns to compress context into gist tokens. The compressed context (the prefix) is augmented with interleaved gist tokens and processed under the standard gist causal mask $\mathbf{M}_{\text{gist}}$ described in Appendix §B. The generation context tokens (the suffix) attend to the gist tokens produced by the compressed context, and to each other under the standard causal mask. The next token prediction loss over the generation context encourages the gist tokens to encode sufficient information for accurate next-token prediction. Since the entire forward pass is a single teacher-forced computation with a designed causal mask, training is fully parallelizable and requires no specialized kernels.

Importantly, after this stage alone, the model already supports selective unfolding at inference time. Because the gist tokens have learned to encode meaningful chunk-level summaries, they can be used directly as routing signals via equation 1–equation 2 without any further training. As shown in §3, this inference-time selective unfolding already yields substantial gains over standard gist compression (both appended and interleaved) as well as other inference-time-only sparse attention approaches.

Stage 2: Selective Finetuning (Optional). While continued pretraining alone enables effective inference-time selection, exposing the model to the selective unfolding mechanism during training can further improve performance. In the finetuning stage, the sequence is again split into compressed and generation contexts, but the attention mask for the generation context is now modified to incorporate Top-k selection. Concretely: The Compressed context is processed identically to continued pretraining, using the standard interleaved gist mask $\mathbf{M}_{\text{gist}}$. This region produces the gist token representations. For the generation context, for each query position t, we compute relevance scores against all gist tokens via equation 1 and select the Top-k chunks via equation 2. The attention mask for position t is then set to `true` only for the gist tokens and raw tokens of the selected chunks; all other compressed-context tokens are masked out.

Unlike continued pretraining, which employs a fixed, static attention mask, selective finetuning requires a position-dependent sparse mask where each query attends to a different subset of keys. Implementing this efficiently may require customized CUDA kernels. However, the sparse blockwise attention patterns involved are structurally similar to those in NSA (Yuan et al., 2025) and MoBA (Lu et al., 2025), and can leverage the same kernel design principles. We emphasize that this stage is optional: practitioners who wish to avoid the additional implementation effort can skip selective finetuning entirely and still benefit from inference-time selective unfolding after continued pretraining.

2.4 EXTENSION TO HIERARCHICAL GIST-OF-GIST COMPRESSION

The single-level framework described above achieves a linear per-step complexity of $O(M + kL)$ where $M = \lceil n/L \rceil$. We now show that the same principle—compressing representations and using them as routing signals—can be applied recursively, yielding a hierarchical structure with log-linear complexity. We refer to this extension as H-SSA.

The motivation is again grounded in how humans organize and retrieve information. Long documents are rarely flat sequences—they are structured hierarchically: a book is divided into chapters, chapters into paragraphs. When searching for a specific detail, a reader does not scan every sentence from the beginning. Instead, retrieval proceeds hierarchically: one first recalls a high-level memory (e.g., the chapter or topic), then refines it to a more specific section, and finally reconstructs the precise detail. Each level acts as a progressively sharper filter over memory. H-SSA mirrors this structure:

[1] Some related works instead use a KL-divergence loss against a teacher, which requires an additional forward pass with the full KV cache. This is expensive, and it caps the student's performance at the teacher's long-context ability, since the teacher's predictions serve as the upper bound.

meta-gist tokens encode coarse summaries, gist tokens represent finer summaries, and raw tokens preserve exact details. Hierarchical selection then implements this coarse-to-fine retrieval process, progressively narrowing attention from abstract memory to precise token-level information. We define the meta-gist tokens as follows.

Meta-Gist Tokens. Just as gist tokens summarize chunks of raw tokens, *meta-gist* tokens can be defined to summarize groups of gist tokens. We introduce a meta-gist token G_k after every J consecutive gist-chunk pairs. The meta-gist G_k summarizes the segment $\mathcal{S}_k = \{(C_j, g_j)\}_{j=(k-1)J+1}^{kJ}$, yielding $\lceil M/J \rceil$ meta-gist tokens in total. The augmented sequence becomes:

$$\tilde{\mathbf{X}} = \big(\underbrace{C_1, g_1, \ldots, C_J, g_J}_{\text{Segment 1}},\ G_1,\ \ldots,\ C_M, g_M,\ G_{\lceil M/J \rceil}\big). \tag{5}$$

Figure 1 illustrates the hierarchical attention pattern for a two-level gist hierarchy: tokens following a meta-gist G_k cannot attend to the raw tokens or gist tokens within the scope of G_k; they can only attend to G_k itself. This blocking mechanism forces long-range history to be accessed exclusively through the meta-gist representations. The construction generalizes to arbitrary depth t by introducing additional levels of summary tokens, each summarizing J groups from the level below. This means SSA can theoretically support unbounded context lengths: as the sequence grows, additional hierarchy levels are added to maintain a fixed memory footprint at the top level, with the full detail always recoverable through selective unfolding. Throughout this paper, we use the term *summary tokens* to refer collectively to gist tokens and meta-gist tokens at all levels of the hierarchy, distinguishing them from the raw tokens of the original sequence.

Coarse-to-Fine Selection. The hierarchical structure enables a coarse-to-fine selection strategy that mirrors the single-level mechanism at each scale. Instead of computing relevance scores against all M gist tokens, the model first scores the $\lceil M/J \rceil$ meta-gist tokens, selects the Top-k_2 segments, and then scores only the $k_2 \cdot J$ gist tokens within those segments to identify the final Top-k_1 chunks for unfolding. This reduces the per-query routing cost from $O(M)$ to $O(M/J + k_2 \cdot J)$. For a hierarchy of depth t, the per-query routing cost generalizes to $O(t \cdot M^{1/t})$. When $t = O(\log_J M)$, this reduces to $O(J \cdot \log_J M) = O(\log M)$—that is, each query's selection cost is *logarithmic* in the context length. After selection, the query attends to the selected tokens at *every* level of the hierarchy: the k_t selected top-level summary tokens, the k_{t-1} selected tokens at the next level, down to the k_1 selected gist tokens and their $k_1 \cdot L$ unfolded raw tokens. The total attention context thus contains $\sum_{\ell=1}^{t} k_\ell + k_1 \cdot L$ tokens. Since each k_ℓ is a small constant and $t = O(\log_J M)$, the summary tokens contribute $O(\log M)$ and the raw tokens contribute $O(k_1 L)$. The per-query attention cost is therefore $O(k_1 L + \log M)$, which matches the routing cost in order. The total per-query decoding cost is $O(\log M) = O(\log n)$, logarithmic in the context length. Over the full sequence, the aggregate decoding cost is $O(n \log n)$, which is log-linear.

Table 1 summarizes the complexity comparison. We provide a more detailed analysis in Appendix §D.

Table 1: Complexity comparison. n: context length, L: chunk size, k: number of selected chunks, J: meta-gist grouping factor. All quantities treat d (head dimension) as a constant. For the hierarchical variant, the depth is set to $t = \lceil \log_J M \rceil$ where $M = n/L$.

Method	Per-step Decoding	Prefill
Full Attention	$O(n)$	$O(n^2)$
SSA (single-level)	$O(n/L + kL)$	$O(nL + n^2/L^2)$
SSA (hierarchical)	$O(\log n + kL)$	$O(nL + nJ/L)$

2.5 Kernel Design

SSA replaces a dense attention operator with a sparse one, but sparsity only pays off if the kernel makes it physical: skipped tokens must translate into skipped FLOPs in prefill and skipped memory traffic in decoding. These two phases have opposite cost structures—prefill is compute-bound ($O(n^2)$ over the full query block), while decoding is bound by memory bandwidth and kernel-launch overhead (q_{len}=1 per step). As such, we design a separate kernel for each. Both kernels compute the exact attention defined in §3.1–§3.3, up to differences in floating-point accumulation order. Appendix §E details their design for the two-level hierarchy (H-SSA), with single-level as a special case.

3 EXPERIMENTS

3.1 EXPERIMENT SETUP

We evaluate SSA on two model families of different scales: Qwen2-7B-Instruct (Yang et al., 2024a)[2] and Llama3.2-1B (Meta AI, 2024)[3]. This allows us to assess the generality of our approach across both a large instruction-tuned model and a smaller base model.

We compare against three categories of methods: **(1) Inference-time sparse attention:** LongLLMLingua (Jiang et al., 2024), H_2O (Zhang et al., 2023b), StreamingLLM (Xiao et al., 2023), and Quest (Tang et al., 2024). These methods apply KV-cache eviction or selection to models pretrained with full attention, without any additional training. **(2) Retrieval-based context selection:** BM25 retrieval (Robertson & Zaragoza, 2009), which selects query-relevant context chunks under the same token budget as SSA and feeds them to the unmodified pretrained model. This baseline requires no additional training and isolates explicit query-adaptive context selection from learned sparse attention. **(3) Gist-based compression:** ActivationBeacon (Zhang et al., 2024) (interleaved gist), UniGist (Deng et al., 2025) (interleaved gist), and KVLink (Yang et al., 2025) (appended gist). These methods require continued pretraining and/or finetuning. **(4) Full attention references:** We include the original pretrained model (no compression), continued pretraining with full attention (Full-PT), and finetuning with full attention (Full-FT) as the references. We do not include comparisons with NSA, DSA, or MoBA, as these approaches are not directly compatible with our setting: NSA requires architectural modifications, DSA depends on an external indexer, and MoBA is continued pretraining and sparse decoding limited to prefill-only scenarios (Lu et al., 2025).

We evaluate our method under two primary settings: (1) **Long-context compression** on LongBench (Bai et al., 2024); and (2) **Retrieval-Augmented Generation (RAG)** on five multi-document QA benchmarks. Following NSA (Yuan et al., 2025), we exclude a subset of LongBench tasks that consistently yield low scores across models, as they offer limited discriminatory power. The key distinction between these settings lies in how context is structured. The long-context setting treats each sample as a single continuous sequence, while in the RAG setting, each sample consists of documents with explicit boundaries, and gist tokens are inserted independently within each document.

Beyond these evaluations, we present ablation studies on the composition of the hybrid attention context and compare adaptive Top-k selection with Top-p thresholding (Appendix §I). We further evaluate SSA and H-SSA on passkey retrieval across varying context lengths (Appendix §G), as well as on multi-document RAG with KV-cache reuse (Appendix §H). Finally, we assess prefill and decoding efficiency and scalability at both the end-to-end and attention-operator levels (Appendix §J).

3.2 LONG-CONTEXT RESULTS

Table 2 reports results on LongBench using Qwen2-7B-Instruct across three compression ratios ($8\times$, $16\times$, $32\times$), under both continued pretraining and finetuning. We omit H-SSA at $8\times$ compression, as the chunk size is already small. For H-SSA at $16\times$, we set $L = 4$ and $J = 4$—inserting a gist token every 4 tokens and a meta-gist token every 4 gist tokens. At $32\times$, we use $L = 8$ and $J = 4$—inserting a gist token every 8 tokens.

SSA consistently outperforms gist baselines. Under continued pretraining, SSA achieves an average of 46.20 across tasks at $8\times$ compression, outperforming ActivationBeacon (42.52) and UniGist (43.40) by 3.7 and 2.8 points respectively, and nearly matching the Full-PT model (47.78). The advantage is maintained across higher compression ratios: at $16\times$, SSA scores 45.39 versus 40.64 (ActivationBeacon) and 40.89 (UniGist); at $32\times$, SSA achieves an average of 44.07 versus 38.30 and 38.16. This demonstrates that selective unfolding recovers critical fine-grained information that standard compression discards uniformly.

Selective finetuning further improves performance. With finetuning, SSA continues to lead across all settings. At $8\times$, it achieves 46.74 (vs. 42.67 for ActivationBeacon and 43.33 for UniGist), approaching the Full-FT model (47.32). Notably, on several individual tasks SSA surpasses the full-attention baseline: for instance, on MF-en task at $8\times$ finetuning, SSA scores 54.24 compared to Full-FT model's 50.33, suggesting that the selective unfolding mechanism can act as a beneficial inductive bias.

[2]We choose Qwen2 because ActivationBeacon—an important baseline—has been evaluated on it.

[3]We choose Llama3.2-1B because KVLink—an important baseline—has been evaluated on it.

Hierarchical variant excels at high compression. H-SSA demonstrates its advantage at higher compression ratios where the single-level variant's routing cost becomes more significant. At $16\times$ with finetuning, H-SSA achieves 46.48, surpassing SSA's 45.26. At $32\times$ with finetuning, the gap widens: H-SSA scores 44.94 versus SSA's 43.35. This validates that hierarchical gist-of-gist compression and coarse-to-fine selection become more valuable as the compression ratio grows.

Table 2: Performance comparison on LongBench under different compression ratios ($8\times$, $16\times$, $32\times$) using Qwen2-7B-Instruct. Best results are in bold, second-best are underlined.

	Methods	Single-Document QA			Multi-Document QA			Few-shot		Summarization	Avg.
		NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	SAMSum	TriviaQA	GovRep	
–	Qwen2-7B-Inst.	26.33	42.30	44.50	54.39	55.41	36.19	45.19	89.91	36.16	47.82
	Inference-time Only										
x8	LongLLML.	15.10	20.21	22.16	23.74	28.81	9.94	39.77	85.98	20.51	29.58
	H_2O	23.17	37.99	40.48	50.66	50.60	32.05	41.33	88.49	32.99	44.20
	StreamLLM	22.95	29.16	25.31	44.78	44.56	27.37	42.84	88.99	23.30	38.81
	QUEST	11.56	13.35	24.13	10.89	15.06	11.06	19.34	36.91	13.56	17.32
	Continued Pretraining										
–	Full-PT	26.59	38.37	46.39	57.56	56.28	36.12	45.83	89.06	33.78	47.78
x8	ActBeacon	18.23	35.40	31.69	48.87	**54.45**	30.89	**45.30**	87.59	30.24	42.52
	UniGist	22.88	32.92	36.52	55.14	51.10	31.73	43.02	89.88	27.45	43.40
	SSA	**23.55**	**36.84**	**42.21**	**61.02**	54.23	**32.47**	44.59	**90.56**	**30.36**	**46.20**
	H-SSA	–	–	–	–	–	–	–	–	–	–
x16	ActBeacon	19.08	32.45	28.57	46.47	47.93	31.67	**45.50**	85.02	29.06	40.64
	UniGist	21.68	28.90	35.57	51.97	46.95	27.55	43.56	89.01	22.80	40.89
	SSA	**24.30**	**33.86**	**43.76**	55.70	**51.46**	**34.17**	44.32	**91.02**	29.94	**45.39**
	H-SSA	21.89	31.11	37.46	**56.01**	50.98	30.25	44.63	89.41	**31.05**	43.64
x32	ActBeacon	19.90	27.92	24.71	43.81	46.95	28.13	**44.70**	83.16	25.42	38.30
	UniGist	18.68	28.89	26.19	48.67	43.21	25.73	43.43	88.49	20.17	38.16
	SSA	**22.61**	**34.03**	37.08	**54.19**	**52.03**	30.53	44.42	**90.40**	**31.31**	**44.07**
	H-SSA	16.16	30.82	**39.30**	53.66	49.08	**30.62**	44.35	89.17	29.89	42.56
	Finetuning										
–	Full-FT	27.84	38.56	50.33	52.26	54.87	35.25	43.70	87.99	35.07	47.32
x8	ActBeacon	**26.85**	36.95	47.93	48.91	41.27	22.54	**44.61**	87.32	27.62	42.67
	UniGist	24.17	38.27	47.28	54.02	41.26	24.84	43.63	89.92	26.59	43.33
	SSA	26.24	**43.54**	**54.24**	**54.26**	**48.18**	**28.95**	44.31	**90.11**	**30.79**	**46.74**
	H-SSA	–	–	–	–	–	–	–	–	–	–
x16	ActBeacon	23.81	35.77	38.91	44.29	41.69	22.76	**46.39**	86.06	27.80	40.83
	UniGist	23.15	34.91	40.54	46.79	38.67	22.82	43.82	89.97	23.15	40.42
	SSA	**24.38**	42.47	51.98	54.69	42.91	28.59	42.66	88.43	**31.27**	45.26
	H-SSA	22.72	**43.98**	**53.40**	**56.11**	**48.31**	**29.04**	43.28	**90.36**	31.12	**46.48**
x32	ActBeacon	21.32	32.52	33.18	41.05	42.44	19.10	43.04	85.66	28.33	38.52
	UniGist	19.96	31.98	33.87	40.34	37.63	20.47	**43.43**	**89.56**	21.32	37.62
	SSA	**24.58**	40.24	45.68	50.64	39.38	27.55	42.69	89.10	30.30	43.35
	H-SSA	22.03	**43.32**	**50.79**	**52.47**	**45.18**	**27.93**	42.56	89.47	**30.68**	**44.94**

3.3 RAG RESULTS

Table 3 reports results on five multi-document QA benchmarks using Llama-3.2-1B across two compression ratios ($8\times$, $16\times$), under both continued pretraining and finetuning.

The RAG setting reveals the most striking advantage of SSA. At $8\times$ compression with continued pretraining only, SSA achieves 33.68 average score, not only outperforming KVLink (21.58) and UniGist (22.53) by over 11 points, but **even surpassing the vanilla (27.99) and Full-PT model (27.14).** We attribute this to the nature of RAG: each query is relevant to only one of the retrieved documents, while the remaining documents act as distractors. Full attention distributes capacity uniformly across all documents, diluting focus on the relevant passage. In contrast, SSA's selective unfolding mechanism concentrates attention on the most query-relevant chunks, effectively filtering out distracting documents.

Query-adaptive selection alone does not explain the gain. BM25 receives the same token budget as SSA and is likewise query-adaptive, yet it does not reproduce SSA's behavior: at 8× under continued

pretraining it averages 26.60, below both the vanilla model (27.99) and Full-PT (27.14), while SSA reaches 33.68. The gap widens under finetuning (41.08 vs. 53.39 at $8\times$, 35.89 vs. 49.29 at $16\times$). This sharpens our interpretation: distractor filtering is indeed the mechanism, but performing it in representation space is what lets compression exceed full attention rather than merely approach it.

After finetuning, SSA at $8\times$ achieves 53.39, approaching Full-FT (57.76). This represents an improvement of 11.94 points over KVLink (41.45) and 8.12 points over UniGist (45.27). At $16\times$, SSA scores 49.29, outperforming KVLink's 38.67 and UniGist's 42.24.

Consistent with the long-context results, H-SSA outperforms the single-level variant at $16\times$ compression. Under continued pretraining, H-SSA achieves 30.19, compared to 27.74 for SSA. After finetuning, H-SSA reaches 50.72 versus 49.29 for SSA, suggesting that the coarse-to-fine hierarchical selection more effectively allocates the limited token budget.

A notable pattern in the RAG results is that Full-PT underperforms SSA under continued pretraining, yet surpasses it after finetuning. This is because continued pretraining optimizes a generic next-token prediction objective over the given context, without any notion of a query or a target answer. The model has no signal to distinguish relevant from irrelevant documents, and thus learns to attend to all content indiscriminately. Finetuning, by contrast, trains the model on the downstream task format—given multiple documents and a question, produce an answer—which explicitly teaches it to focus on query-relevant passages and suppress distractors.

Table 3: Performance comparison on RAG under different compression ratios ($8\times$, $16\times$) using Llama3.2-1B. Best results are in bold, second-best are underlined.

	Methods	HotpotQA	TriviaQA	2WikiMQA	MuSiQue	NQ	Avg.
–	Llama3.2-1b	29.98	40.82	25.86	9.10	34.20	27.99
	Continued Pretraining						
–	Full-PT	29.64	39.73	25.45	9.18	31.72	27.14
x8	BM25	<u>30.75</u>	43.70	<u>25.60</u>	<u>8.27</u>	**24.70**	<u>26.60</u>
	KVLink	17.54	44.36	25.03	4.01	16.96	21.58
	UniGist	22.77	<u>44.44</u>	22.84	3.72	18.86	22.53
	SSA	**41.09**	**52.06**	**38.65**	**12.99**	<u>23.62</u>	**33.68**
	H-SSA	–	–	–	–	–	–
x16	BM25	27.68	41.55	24.88	6.29	18.86	23.85
	KVLink	10.67	33.10	21.22	0.74	4.92	14.13
	UniGist	21.51	41.62	30.10	3.23	14.56	22.20
	SSA	<u>30.56</u>	<u>46.28</u>	<u>32.67</u>	<u>8.11</u>	<u>21.10</u>	<u>27.74</u>
	H-SSA	**34.34**	**50.98**	**33.61**	**9.06**	**22.98**	**30.19**
	Finetuning						
–	Full-FT	63.82	71.38	75.83	24.20	53.56	57.76
x8	BM25	48.28	61.22	44.89	<u>16.59</u>	34.42	41.08
	KVLink	41.15	61.65	62.68	10.47	31.28	41.45
	UniGist	<u>49.25</u>	<u>64.00</u>	<u>64.17</u>	12.37	<u>36.56</u>	<u>45.27</u>
	SSA	**59.88**	**68.62**	**71.99**	**20.27**	**46.20**	**53.39**
	H-SSA	–	–	–	–	–	–
x16	BM25	41.57	55.36	44.24	10.96	27.34	35.89
	KVLink	36.92	58.85	62.21	8.11	27.24	38.67
	UniGist	43.50	61.83	60.72	10.88	34.26	42.24
	SSA	<u>54.29</u>	<u>66.16</u>	<u>67.93</u>	<u>16.30</u>	<u>41.78</u>	<u>49.29</u>
	H-SSA	**56.43**	**66.76**	**69.56**	**18.37**	**42.48**	**50.72**

4 CONCLUSIONS

We presented SSA, a framework that unifies gist-based context compression with sparse attention through selective unfolding. By leveraging interleaved gist tokens as both compressed summaries and learned routing signals, SSA enables query-adaptive recovery of fine-grained detail without architectural modifications, external indexers, or non-differentiable gates. The framework naturally extends to hierarchical gist-of-gist compression, achieving logarithmic per-step decoding complexity through coarse-to-fine selection. Experiments on LongBench and RAG benchmarks demonstrate consistent gains over both gist compression and sparse attention baselines across compression ratios from $8\times$ to $32\times$.

5 AI USE STATEMENT

In this work, we used generative AI tools for coding assistance and for writing polish. Specifically, LLM-based coding assistants were used to help implement and debug parts of the training and evaluation pipeline. They were also used to improve the grammar and clarity of the manuscript. We have not used generative AI tools for research ideation, experimental design, or the derivation of the complexity analysis. We have reviewed all AI-assisted work. All AI-assisted code was read line by line, and the correctness of the sparse kernels was verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of AI.

6 ETHICS STATEMENT

This work studies an efficiency-oriented training and inference method for language models. All experiments use publicly available models (Qwen2-7B-Instruct, Llama-3.2-1B) and publicly available datasets (RedPajama, FineWeb, LongBench, and RAG benchmarks), used in accordance with their respective licenses and intended research use. No human subjects were involved, and no new data was collected or annotated; to our knowledge the datasets used contain no personally identifiable or offensive content beyond what has been documented by their original authors.

7 REPRODUCIBILITY STATEMENT

We have aimed to make all components of SSA reproducible from the paper alone. The method is described in full in §2: the gist-based relevance score, top-k selection and hybrid-context construction, grouped unfolding for GQA, and the two training stages. The attention masks used for continued pretraining are specified and Appendix §2.5 documents the prefill and decoding kernel designs.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

AlphaEvolve team. AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms. Google DeepMind Blog, May 2025. URL https://deepmind.google/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/. Accessed: 2026-06-03.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. PyTorch Blog, October 2023. URL https://pytorch.org/blog/flash-decoding/. Accessed: 2026-06-25.

Chenlong Deng, Zhisong Zhang, Kelong Mao, Shuaiyi Li, Tianqing Fang, Hongming Zhang, Haitao Mi, Dong Yu, and Zhicheng Dou. Unigist: Towards general and hardware-aligned sequence-level long context compression. *arXiv preprint arXiv:2509.15763*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arxiv. *arXiv preprint arXiv:2312.00752*, 10, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

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

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

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression, 2024. URL https://arxiv.org/abs/2310.06839.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. In *Findings of the association for computational linguistics: EMNLP 2022*, pp. 6536–6558, 2022.

Michael Y. Li, Jubayer Ibn Hamid, Emily B. Fox, and Noah D. Goodman. Neural garbage collection: Learning to forget while learning to reason, 2026. URL https://arxiv.org/abs/2604.18002.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, et al. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025.

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

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Matei Zaharia, Alexandros G. Dimakis, and Ion Stoica. Skydiscover: A flexible, adaptive framework for ai-driven scientific and algorithmic discovery. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 1223–1227. Association for Computing Machinery, 2026. doi: 10.1145/3786335.3813221. URL https://doi.org/10.1145/3786335.3813221.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, September 2024. Accessed: 2026-03-29.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, et al. Rwkv: Reinventing rnns for the transformer era. In *Findings of the association for computational linguistics: EMNLP 2023*, pp. 14048–14077, 2023.

Aleksandar Petrov, Mark Sandler, Andrey Zhmoginov, Nolan Miller, and Max Vladymyrov. Long context in-context compression by getting to the gist of gisting. *arXiv preprint arXiv:2504.08934*, 2025.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Jiaxin Shi, Ke Alexander Wang, and Emily B. Fox. Sequence modeling with multiresolution convolutional memory. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

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

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin

Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024a. URL https://arxiv.org/abs/2407.10671.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. *arXiv preprint arXiv:2502.16002*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2405.15793.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024c.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. 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, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, 2023a.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon. *arXiv preprint arXiv:2401.03462*, 2024.

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

A Related Works

Sparse Attention Mechanisms. Addressing the quadratic complexity of the self-attention mechanism has been a central theme in scaling Transformers to long contexts. Early approaches introduced fixed or heuristic sparsity patterns. Sparse Transformer (Child, 2019) and Longformer (Beltagy et al., 2020) utilize fixed local windows combined with global attention tokens to reduce complexity to $O(N\sqrt{N})$ or $O(N)$. Many recent KV-cache management works such as H_2O (Zhang et al., 2023b), StreamingLLM (Xiao et al., 2023), LongLLMLingua (Jiang et al., 2024), and Quest (Tang et al., 2024) aim to evict less important KV pairs. More recently, hardware-aware and learnable sparse attention mechanisms—such as Native Sparse Attention (Yuan et al., 2025), DeepSeek Sparse Attention (Liu et al., 2025), and MoBA (Lu et al., 2025)—have been proposed to dynamically identify and attend to the most relevant blocks in a hardware-efficient manner.

Context Compression. An alternative approach to efficient long-context processing is compressing the context into compact representations. Gist Tokens (Mu et al., 2023) and AutoCompressors (Chevalier et al., 2023) train models to compress prompt instructions into special tokens via an attention mask bottleneck. KVLink (Yang et al., 2025) appends learnable compression tokens to the end of each document, where each token can attend to the entire preceding context to produce a global summary. ActivationBeacon (Zhang et al., 2024), UniGist (Deng et al., 2025), and GistPool (Petrov et al., 2025) interleave gist tokens throughout the sequence, associating each with a local chunk. While these methods effectively reduce sequence length, they typically treat compression as a one-way process: once the context is compressed, the raw tokens are discarded and irreversibly lost.

Alternative Sequence Modeling Architectures. A separate line of research seeks to replace the softmax attention mechanism entirely with architectures that achieve linear or near-linear complexity. MultiresLayer (Shi et al., 2023) captures multiscale trends in the input sequence using convolutional networks. Linear attention methods (Katharopoulos et al., 2020) approximate the softmax kernel with feature maps, enabling recurrent computation with a fixed-size state. State space models such as S4 (Gu et al., 2021) and Mamba (Gu & Dao, 2023) model sequences through structured recurrences with selective gating, achieving linear-time inference with competitive language modeling performance. Recurrent alternatives including RWKV (Peng et al., 2023) and RetNet (Sun et al., 2023) combine recurrent architectures with attention-like mechanisms, offering constant-memory inference. More recent gated linear attention variants such as GLA (Yang et al., 2024c), refine selective forgetting through per-token gates. These approaches represent a fundamentally different design axis from our work. They modify or replace the core attention mechanism with alternative sequence modeling primitives, often requiring pretraining from scratch on the new architecture.

Memory-Augmented Language Models. A complementary line of research equips language models with external memory modules to handle contexts that exceed their native attention window. LongMem (Wang et al., 2023) decouples memory encoding from retrieval by maintaining a frozen backbone alongside a trainable side network that retrieves cached key–value pairs from past inputs. MemGPT (Packer et al., 2023) treats the LLM as an operating system that explicitly pages information between a limited context window and an external store, enabling long-horizon reasoning over conversations and documents. More recent systems such as Mem0 (Chhikara et al., 2025) and MemOS (Li et al., 2025) extend this idea by introducing structured, persistent memory layers with mechanisms for adding, updating, and consolidating memories across sessions. From this perspective, SSA can itself be viewed as a memory mechanism: each gist token is a learned latent summary of a local chunk that is stored in the KV-cache and later retrieved on demand via attention-based scoring, while meta-gist tokens form a hierarchical memory index that supports coarse-to-fine recall—closely paralleling the multi-tier memory hierarchies of MemGPT and MemOS.

B Preliminaries

Sparse Attention. In standard attention, each query $\mathbf{q}_t \in \mathbb{R}^d$ attends to all preceding key-value pairs. As the sequence length grows, this scales quadratically in t. Sparse attention methods reduce

this cost by restricting each query to attend to a subset $\mathcal{S}_t \subset \{1, \ldots, t\}$ of positions:

$$\mathbf{o}_t = \sum_{i \in \mathcal{S}_t} \alpha_{t,i}\, \mathbf{v}_i, \quad \alpha_{t,i} = \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_i / \sqrt{d})}{\sum_{j \in \mathcal{S}_t} \exp(\mathbf{q}_t^\top \mathbf{k}_j / \sqrt{d})}, \tag{6}$$

where $|\mathcal{S}_t| \ll t$. The core challenge lies in how to construct $\mathcal{S}_t$: fixed patterns such as sliding windows or stride-based selection impose structural priors that may miss relevant tokens, while dynamic methods that adapt $\mathcal{S}_t$ to each query require an efficient mechanism for estimating token relevance without computing full attention.

Interleaved Gist Tokens. Following prior work on gist-based context compression (Mu et al., 2023; Zhang et al., 2024; Petrov et al., 2025; Deng et al., 2025), we partition the input sequence $\mathbf{X} = (x_1, \ldots, x_n)$ into $M = \lceil n/L \rceil$ non-overlapping chunks of length L: $\mathcal{C} = \{C_1, \ldots, C_M\}$, where $C_m = (x_{(m-1)L+1}, \ldots, x_{mL})$. A learnable gist token g_m is appended to each chunk, yielding the augmented sequence:

$$\tilde{\mathbf{X}} = (C_1,\ g_1, C_2,\ \ldots,\ C_M, g_M). \tag{7}$$

As illustrated in Figure 1, a causal mask $\mathbf{M}_{\text{gist}}$ is applied such that tokens within chunk C_m can attend to tokens within C_m and to all preceding gist tokens $g_{<m}$, while tokens following g_m cannot attend to raw tokens within C_m's scope—only to g_m and the preceding gist tokens $g_{<m}$. This creates an information bottleneck that forces each gist token to learn a compressed representation of its corresponding chunk. After a forward pass, the embedding $\mathbf{g}_m$ summarizes C_m, and its associated KV pair $(\mathbf{k}_{g_m}, \mathbf{v}_{g_m})$ can then replace the full KV-cache of C_m during decoding.

C More Details on Adaptive Top-K.

Concretely, let n_{kv} denote the total number of KV positions available for the generation context, L_{eff} the effective compression factor (equal to the chunk size L for single-level SSA, or $L \cdot J$ for the hierarchical variant H-SSA), and G the number of query heads per KV group. The per-level selection budget is computed as:

$$k = \left\lfloor \frac{n_{\text{kv}}}{L_{\text{eff}} \cdot G \cdot L} \right\rfloor + 1, \tag{8}$$

where the division by G accounts for the grouped unfolding mechanism (§2.2): since each head within a GQA group may select different chunks and their union is unfolded into the shared KV-cache, the effective number of unfolded chunks per group scales with G. The $+1$ ensures that at least one chunk is always selected. For the hierarchical variant, the same k is used at both the meta-gist and gist levels. This adaptive scheme ensures that the total number of unfolded tokens remains approximately proportional to n_{kv} regardless of the compression ratio, providing a fair comparison across settings.

D Complexity Analysis

We provide a detailed complexity analysis for both the single-level and hierarchical variants of SSA. Let n denote the input length, and d the head dimension (treated as a constant).

D.1 Single-Level Complexity

Given chunk size L, the number of chunks (and thus gist tokens) is $M = \lceil n/L \rceil$, and let k denote the number of chunks selected for unfolding.

Prefill. During prefill, the computation decomposes into two parts:

1. **Intra-chunk attention.** Each chunk C_m has L raw tokens plus one gist token. Under the causal mask, tokens within C_m attend to at most $L + 1$ positions. Across all M chunks, this costs $O(M \cdot L \cdot (L + 1) \cdot d) = O(nL)$, since $M = \lceil n/L \rceil$ and d is constant.
2. **Gist-to-gist attention.** Each gist token g_m can attend to all preceding gist tokens $g_1, \ldots, g_{m-1}$. The total cost across all M gist tokens is $\sum_{m=1}^{M} m = O(M^2) = O(n^2/L^2)$.

The overall prefill cost is therefore $O(nL + n^2/L^2)$. For typical settings where $L = O(\sqrt{n})$, both terms are $O(n\sqrt{n})$, yielding sub-quadratic prefill. For fixed L, the gist-to-gist term $O(n^2/L^2)$ dominates, which is quadratic but reduced by a factor of L^2 compared to full attention.

Decoding (per step). At each decoding step t, the computation involves:

1. **Relevance scoring.** The query $\mathbf{q}_t$ computes dot products with all M gist keys: $s_{t,m} = \mathbf{q}_t^\top \mathbf{k}_{g_m}/\sqrt{d}$ for $m = 1, \ldots, M$. This costs $O(M \cdot d) = O(M)$.
2. **Top-k selection.** Selecting the k highest scores from M candidates costs $O(M)$ (via a partial sort).
3. **Hybrid attention.** The query attends to k gist tokens and $k \cdot L$ raw tokens from the selected chunks, totaling $k(1+L)$ key-value pairs. The attention computation costs $O(k(1+L)\cdot d) = O(kL)$.

The total per-step decoding cost is:

$$\underbrace{O(M)}_{\text{scoring}} + \underbrace{O(M)}_{\text{top-}k} + \underbrace{O(kL)}_{\text{attention}} = O(M + kL) = O(n/L + kL)\,. \tag{9}$$

Since k and L are fixed constants independent of n, this is *linear* in the context length n. Over a full sequence of n decoding steps, the aggregate decoding cost is $O(n^2/L + nkL)$, which is quadratic in n but reduced by a factor of L compared to full attention's $O(n^2)$.

D.2 Hierarchical Complexity

We now analyze the hierarchical variant with meta-gist grouping factor J and hierarchy depth t. At each level $\ell \in \{1, \ldots, t\}$, there are $M_\ell = \lceil M/J^{\ell-1} \rceil$ summary tokens: $M_1 = M$ gist tokens at the bottom, $M_2 = \lceil M/J \rceil$ meta-gist tokens at level 2, and so on up to $M_t = \lceil M/J^{t-1} \rceil$ tokens at the top.

Prefill. The blocking mechanism ensures that tokens at level ℓ can only attend to summary tokens at the same or higher levels within their local scope. The computation decomposes as:

1. **Intra-chunk attention.** Same as the single-level case: $O(nL)$.
2. **Level-ℓ summary attention.** At level ℓ, each of the M_ℓ summary tokens attends to at most J tokens from level $\ell - 1$ (its children) plus the preceding summary tokens at level ℓ. Due to the blocking mechanism, each level-ℓ token's effective attention span is bounded by $O(J + M_\ell)$. However, the blocking ensures that level-ℓ tokens only see level-ℓ peers within their parent's scope at level $\ell + 1$, bounding the span to $O(J)$ per token. The total cost at level ℓ is $O(M_\ell \cdot J)$. Summing across all levels:

$$\sum_{\ell=1}^{t} O(M_\ell \cdot J) = O\left(J \sum_{\ell=1}^{t} \frac{M}{J^{\ell-1}}\right) = O\left(MJ \cdot \frac{1 - J^{-t}}{1 - J^{-1}}\right) = O(MJ). \tag{10}$$

   The geometric series converges since $J \geq 2$, so the total summary attention cost is $O(MJ) = O(nJ/L)$, independent of the depth t.

The overall prefill cost is therefore $O(nL + nJ/L)$, which is *linear* in n—a substantial improvement over the single-level prefill cost of $O(nL + n^2/L^2)$.

Decoding: Routing. Coarse-to-fine selection proceeds top-down through the hierarchy. At each level ℓ, the model selects k_ℓ summary tokens from the candidates exposed by the previous level's selection:

1. **Level t (top).** Score all $M_t = \lceil M/J^{t-1} \rceil$ top-level tokens and select the top-k_t. Cost: $O(M_t)$.
2. **Level $\ell < t$.** The $k_{\ell+1}$ selected tokens at level $\ell + 1$ expose $k_{\ell+1} \cdot J$ candidate tokens at level ℓ. Score these and select the top-k_ℓ. Cost: $O(k_{\ell+1} \cdot J)$.

The total routing cost is:

$$\underbrace{O(M_t)}_{\text{top level}} + \sum_{\ell=1}^{t-1} \underbrace{O(k_{\ell+1} \cdot J)}_{\text{level } \ell} = O\left(\frac{M}{J^{t-1}} + (t-1) \cdot k_{\max} \cdot J \right), \tag{11}$$

where $k_{\max} = \max_\ell k_\ell$. When $t = \lceil \log_J M \rceil$, the first term becomes $O(M/J^{\log_J M - 1}) = O(J)$, a constant. The second term is $O(k_{\max} \cdot J \cdot \log_J M)$. Since $k_{\max}$ and J are fixed constants, the total routing cost is:

$$O(J + k_{\max} J \log_J M) = O(\log M) = O(\log(n/L)) = O(\log n). \tag{12}$$

Decoding: Attention. After routing, the query attends to the selected tokens at every level of the hierarchy: k_ℓ selected summary tokens at each level $\ell \in \{2, \ldots, t\}$, plus k_1 selected gist tokens and their $k_1 \cdot L$ unfolded raw tokens at the bottom level. The total number of key-value pairs in the attention context is $\sum_{\ell=1}^{t} k_\ell + k_1 \cdot L$. Since each k_ℓ is a small constant and $t = O(\log_J M)$, the summary tokens contribute $O(\log M)$ and the raw tokens contribute $O(k_1 L)$. The attention cost is therefore $O(k_1 L + \log M)$.

Decoding: Total. Combining routing and attention, the per-step decoding cost is:

$$O(\log n + k_1 L) = O(\log n), \tag{13}$$

which is *logarithmic* in the context length. Over a full sequence of n decoding steps, the aggregate cost is $O(n \log n)$, which is *log-linear*.

E More Details on Kernel Design

Prefill: block-sparse attention via key-column permutation. Under the gist causal mask $\mathbf{M}_{\text{gist}}$, each token attends to its local chunk, an attention sink, and all preceding summary tokens. The difficulty is that this sparsity is unstructured: summary columns recur every L tokens throughout the sequence, so under the 128×128 tiling of a standard block-sparse kernel, every key block may contain summary columns visible to all later queries, no block is empty, and nothing is skipped. We restore structure with a key-column permutation. Before attention, we measure each key column's visibility frequency across queries and label high-frequency columns (summary tokens and the sink, visible to most queries) as global; we then permute the key/value sequence so all global columns come first, followed by the local columns. Because softmax normalizes over the key axis, attention is invariant under any permutation of keys, so this is exact. On the permuted layout, the same mask becomes block-structured: a thin dense slab of global columns ($q_{\text{len}} \times O(n/L_{\text{eff}})$) plus a diagonal band for the local windows, with everything else empty. We materialize this as a functional block mask and dispatch block-sparse FlexAttention, which skips the empty blocks. The permutation and block mask depend only on the mask, not on layer weights, so they are computed once per prefill and shared across all layers; only the per-layer K/V gather differs.

Decoding: selection bookkeeping without mask materialization. At each decoding step, the hybrid context of equation 3 is fully determined by a small set of $O(n)$ metadata arrays—per-level chunk identities, gist/meta-gist indicator flags, a compressed-region flag, and the always-attended suffix—together with the per-group selection bitmaps of size $[\,\#\text{KV groups}, M{+}1\,]$ produced by grouped unfolding (§3.3). A naïve implementation expands these into a dense $[H, n]$ attention mask every layer and runs masked SDPA over the full KV cache, which forfeits the entire benefit of sparsity: the bytes are still moved. Our decode kernel instead evaluates the keep predicate on the fly inside the kernel—a key i in KV group u is kept iff it is a raw token of a selected chunk in the compressed region, a selected gist or meta-gist token, or part of the uncompressed suffix—reading only the byte-sized metadata arrays. These arrays are query-independent within a step, so they are computed once and reused by all layers.

Decoding: three-kernel split-K sparse flash-decode. The decode operator is factorized into three phases, each parallelized independently. *(i) Parallel compaction.* A first kernel scans the keep predicate and writes the indices of kept keys into a compact per-group list. Compaction is an $O(n)$

scan, so its parallelization—not its arithmetic—determines its cost: we grid it over (#KV groups $\times$ up to 512 blocks) and use warp-aggregated atomics (one ballot/population-count and a single `atomicAdd` per warp rather than per key) to claim output slots; output order is arbitrary, which is sound because attention is order-independent over the key set. *(ii) Split-K partial attention.* A second kernel computes online-softmax partial results (m, ℓ, acc) over the compact list, gridded over (KV group $\times$ split); each block derives its index range from the on-device compaction count, avoiding any host–device synchronization. Within a block, tiles of 64 kept keys' K/V are staged into shared memory *once per KV group* and reused by all query heads in the group—the kernel-level counterpart of grouped unfolding (§3.3) and analogous to NSA's group-centric design—with fp32 accumulation throughout. *(iii) Combine.* A final kernel merges the per-split partials into the output with the standard log-sum-exp correction, one block per head. Decoding therefore touches only the selected $\sim k(1+L)$ keys plus $O(n)$ bytes of metadata, never the full KV cache.

End-to-end decode overhead. After the compact decode operator is reduced to microsecond scale, the full decoding step is dominated by host-side launch overhead from the routing path: gist scoring, masking, top-k, and bitmap construction are small operations but are repeated across layers. We therefore cache all sequence-derived metadata once per decode step and share the one-token gist-state extension across layers. As an optional serving optimization, we also compile the two-level per-group selection into a graph-captured region. To make graph capture robust under variable context lengths, we bucket the number of summary tokens to powers of two and mask padded entries to $-\infty$. We may also round the selected budget up to a small grid, which selects a superset of the default top-k chunks; this path is used only as an optimized serving mode, while the default compact kernel preserves the reference selected set.

F More Details on Experimental Setting

Following the two-stage procedure described in §2.3, we first perform continued pretraining with interleaved gist tokens to learn context compression, then optionally apply selective finetuning to incorporate the unfolding mechanism into training. Following ActivationBeacon (Zhang et al., 2024), we use data sampled from RedPajama (Weber et al., 2024) for continued pretraining of Qwen2-7B-Instruct, with sequence lengths ranging from 7K to 20K. Following KVLink (Yang et al., 2025), we use data sampled from FineWeb (Penedo et al., 2024) for pretraining of Llama-3.2-1B, with sequence lengths ranging from 2K to 4K. For finetuning data, we use the dataset provided by ActivationBeacon (Zhang et al., 2024) for Qwen2-7B-Instruct, which contains samples from LongAlpaca (Chen et al., 2023), BookSum (Kryściński et al., 2022), and a synthesized QA dataset. For Llama-3.2-1B, we use the dataset provided by KVLink (Yang et al., 2025), which contains samples from the training sets of TriviaQA (Joshi et al., 2017) and 2WikiMQA (Ho et al., 2020), with answers generated by DeepSeek-V3.2 (Liu et al., 2025). All training is conducted on 8 NVIDIA H100 GPUs. Table 4 summarizes the hyperparameters for each model and training stage.

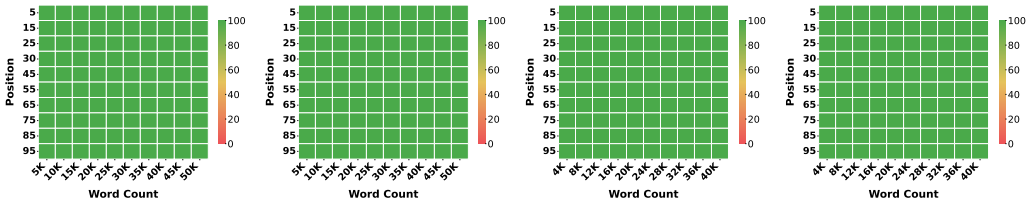
Table 4: Training hyperparameters for each model and stage.

Model	Stage	Batch Size	Learning Rate	Weight Decay	LR Schedule	#Epoch
Qwen2-7B-Inst.	Pretraining	8	1e-5	✓	Cosine w/ warmup	1
	Finetuning	8	5e-6	✓	Cosine w/ warmup	1
Llama3.2-1B	Pretraining	128	1e-5	✓	Cosine w/ warmup	1
	Finetuning	64	5e-6	✓	Cosine w/ warmup	2

G Experiments on Passkey Retrieval

To evaluate fine-grained retrieval under compression, we test on the passkey retrieval benchmark, where a short passkey is inserted at a random position within a long distractor context and the model must locate and reproduce it exactly. Figure 2 reports retrieval accuracy as a function of context length (horizontal axis) and relative insertion position (vertical axis) for both after continue pretrained SSA and H-SSA on Qwen2-7B-Instruct (up to 50K words, $2.5\times$ the maximum sequence length seen during continued pretraining) and Llama-3.2-1B (up to 40K words, $10\times$ the maximum sequence length seen during continued pretraining).

Both SSA and H-SSA achieve **perfect 100% accuracy** across all context lengths, all insertion positions, and both model families. The uniformly green heatmaps confirm that the selective unfolding mechanism reliably identifies and recovers the passkey regardless of where it is placed—beginning, middle, or end—and regardless of how long the surrounding context is. This holds even for the hierarchical variant, where the passkey must be located through multi-level coarse-to-fine routing. The result demonstrates that SSA's selective unfolding mechanism is robust and position-unbiased. It also demonstrates that SSA generalizes robustly to sequence lengths far exceeding those seen during continued pretraining.



(a) SSA on Qwen2 (b) H-SSA on Qwen2 (c) SSA on Llama3.2 (d) H-SSA on Llama3.2

Figure 2: Passkey retrieval accuracy of SSA and H-SSA on Qwen2-7B-Instruct (left two panels) and Llama3.2-1B (right two panels). The vertical axis denotes the context length, while the horizontal axis represents the relative insertion position (%) of the passkey.

H Parallel Computing with KV-cache Reuse

KV-cache reuse—where KV caches of long documents are stored and reused across repeated queries to avoid redundant prefilling and support parallel computing—is a natural fit for SSA. In this setting, SSA encodes documents independently with interleaved gist tokens and reuses their KV caches at query time.

Following the KVLink (Yang et al., 2025) protocol, we insert link tokens between documents. These Link tokens attend only to gist and meta-gist tokens from preceding documents, enabling efficient cross-document routing, while selective unfolding recovers full-resolution detail only for the most relevant segments. This avoids traversing full raw KV caches and improves efficiency without sacrificing quality. In Table 5, SSA achieves an average RAG performance of 48.07 at $8\times$ compression and 41.59 at $16\times$, substantially outperforming KVLink and UniGist; H-SSA further improves the $16\times$ compression result to 46.34. These results support gist tokens as effective routing signals for cross-document reasoning under KV-cache reuse.

Table 5: RAG benchmarks with KV-cache Reuse under different compression ratios ($8\times$, $16\times$) using Llama3.2-1B. Best results are in bold.

	Methods	HotpotQA	TriviaQA	2WikiMQA	MuSiQue	NQ	Avg.
x8	KVLink	41.55	60.00	63.91	9.68	30.86	41.20
	UniGist	42.74	59.98	61.51	9.02	28.48	40.35
	SSA	**54.23**	**64.81**	**70.21**	**13.98**	**37.10**	**48.07**
	H-SSA	–	–	–	–	–	–
x16	KVLink	37.11	57.97	61.43	8.03	27.84	38.48
	UniGist	37.20	57.65	58.14	7.53	25.42	37.19
	SSA	44.58	59.32	61.88	9.68	32.50	41.59
	H-SSA	**50.95**	**62.77**	**68.26**	**12.25**	**37.46**	**46.34**

I Ablation Analysis

Selective Rules Comparison. Table 6 ablates the design of the hybrid attention context (Eq. equation 3). We compare three variants: (1) **SR-only**: the attention context includes only the selected raw tokens, without their corresponding gist tokens; (2) **AG+SR**: all gist tokens are retained in the context

alongside the selected raw tokens; and (3) **SG+SR**: only the selected gist tokens are retained together with their raw tokens (our default). SG+SR achieves the best average score (53.39), outperforming SR-only (52.76) and slightly surpassing AG+SR (53.27). The result confirms our design choice in §2.2: retaining selected gist tokens alongside their raw tokens provides complementary compressed information, while including all gist tokens (AG+SR) dilutes the attention with less relevant summaries. Dropping gist tokens entirely (SR-only) loses the compressed cross-chunk patterns that gist tokens capture.

Table 6: Ablation Study on different selective rules. We adopt SG+SR as the default configuration.

Variants	HotpotQA	TriviaQA	2WikiMQA	MuSiQue	NQ	Avg.
SR-only	58.74	68.33	71.70	**20.76**	44.28	52.76
AG+SR	<u>59.53</u>	**68.74**	<u>71.72</u>	<u>20.68</u>	<u>45.70</u>	<u>53.27</u>
SG+SR (ours)	**59.88**	<u>68.62</u>	**71.99**	20.27	**46.20**	**53.39**

What the Gist Score Selects. The claim that gist tokens act as learned routing signals predicts that replacing the gist score with any query-independent ranking should collapse performance, even at an identical token budget. We test this directly on the finetuned Qwen2-7B model ($L = 4$, $J = 4$), holding the entire sparse-attention machinery fixed—same top-k budget ($\approx 1\%$ of tokens), same parent constraint, same kernel, same always-attended local window—and swapping only the score used to rank chunks: **gist**, our $q^\top k_{g_m}$ score; **recent**, which keeps the most recent chunks (position-only); and **random**, which samples chunks uniformly (a lower bound). Table 7 reports results on the three dispersed multi-document QA tasks, where the evidence a query needs is scattered across the context. The gist pivot beats random by 11.1, 8.5 and 9.9 points and recent by 12.0, 9.5 and 11.8 points, showing that the gist score carries genuine query-conditioned relevance rather than merely inducing sparsity.

Table 7: Ablation on the selection pivot: only the chunk ranking score is changed, with the token budget ($\approx 1\%$) and all other components held fixed.

Selection pivot	HotpotQA	2WikiMQA	MuSiQue
Full attention (dense)	53.31	49.33	31.06
Recent	43.11	38.44	17.19
Random	44.03	39.45	19.13
Gist (ours)	**55.12**	**47.95**	**29.02**

Top-k vs. Top-p. Figure 3 compares our adaptive top-k selection (red dashed line) against top-p (nucleus) selection with varying thresholds $p \in \{0.80, 0.85, 0.90, 0.95\}$ (green/blue solid lines) across six LongBench tasks. Top-p selection dynamically adjusts the number of unfolded chunks by including chunks until their cumulative score mass reaches p. Across all six tasks, adaptive top-k consistently outperforms every top-p threshold, often by a substantial margin. For instance, on 2WikiMQA, top-k achieves over 51% while the best top-p variant ($p = 0.80$) reaches only 47.5%. On GovReport, top-k scores approximately 30 compared to 28–29 for all top-p settings. The performance of top-p is also notably unstable across tasks: on NarrativeQA it peaks at $p = 0.85$ but drops sharply at other thresholds, and on NQ it fluctuates between 19 and 21 with no consistent trend. This instability arises because the score distribution varies across queries and tasks, making a fixed probability threshold a poor proxy for the number of relevant chunks. In contrast, adaptive top-k directly controls the token budget, providing a more reliable and stable selection mechanism.

J Efficiency and Scalability

Setup. We benchmark the kernels of §E on a single NVIDIA H100 80GB (3.35 TB/s HBM3) with the finetuned SSA Qwen2-7B-Instruct model at 16× compression (28 query heads, 4 KV heads, d=128, bf16). We evaluate two variants: a single-level SSA (L=16) and the hierarchical H-SSA (L=4, J=4). The dense baselines are: FlashAttention (Dao et al., 2022) for prefill and Flash-Decoding (Dao et al., 2023) for decoding. We report time-to-first-token (TTFT) for prefill and time-per-output-token (TPOT) for decoding.

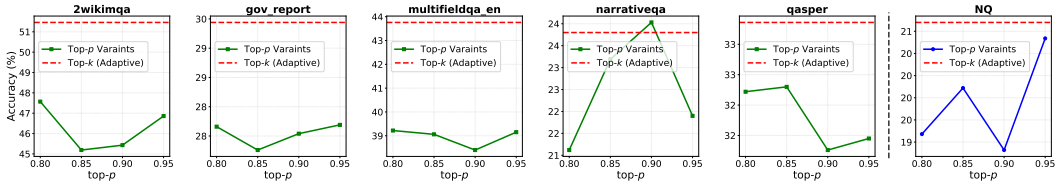


Figure 3: Top-k vs. top-p selection across five LongBench tasks (on Qwen2) and NQ (on Llama3.2). The red dashed line indicates our adaptive top-k selection; green/blue solid lines show top-p variants with $p \in \{0.80, 0.85, 0.90, 0.95\}$. Adaptive top-$k$ consistently outperforms all top-p thresholds and exhibits greater stability across tasks.

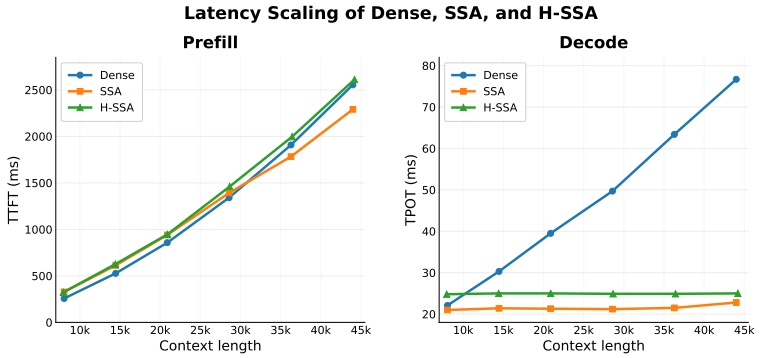


Figure 4: **End-to-end latency scaling on Qwen2-7B.** During prefill, SSA and H-SSA achieve TTFT comparable to the dense FlashAttention baseline across the measured context range (8K–44K). During decoding, the main scalability benefit emerges: dense Flash-Decoding's TPOT grows approximately linearly with context length, whereas SSA and H-SSA remain nearly flat.

End-to-end scaling. As shown in Figure 4, the clearest end-to-end gain comes from decoding. Dense Flash-Decoding rereads the entire KV cache at every step, so its per-token latency grows with context length, increasing from 21.9 ms at 8K to 76.4 ms at 44K. In contrast, our method reads only around 1% of the cache, making this cost negligible rather than dominant. As a result, its per-token latency remains nearly flat: the two-level H-SSA stays close to 25 ms, while the single-level SSA remains around 21–23 ms across the full range. Consequently, the speedup widens with context length, reaching 3.05× for H-SSA and 3.37× for SSA at 44K. For prefill, SSA keeps pace end-to-end with FlashAttention and pulls ahead at longer contexts. H-SSA is 1.29× slower at 8K but closes the gap to parity by 44K. The single-level SSA crosses over near 33K and becomes faster than the baseline thereafter, reaching 0.90× the dense latency at 44K.

Table 8: Attention-operator scaling for dense and H-SSA operators on an H100 GPU.

	Prefill operator (ms)			Decode operator (μs)		
k_{len}	FlashAttn	H-SSA	speedup	Flash-Dec	H-SSA	speedup
32,768	22.30	3.82	5.8×	44.2	35.6	1.24×
50,000	52.70	7.95	6.6×	57.1	40.8	1.40×
65,536	90.13	13.17	6.8×	64.7	45.5	1.42×
100,000	226.83	28.28	8.0×	88.1	56.3	1.56×
131,072	393.50	47.71	8.2×	107.7	66.3	1.62×
200,000	924.18	111.31	8.3×	151.9	96.3	1.58×

Attention-operator scaling. The operator-level results in Table 8 isolate the attention computation from model-level and host-level overheads; we use H-SSA as the representative configuration. For prefill, its sparse operator is consistently faster than dense FlashAttention, with the gap widening as sequence length increases: the speedup rises from 5.8× at 32K tokens to 8.3× at 200K tokens. For decoding, the sparse operator is 1.24–1.62× faster than Flash-Decoding over 32K–200K tokens.