SALSA: TRAINING-CONSISTENT SPARSE CACHING WITH SINK PRESERVATION FOR LINEARIZED LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The full attention mechanism in standard Transformer architectures incurs quadratic computational complexity with respect to sequence length and linearly growing KV cache overhead, posing severe efficiency bottlenecks in long-sequence scenarios. Although linear attention reduces computational and cache overhead, its compression of history into a fixed-size hidden state prevents precise recall of distant information. Existing conversion methods, such as LoLA, attempt to introduce a sparse cache into hybrid architectures to retain hard-to-compress tokens, but add it only during inference. This causes the attention mechanism learned during conversion to differ from the one actually executed at deployment, leading to a training-inference computation graph mismatch. The core insight of this paper is that the sparse cache should not be merely an inference-time patch; rather, it should be incorporated into the conversion computation graph from the attention distillation stage onward, so that training and inference execute the same attention mechanism. Based on this, we propose SALSA (Sink-Aligned Linearization with Sparse Adaptive Caching), a training-inference-consistent sparse caching linearization framework. SALSA partitions history into three components—a sliding window, a sparse global cache, and a linear attention hidden state—and enables the sparse cache to participate in both attention distillation and fine-tuning, thereby eliminating training-inference distribution shift. On this basis, we further propose two supporting refinements: (1) retaining initial sink tokens in the sliding window to stabilize the attention distribution; and (2) offline layer-wise sparse cache budget allocation based on inter-layer heterogeneity. Experiments on Llama-3.1-8B show that SALSA substantially outperforms prior state-of-the-art methods such as LoLCATs and LoLA on S-NIAH, improving average accuracy by more than 20 absolute points over LoLA while closely matching the full-attention baseline on S-NIAH-1 at 0.5K–2K context lengths. On the efficiency side, SALSA keeps decode cost nearly constant as context grows and, at 32K, achieves $4.0\times$ higher maximum decode throughput than FA2 under memory constraints, with nearly flat decode memory.

1 INTRODUCTION

In recent years, large language models built upon the Transformer (Vaswani et al., 2017) architecture have advanced rapidly in natural language processing. As an increasing number of application scenarios demand stronger long-context processing capabilities, long-sequence modeling ability has become a key metric for assessing the practical utility of large language models. However, the full attention mechanism in standard Transformer architectures imposes a severe efficiency bottleneck when processing long sequences.

To alleviate the efficiency bottleneck of full attention, existing research has mainly followed three directions: sparse attention, linear attention, and hybrid attention that combines the strengths of both. Against this background, the model conversion paradigm has attracted widespread attention. Its key advantage is that it avoids costly re-pretraining by directly and efficiently migrating existing full-attention models into hybrid architectures of sparse and linear attention, thereby inheriting pretrained knowledge and substantially reducing deployment costs. However, existing approaches inherit the inherent limitations of linear attention and struggle to achieve precise recall of distant information. Although methods such as LoLA (McDermott et al., 2025) introduce a sparse cache to

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.

retain hard-to-compress tokens, they add it only during inference. This causes a mismatch between the attention mechanism learned during conversion and the one actually executed at deployment, leading to training-inference computation graph inconsistency and consequently distribution shift.

The core insight of this paper is that the sparse cache should not be merely an inference-time patch; rather, it should be incorporated into the conversion computation graph starting from attention distillation, so that training and inference follow the same attention computation paths (up to cache capacity). Based on this, we propose SALSA, a training-inference-consistent sparse caching linearization framework. SALSA partitions sequence history into three complementary memory structures: a sliding window for local precise modeling, a sparse global cache for storing tokens that are difficult to memorize precisely, and a linear attention hidden state for compressing global information. The key to SALSA is that the sparse cache participates throughout attention distillation and parameter-efficient fine-tuning, thereby fundamentally eliminating training-inference distribution shift.

On this basis, we further propose two supporting refinements. First, we retain a small number of initial sink tokens in the sliding window to restore attention coverage at early positions, stabilize the attention distribution, and accelerate distillation convergence. Second, based on the heterogeneity of attention behavior across Transformer layers, we adopt pyramid-shaped offline layer-wise sparse cache budget allocation, assigning larger cache capacities to shallow layers and smaller capacities to deep layers, thereby improving model performance without introducing runtime overhead. Together with the training-inference-consistent sparse caching mechanism, these two refinements form an end-to-end framework for converting full-attention models without re-pretraining.

Experiments on Llama-3.1-8B (Grattafiori et al., 2024) show that SALSA substantially outperforms prior state-of-the-art methods such as LoLCATs (Zhang et al., 2025) and LoLA on S-NIAH. We further validate the generalization of our method on Qwen3-8B (Yang et al., 2025) and Mistral-7B-v0.1 (Jiang et al., 2023). The main contributions of this paper can be summarized as follows: (1) We propose SALSA, a training-inference-consistent sparse caching linearization framework that elevates the sparse cache from an inference-time patch to a core component of the conversion stage. (2) We design two supporting refinements: sink token preservation and offline layer-wise budget allocation. (3) We validate the effectiveness and generalization of our method on multiple long-context and commonsense reasoning benchmarks.

2 Related work

Sparse Attention. Sparse attention reduces computational and memory costs by restricting each token to attend to a subset of positions. Existing methods differ in their sparsity patterns: local window approaches limit attention to neighboring tokens; global anchor approaches introduce special tokens for global information aggregation (Xiao et al., 2024; Beltagy et al., 2020; Zaheer et al., 2020); and dynamic selection approaches adaptively identify key positions based on content importance (Li et al., 2024; Yuan et al., 2025; Lu et al., 2026; Zhang et al., 2023).

Linear Attention. Linear attention methods replace the exponential dot-product kernel with low-rank approximations, enabling recursive computation via hidden and normalization states. This reduces memory to a constant-size state and avoids explicit QK matrix multiplication. However, compressing history into a fixed-dimensional hidden state causes *memory collisions*, where non-orthogonal keys interfere and impede precise recall, particularly in copy-intensive tasks. Consequently, numerous improvements to standard linear attention have been proposed, including data-dependent gating (Yang et al., 2023), delta-rule updates (Yang et al., 2024), and state-space extensions (Gu & Dao, 2023).

Hybrid Attention Architectures. Sparse attention loses global information due to its restricted positional coverage. Moreover, when the subset of attended positions is dynamically determined from all historical tokens, the full KV cache remains necessary, and thus memory footprint is not reduced. Linear attention compresses the entire history into a bounded representation, making it difficult to precisely recall local patterns. Consequently, hybrid attention that combines the strengths of sparse and linear mechanisms has become a popular and effective direction. Recently, many hybrid attention architectures have been proposed. NHA (Du et al., 2026) uses a linear RNN to maintain a fixed number of long-term memory slots while retaining short-term tokens within a sliding win-

dow, concatenates the two, and then applies a unified softmax. MiniCPM-SALA (Team et al., 2026) adopts inter-layer hybridization, alternating sparse attention layers and linear attention layers at a 1:3 ratio. Unlike these works, SALSA maintains all three memory structures within each layer: a sliding window, a linear attention hidden state, and a sparse global cache. This avoids both the full linear compression of long-term memory as in NHA, and the inter-layer separation of sparse/linear mechanisms as in MiniCPM-SALA, whose sparse layers still require storing the full KV cache.

Model Conversion. Model conversion replaces the softmax attention in pretrained models with linear or sub-quadratic alternatives, inheriting pretrained knowledge while reducing cost. LoLCATs (Zhang et al., 2025) proposes a two-stage pipeline: attention distillation, which trains linearized layers to match the original attention outputs, followed by LoRA (Hu et al., 2021) fine-tuning to recover task performance. HALO (Chen et al., 2026) transfers attention weights to RNN layers and, through layer selection, retains critical softmax attention layers while converting the remaining layers into RNN layers, followed by end-to-end distillation and long-context fine-tuning. LAWCAT (Liu et al., 2025) adds a causal depthwise Conv1D to Q and K respectively, and employs Gated Linear Attention with retained normalization, following the same training pipeline as LoLCATs. LoLA (McDermott et al., 2025) builds on LoLCATs by further introducing a sparse cache at inference time to retain tokens that are difficult to memorize precisely. SALSA incorporates the sparse cache into attention distillation and parameter-efficient fine-tuning, ensuring that training and inference execute the same attention mechanism (up to cache capacity), thereby eliminating distribution shift.

Layer-wise Budget Allocation. Transformer layers exhibit substantial heterogeneity in attention behavior, motivating adaptive per-layer budget allocation. CAKE (Qin et al., 2025) distribute KV cache budgets according to spatial attention dispersion and temporal attention shift, while SimLayerKV (Zhang et al., 2024) prune caches in lazy layers that primarily attend to a small subset of tokens. These methods demonstrate that uniform allocation is suboptimal, yet dynamic runtime adjustment introduces non-trivial overhead.

3 METHOD

3.1 SALSA ARCHITECTURE

Our hybrid attention architecture combines sparse attention with linear attention. The sparse component comprises a sliding window and a sparse global cache, while the linear component maintains a compressed hidden state for global information. We now describe this architecture in detail.

Sparse cache. LoLCATs (Zhang et al., 2025) demonstrates the feasibility of converting pretrained models into linear attention while preserving reasonable performance. However, on long-context tasks, the sliding window fails to retain distant tokens, and the linear state lacks fine-grained local modeling capacity, leading to degraded performance. Motivated by this limitation and following LoLA (McDermott et al., 2025), we augment LoLCATs' sliding window with a *sparse cache* that stores tokens the model cannot precisely memorize.

Specifically, we employ Self-Recall Error (SRE) to measure the ability of past keys to retrieve their associated values:

$$\mathrm{SRE}(k, v \mid H_t, s_t) = \left\| \frac{\phi_k(k)^\top H_t}{\phi_k(k)^\top s_t} - v \right\|_2 = \|\hat{v} - v\|_2 \,, \tag{1}$$

where H_t and s_t denote the linear attention hidden state and normalization state, respectively. This quantifies the discrepancy between the predicted and true values for a given key, serving as the core criterion for cache updates.

The sparse cache is maintained as follows. Let ε_t denote the candidate pool at time t, formed by unioning the previous cache G_{t-1} with the token evicted from the sliding window $(k_{t-\eta}, v_{t-\eta})$. We retain the top-λ pairs with highest SRE:

$$\varepsilon_t = G_{t-1} \cup \{(k_{t-\eta}, v_{t-\eta})\}, \quad G_t = \underset{G \subset \varepsilon_t : |G| = \lambda}{\arg\max} \sum_{(k,v) \in G} \mathrm{SRE}(k, v \mid H_t, s_t). \tag{2}$$

The remaining pairs $S_t = \varepsilon_t \setminus G_t$ are compressed into the linear state:

$$H_t = H_{t-1} + \sum_{(k,v)\in S_t} \phi_k(k)v^\top, \quad s_t = s_{t-1} + \sum_{(k,v)\in S_t} \phi_k(k). \tag{3}$$

Sliding window with sink tokens. Drawing on StreamingLLM's (Xiao et al., 2024) insight, initial tokens in autoregressive language models attract substantial attention scores even when semantically unrelated to the current generation task. This phenomenon stems from the softmax operation itself: since softmax requires attention weights to sum to 1, the model must allocate "surplus" attention to visible tokens. Initial tokens, visible to all subsequent positions, are naturally trained as *attention sinks* that absorb excess attention mass. Discarding these tokens in the sliding window causes severe attention distribution mismatch at early positions, making attention transfer difficult to converge and increasing conversion cost. Moreover, LoLCATs shows that attention transfer quality (measured by MSE) strongly correlates with final model quality (measured by perplexity), suggesting that preserving sink tokens can accelerate transfer convergence and reduce MSE.

Accordingly, we prepend s fixed initial tokens to the sliding window to stabilize attention distributions and facilitate convergence. Let the original window size be η; we add s sink tokens at the front (default: $s = 4$), yielding a total window of length $\eta + s$:

$$\text{window}(t) = \{x_0, x_1, \ldots, x_{s-1}\} \cup \{x_{t-\eta+1}, \ldots, x_t\}. \tag{4}$$

Linear attention. Finally, the linear attention component employs standard linear attention with distinct feature maps for queries and keys. The detailed formulas are provided in the Appendix A.1.

Cache mechanism. The cache mechanism operates as follows. During initialization, the first s input tokens are placed into the sliding window as sink tokens. New tokens enter the sliding window first. Once full, newly arriving tokens evict the oldest non-sink token (at position $s + 1$). Sink tokens are permanently retained in the sliding window and never enter the sparse cache or linear state. Tokens evicted from the sliding window enter the sparse cache; when the sparse cache is full, the lowest-SRE tokens are compressed into the linear state. Figure 1 illustrates this architecture.

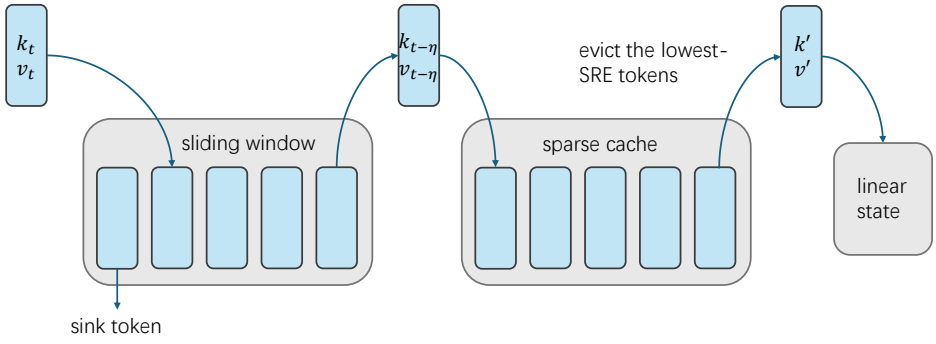


Figure 1: Schematic diagram of the cache architecture.

Attention output. The final attention output combines all three memory structures; the detailed formulas are provided in the Appendix A.2.

Complexity analysis. Full attention incurs $O(L^2d)$ complexity for attention scores, whereas our hybrid architecture decomposes computation into three constant-cost components—sliding window, sparse cache, and linear state—reducing per-step attention to $O(d)$ and yielding $O(Ld^2)$ overall for the full sequence, compared to $O(L^2d + Ld^2)$ for full attention. For memory, full attention stores all L key-value pairs per layer, whereas our method maintains only fixed-size structures, yielding constant memory per layer regardless of sequence length.

3.2 Model Conversion

SALSA adopts the two-stage conversion pipeline of LoLCATs—attention distillation followed by parameter-efficient fine-tuning—with one critical distinction: SALSA integrates the sparse cache

into the training objective, ensuring that training and inference share the same attention computation paths (up to cache capacity; see below).

Training-inference consistency. SALSA incorporates the sparse cache into the full training pipeline. After extending the architecture with sparse caches and sink tokens, we apply the same two-stage training as LoLCATs. This guarantees that the sparse cache participates in both attention distillation and fine-tuning, eliminating distribution shift. Importantly, our notion of consistency concerns the *mechanism*, not the *capacity*. The distribution shift caused by LoLA stems from a *mechanism* mismatch: the model is trained to attend only through a sliding window plus a linear state, but at inference a third computation path (the sparse cache) appears that was never optimized, so the attention operator itself differs between training and deployment. The cache size, in contrast, is a *capacity* hyperparameter of a mechanism that is present throughout training: all three computation paths—window, sparse cache, and linear state—are exercised, optimized, and normalized together during distillation and fine-tuning, regardless of the final cache size. Increasing the cache capacity at inference therefore introduces no unseen computation; it merely widens a path the model already knows how to use. This distinction also explains why inference-time changes of other capacity hyperparameters, such as the sliding-window size or a LoRA adapter's rank, are not regarded as train–inference mismatch.

Notably, training with an excessively large sparse cache degrades performance (Appendix B.3). We attribute this to a capacity–competition effect among the three memory structures. With a large cache, the SRE-based selection retains most hard-to-compress tokens in the sparse cache, so the linear state receives supervision only from the remaining "easy" tokens; the feature map ϕ is then trained to compress an unrepresentative subset of history and fails to learn effective global compression. Training with a small cache, by contrast, forces the linear attention component to absorb tokens that the cache cannot hold, yielding a better-trained compressed global state. This observation leads to our training principle: *capacity mismatch can be beneficial, whereas mechanism mismatch is purely harmful*. We therefore adopt small-cache training with large-cache inference: the model is trained under a tight cache budget and the capacity is enlarged at deployment, which widens an already-trained computation path. Appendix B.3 verifies this principle, showing that a model trained with a small cache and evaluated with a large cache clearly outperforms one trained and evaluated with the large cache.

Pre-fill chunking. Our hybrid architecture requires updating the sparse cache whenever a token is evicted from the sliding window, which is feasible during autoregressive decoding. However, during the pre-fill phase, per-token updates are too slow and prevent efficient training. We therefore employ a chunking strategy, partitioning the input into chunks of size C (by default equal to the sliding window size). Tokens within a chunk share the same sparse cache; after processing a chunk, the C tokens evicted from the sliding window are evaluated together against the current cache. We compute SRE scores for all C new tokens and existing cache entries, then evict the lowest-SRE tokens into the linear state. This amortizes cache maintenance overhead without altering training dynamics.

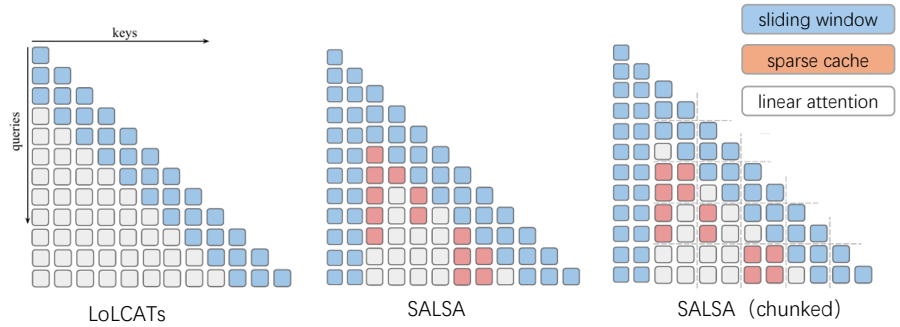


Figure 2: Schematic diagram of the storage location of each KV pair at each time step for each method.

3.3 LAYER-WISE BUDGET ALLOCATION

Existing methods typically allocate identical cache capacities across all Transformer layers, ignoring significant inter-layer heterogeneity in attention patterns. PyramidKV (Cai et al., 2024) observes that lower layers exhibit dispersed, broad-spectrum attention for global aggregation, while deeper layers concentrate overwhelmingly on a few key tokens. Uniform allocation thus leads to insufficient capacity in shallow layers and wasted resources in deep layers.

Following PyramidKV, we address this issue by adopting a pyramid-shaped budget allocation for the sparse cache: larger capacities for shallow layers and smaller capacities for deep layers, with the last layer set to half the average budget. The sliding window size remains uniform across layers; the detailed formulas are provided in Appendix A.3. This allocation is performed offline and introduces no runtime overhead.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Base model. We conduct experiments on Llama-3.1-8B as the primary model, and further verify generalization on Qwen3-8B and Mistral-7B-v0.1.

Baselines. We compare against LoLCATs (Zhang et al., 2025) and LoLA (McDermott et al., 2025). Specifically, we use the attention distillation procedure provided by LoLCATs to train the base model. During training, we replace the attention modules in Llama-3.1-8B with sliding window and linear attention modules. We use a sliding window size of 64 and a trainable feature map ϕ with output dimension $D = 2d_k$. First, we freeze all non-attention layers in the linearized transformer and distill ϕ for two epochs. Then, we apply LoRA fine-tuning to the entire model for two epochs. LoLA additionally uses a sparse cache of size 64 during inference.

Evaluation tasks. We evaluate on two broad categories of tasks: (1) RULER (Hsieh et al., 2024), which consists of challenging long-context tasks; and (2) commonsense reasoning, including PiQA (Bisk et al., 2020), ARC Easy, ARC Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and MMLU (Hendrycks et al., 2020).

4.2 RULER

To investigate how SALSA improves associative recall, we study the S-NIAH tasks from RULER. S-NIAH tests a model's ability to precisely retrieve a specific piece of information from an extremely long context. Table 1 reports the performance of our improved model against LoLCATs, LoLA, and Llama-3.1-8B on S-NIAH tasks. We use the LM Evaluation Harness for evaluation. Since SALSA uses a sliding window of size 68 (including 4 sink tokens), and a sparse cache of size 64, for a total cache size of 132, we expand LoLCATs' sliding window to 132 and train it with the same procedure (denoted LoLCATs-132) for fair comparison.

Table 1: S-NIAH results on Llama-3.1-8B.

	S-NIAH-1					S-NIAH-2			S-NIAH-3	
	0.5K	1K	2K	4K	8K	0.5K	1K	2K	0.5K	1K
Llama-3.1-8B	100	100	100	100	100	100	100	100	100	100
LoLCATs	8.0	3.2	1.2	0.8	0	**100**	7.8	2.2	69.4	1.4
LoLCATs-132	33.2	12.2	3.8	2	0	**100**	19.0	**7.6**	**99.8**	**16.2**
LoLA	93.8	85.4	47.4	1	0	**100**	19.6	2	99.4	1.6
SALSA	**100**	**99.8**	**98.4**	**90.2**	**0.8**	**100**	**69.2**	2.4	99.2	1.6

As shown, SALSA achieves near-base-model performance on some tasks, with substantial gains over LoLCATs and LoLA on most tasks. Performance degrades as context length increases, but this can be mitigated by increasing sliding window and sparse cache sizes.

Since the remaining RULER benchmarks contain harder context tasks, we test with a larger sparse cache. The settings are as follows: LoLCATs uses a sliding window size of 516 and is trained with the same training pipeline (denoted LoLCATs-516), while SALSA is configured with a sliding window size of 132 (including 4 sink tokens) and a sparse cache size of 384, but reuses the previously trained weights rather than being retrained. Both have identical total cache sizes for a fair comparison. LoLA uses the same settings as ours but without sink tokens.

Tables 2 shows results on RULER at context lengths 2K, covering harder tasks including multi-key retrieval (MK1, MK2, MK3), multi-query (MQ), multi-value (MV), variable tracking (VT), common word extraction (CWE), frequent word extraction (FWE), and Squad-QA (SQA). Hotpot-QA (HQA) requires at least 4K, so it is omitted where noted. At a 2K context length, SALSA improves average accuracy by 47.8 absolute points over LoLA , demonstrating the effectiveness of our method.

In addition to Llama-3.1-8B, we also evaluate our method on Qwen3-8B and Mistral-7B-v0.1, with the corresponding S-NIAH results shown in Tables 3 and 4, respectively. Although SALSA exhibits some performance fluctuations across different base models, its improvements over LoLCATs and LoLA remain substantial. Additional experimental results are provided in the Appendix B.1.

Table 2: RULER results at 2K context length.

Model	MK1	MK2	MK3	MQ	MV	VT	CWE	FWE	SQA	Avg
Llama-3.1-8B	100	100	100	100	99.05	100	98.92	90.4	66.6	95
LoLCATs-516	20.8	3.6	**1.2**	2.45	11.25	4.8	0	0	8.23	5.81
LoLA	23	1.6	0	14.5	28.75	65.6	1	25.33	42.2	22.44
SALSA	**91.6**	**58.4**	0	**92.75**	**94.1**	**69.36**	**86.9**	**75.67**	**63.33**	**70.23**

Table 3: S-NIAH results on Qwen3-8B.

	S-NIAH-1				S-NIAH-2			S-NIAH-3	
	0.5K	1K	2K	4K	0.5K	1K	2K	0.5K	1K
Qwen3-8B	100	100	100	100	100	100	100	100	100
LoLCATs	4.4	1.6	0	0	**100**	5.6	2.4	14.6	0
LoLA	15.4	8.6	0	0	**100**	5.8	14.8	**98.4**	0
SALSA	**100**	**99.8**	**98.8**	0	**100**	**98.6**	**43.6**	97.6	**0.2**

Table 4: S-NIAH results on Mistral-7B-v0.1.

	S-NIAH-1				S-NIAH-2			S-NIAH-3	
	0.5K	1K	2K	4K	0.5K	1K	2K	0.5K	1K
Mistral-7B-v0.1	100	100	100	100	100	100	100	100	100
LoLCATs	0.2	0	0	0	99.2	3.6	0	0	0
LoLA	72	51	0	0	**100**	9.2	0	**90.8**	0
SALSA	**98.8**	**96.2**	0	0	**100**	**83**	0	43.4	0

4.3 Commonsense Reasoning

Table 5 compares SALSA against LoLCATs and Llama-3.1-8B on various commonsense reasoning tasks. We use the evaluation code provided by LoLCATs, which employs the LM Evaluation Harness. For ARC Challenge and HellaSwag, we report acc_norm; for the remaining tasks, we report acc. Except for MMLU, which uses 5-shot prompting, all other tasks use zero-shot prompting. These settings are consistent with LoLCATs and LoLA.

On most tasks, SALSA achieves stable improvements over LoLCATs and LoLA, even surpassing Llama-3.1-8B. On the more difficult MMLU task, SALSA shows particularly notable improvement,

approaching the performance of Llama-3.1-8B. This indicates that SALSA brings more significant benefits on challenging commonsense reasoning tasks.

Table 5: Commonsense reasoning results on Llama-3.1-8B.

Model	PiQA	ARC-E	ARC-C	HellaSwag	WinoGrande	MMLU	Avg
Llama-3.1-8B	80.14	81.65	53.67	78.89	73.88	**65.97**	72.37
LoLCATs	81.18	82.74	55.55	79.18	**74.74**	52.46	70.98
LoLA	81.12	**83.5**	56.06	79.74	73.01	55	71.41
SALSA	**81.34**	82.87	**56.14**	**79.89**	73.48	63.62	**72.89**

4.4 Ablations

We conduct ablation studies to verify the effectiveness of our improvements. First, we verify that training with the sparse cache (training-inference consistency) indeed improves performance. Then, we verify that adding sink tokens and using adaptive budget allocation both improve performance under the same training pipeline. We primarily use S-NIAH tasks for ablations, since SALSA shows substantial gains over baselines on this dataset, making the contributions of different components more apparent.

Table 6 shows ablation results on training. SALSA (w/o training) uses the LoLCATs weights, while SALSA (w/ training) uses weights trained with sink tokens and offline budget allocation. Both variants employ sink tokens and offline budget allocation at inference. The results confirm that training–inference consistency improves model performance. Notably, the w/o-training variant constitutes a controlled case of *mechanism* mismatch under *capacity* consistency—its large performance gap confirms that the distribution shift harmful to performance arises from the mechanism, not the capacity.

Table 6: Ablations on training consistency for S-NIAH.

	S-NIAH-1				S-NIAH-2			S-NIAH-3	
	0.5K	1K	2K	4K	0.5K	1K	2K	0.5K	1K
SALSA (w/o training)	**100**	89.8	57	2.4	**100**	33	2.2	**100**	**1.6**
SALSA (w/ training)	**100**	**99.8**	**98.4**	**90.2**	**100**	**69.2**	**2.4**	99.2	**1.6**

Next, with training enabled, we ablate sink tokens and offline budget allocation. Table 7 shows performance on S-NIAH when adding sink tokens and offline budget allocation on top of the SALSA (w/ training) baseline. Although adding budget allocation alone yields only marginal improvements, the gains brought by combining the two improvements remain substantial, for example, on the 4K S-NIAH-1 task and the 1K S-NIAH-2 task. This validates that our improvements enhance model performance on such long-context tasks.

Table 7: Ablations on sink tokens and budget allocation for S-NIAH.

	S-NIAH-1				S-NIAH-2			S-NIAH-3	
	0.5K	1K	2K	4K	0.5K	1K	2K	0.5K	1K
SALSA (w/o sink & budget)	94.4	90.4	42.2	2.4	**100**	25.4	1.4	99.8	**1.6**
+ Sink	99.8	**99.8**	97.2	75.4	**100**	64.6	**2.6**	**100**	**1.6**
+ Budget	89.8	89.8	41.8	3.4	**100**	27.4	1.2	99.2	**1.6**
+ Sink & Budget	**100**	**99.8**	**98.4**	**90.2**	**100**	**69.2**	2.4	99.2	**1.6**

4.5 Efficiency and Wall-Clock Speed

We benchmark serving efficiency on one NVIDIA H800 GPU (80 GiB) with bfloat16 models, comparing the Llama-3.1-8B FA2 baseline and SALSA on prompts from 512 to 32K tokens, batch sizes from 1 to 32, generating 512 tokens. For decoding, we report the largest batch that fits at each context length; OOM points are marked in Figure 3.

SALSA is not prefill-optimized: its prefill throughput is roughly an order of magnitude lower than FA2. However, its decode cost stays nearly constant as context grows, keeping TPOT stable across 512–32K tokens. Consequently, once FA2 becomes memory-constrained, SALSA sustains larger serving batches; at 32K tokens, it achieves $4.0\times$ higher maximum decode throughput than FA2. The memory trend is consistent: FA2's peak decode memory grows steadily with context length and OOMs at 32K under batch 8, whereas SALSA stays nearly flat because its decode cache is fixed. Thus, the practical efficiency claim is not uniform speedup, but stable decode latency and a large advantage in memory-constrained long-context throughput. We leave optimized CUDA/ThunderKittens kernels to future work.

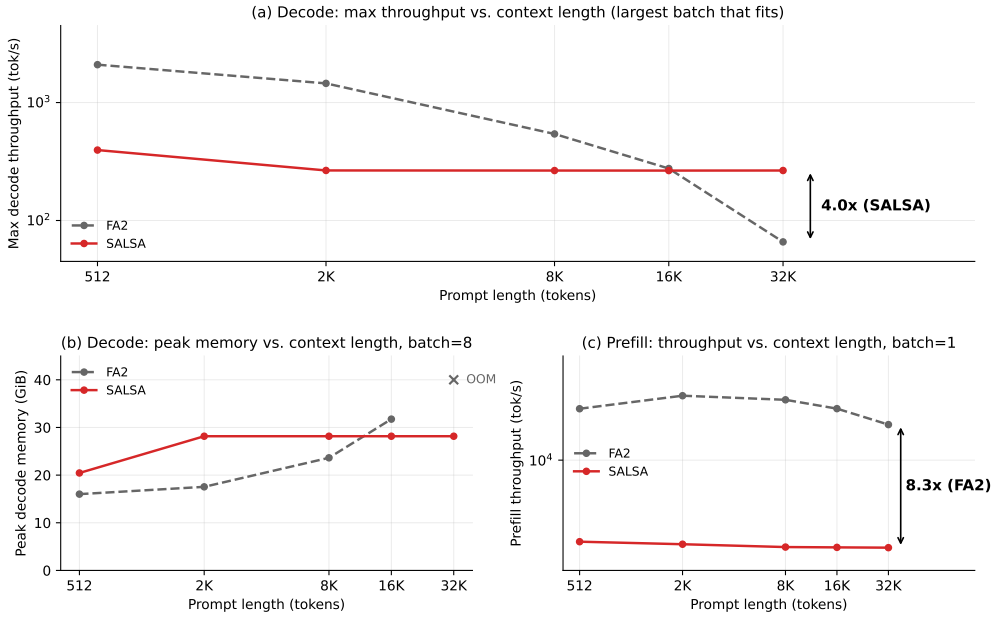


Figure 3: Serving efficiency on one H800 GPU.

5 Conclusion

This paper addresses the quadratic complexity and linear KV cache growth of full attention mechanisms in long-sequence modeling for large language models. We propose SALSA, an efficient hybrid attention approach combining sparse and linear attention, with three key improvements: retaining a small set of initial sink tokens, integrating sparse cache into the training objective, and introducing offline layer-wise budget allocation.

Experimental results demonstrate stable improvements on commonsense reasoning tasks, approaching full-attention baseline performance. On the RULER long-context benchmark, particularly the S-NIAH task, SALSA achieves significant gains, validating the enhanced key-information retrieval capability provided by sparse caching.

Our method is limited by the long-sequence bottleneck of fixed-capacity caches and the relative simplicity of offline budget allocation based on inter-layer heterogeneity. Future directions include improving the sparse cache selection function and exploring more powerful linear attention variants.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods, including creating and editing code, and for assisting with translation.

We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, or interpreting results.

The following tasks are not applicable to this work: generating synthetic data sets, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis.

Additionally, we used generative AI tools for creating or modifying scientific figures (specifically, the speed comparison figure was produced using AI-generated plotting code), editing for readability, and drafting sections of the manuscript.

We have reviewed all AI-assisted work. All AI-generated code was manually inspected, executed, and validated against expected outputs by the authors. All AI-polished text was cross-checked against the original drafts to ensure that no factual errors, hallucinations, or unintended changes in meaning were introduced. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our implementation extends the publicly available LoLCATs codebase. We follow the same two-stage conversion pipeline—attention distillation followed by LoRA fine-tuning—while incorporating the sparse cache into both stages as described in Section 3. The SALSA-specific modifications, including sink token preservation and offline layer-wise budget allocation, are detailed in Section 3 and Appendix A.3. All experimental settings, including base models, baselines, hyperparameters, evaluation tasks, and hardware, are provided in Section 4 and Appendix B . Our code modifications will be released upon publication.

REFERENCES

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

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

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

Yingfa Chen, Zhen Leng Thai, Zihan Zhou, Zhu Zhang, Xingyu Shen, Shuo Wang, Chaojun Xiao, Xu Han, and Zhiyuan Liu. Hybrid linear attention done right: Efficient distillation and effective architectures for extremely long contexts. *arXiv preprint arXiv:2601.22156*, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Jusen Du, Jiaxi Hu, Zhang Tao, Weigao Sun, and Yu Cheng. Native hybrid attention for efficient sequence modeling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3826–3842, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Zeyu Liu, Souvik Kundu, Lianghao Jiang, Anni Li, Srikanth Ronanki, Sravan Babu Bodapati, Gourav Datta, and Peter Anthony Beerel. Lawcat: Efficient distillation from quadratic to linear attention with convolution across tokens for long context modeling. In *EMNLP (Findings)*, pp. 20865–20881, 2025.

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. *Advances in Neural Information Processing Systems*, 38:17790–17815, 2026.

Luke McDermott, Robert W Heath Jr, and Rahul Parhi. Lola: Low-rank linear attention with sparse caching. *arXiv preprint arXiv:2505.23666*, 2025.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. In *International Conference on Learning Representations*, volume 2025, pp. 89945–89977, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

MiniCPM Team, Wenhao An, Yingfa Chen, Yewei Fang, Jiayi Li, Xin Li, Yaohui Li, Yishan Li, Yuxuan Li, Biyuan Lin, et al. Minicpm-sala: Hybridizing sparse and linear attention for efficient long-context modeling. *arXiv preprint arXiv:2602.11761*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

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

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

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.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Michael Zhang, Simran Arora, Rahul Chalamala, Benjamin Spector, Alan Wu, Krithik Ramesh, Aaryan Singhal, and Christopher Ré. Lolcats: On low-rank linearizing of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 46200–46253, 2025.

Xuan Zhang, Fengzhuo Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. Lighttransfer: Your long-context llm is secretly a hybrid model with effortless adaptation. *arXiv preprint arXiv:2410.13846*, 2024.

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

A Formulas

A.1 Linear Attention

The linear attention component in our architecture employs standard linear attention, with the detailed formulas provided as follows:

$$\exp\left(\frac{q_t^\top k_j}{\sqrt{d_k}}\right) \approx \phi_q(q_t)^\top \phi_k(k_j), \quad \phi_k, \phi_q : \mathbb{R}^{d_k} \to \mathbb{R}^D, \tag{5}$$

$$y_t^\top = \frac{\phi_q(q_t)^\top H_t}{\phi_q(q_t)^\top s_t}, \tag{6}$$

$$H_t = H_{t-1} + \phi_k(k_t) v_t^\top, \quad s_t = s_{t-1} + \phi_k(k_t). \tag{7}$$

A.2 Attention Output

The final attention output combines all three memory structures:

$$y_t = \frac{\phi_q(q_t)^\top H_t + \beta \left(\sum_{i \in G_t} e^{q_t^\top k_i/\sqrt{d}} v_i + \sum_{\substack{0 \le j < s \\ t-\eta < j \le t}} e^{q_t^\top k_j/\sqrt{d}} v_j \right)}{\phi_q(q_t)^\top s_t + \beta \left(\sum_{i \in G_t} e^{q_t^\top k_i/\sqrt{d}} + \sum_{\substack{0 \le j < s \\ t-\eta < j \le t}} e^{q_t^\top k_j/\sqrt{d}} \right)}, \tag{8}$$

where G_t denotes the sparse cache content at time t, $\eta + s$ is the sliding window size, and β is a learnable parameter that balances the contribution of softmax-based local attention (sliding window + sparse cache) against linear global attention.

A.3 Layer-wise Budget Allocation

Following PyramidKV, we address this issue by adopting a pyramid-shaped budget allocation for the sparse cache: larger capacities for shallow layers and smaller capacities for deep layers. The sliding window size remains uniform across layers. Formally, let $k^{(l)}$ denote the sparse cache size at layer l with total budget $K^{\text{total}} = \sum_{l=0}^{m-1} k^{(l)}$ fixed. Using an arithmetic progression with common difference $\Delta = \frac{k^{(0)} - k^{(m-1)}}{m-1}$, and setting the last layer to half the average budget $k^{(m-1)} = \frac{K^{\text{total}}}{2m}$, we obtain:

$$k^{(0)} = \frac{3K^{\text{total}}}{2m}, \quad k^{(l)} = k^{(0)} - \frac{k^{(0)} - k^{(m-1)}}{m-1} \cdot l. \tag{9}$$

This offline allocation requires no runtime overhead yet respects inter-layer heterogeneity. Figure 4 provides a schematic illustration of the layer-wise sparse cache budget allocation.

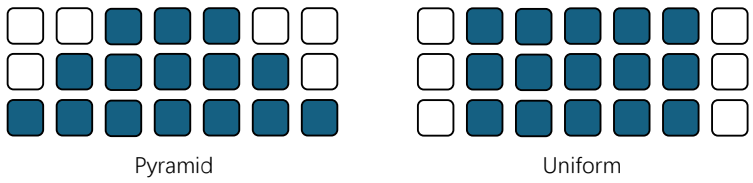


Figure 4: Layer-wise sparse cache budget allocation.

B Additional Experiments

B.1 Generalization to Other Base Models

We further evaluate SALSA on commonsense reasoning tasks using Qwen3-8B and Mistral-7B-v0.1 as base models, with the results shown in Tables 8 and 9, respectively. Similar to the results on Llama-3.1-8B, our method achieves the most pronounced gains on the most challenging task, MMLU, confirming the generalizability of our approach across different model families.

Table 8: Commonsense reasoning results on Qwen3-8B.

Model	PiQA	ARC-E	ARC-C	HellaSwag	WinoGrande	MMLU	Avg
Qwen3-8B	**76.71**	**83.33**	**56.23**	**74.85**	**67.96**	**76.76**	**72.64**
LoLCATs	73.56	71.38	46.42	64.6	65.27	59.1	63.39
LoLA	73.61	71.55	46.25	66.31	65.27	61.28	64.05
SALSA	<u>74.21</u>	<u>77.23</u>	<u>49.74</u>	<u>67.07</u>	<u>65.59</u>	<u>67.48</u>	<u>66.89</u>

Table 9: Commonsense reasoning results on Mistral-7B-v0.1.

Model	PiQA	ARC-E	ARC-C	HellaSwag	WinoGrande	MMLU	Avg
Mistral-7B-v0.1	80.58	80.85	54.01	**81.07**	<u>73.8</u>	**63.56**	**72.31**
LoLCATs	**81.50**	<u>82.74</u>	<u>56.83</u>	79.77	73.72	50.31	70.81
LoLA	<u>81.34</u>	**82.87**	**56.91**	80.58	**73.95**	48.48	70.69
SALSA	81.12	82.53	55.72	<u>81.04</u>	72.93	<u>55.66</u>	<u>71.86</u>

B.2 Effect of Sink Tokens on Distillation Convergence

To further investigate the effect of sink tokens, we additionally examine their impact on the distillation loss during the attention transfer stage. We expect that incorporating sink tokens will lead

to faster convergence. Figures 5 and 6 illustrate the effect of introducing sink tokens on the convergence of the distillation loss during this stage. Specifically, Figure 5 records the loss every 100 training steps, but the interval over the first 100 steps is too coarse to capture fine-grained dynamics; therefore, Figure 6 displays the loss at every step within the initial 100 steps. It can be observed that the configuration with sink tokens not only converges significantly faster than the one without, but also achieves a lower final MSE loss, validating the effectiveness of sink tokens in stabilizing attention distributions and accelerating transfer convergence.

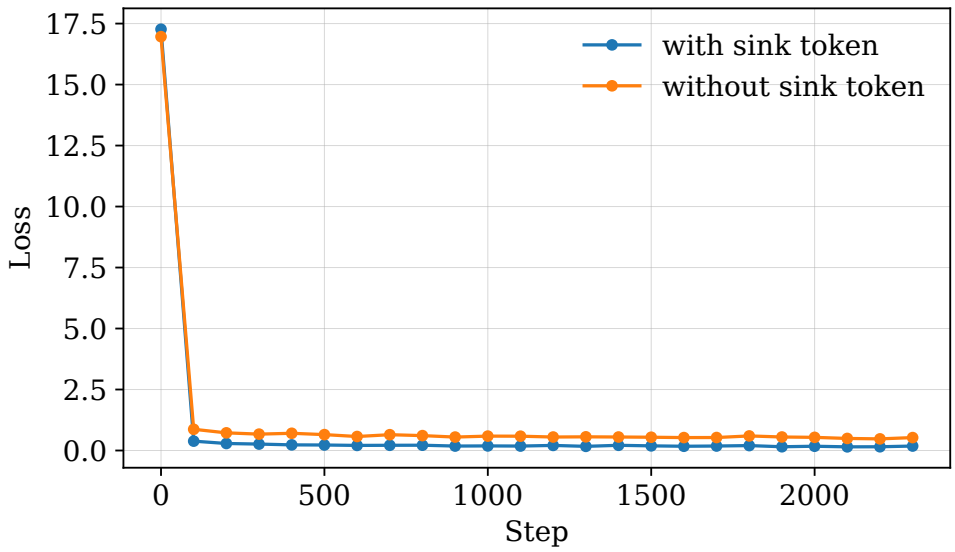


Figure 5: Effect of sink tokens on the convergence of distillation loss.

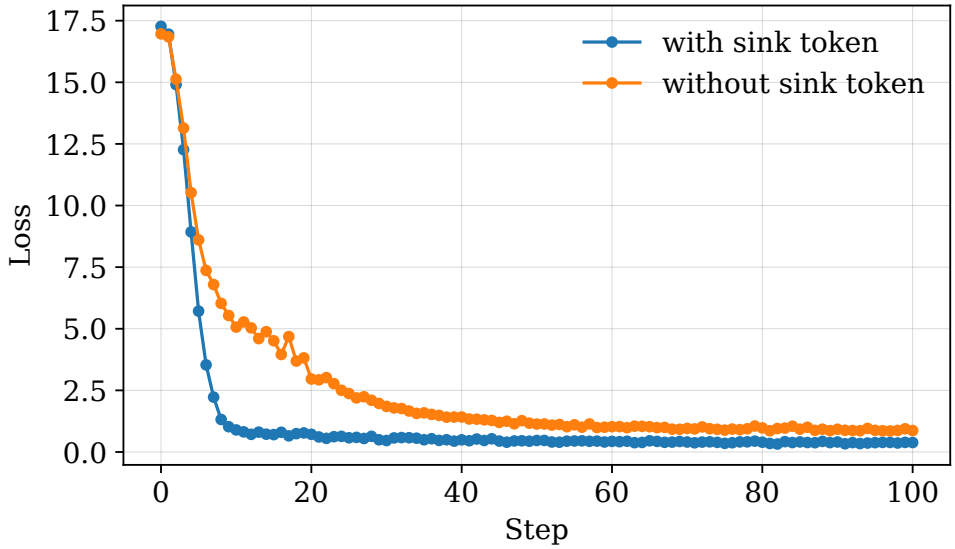


Figure 6: Effect of sink tokens on distillation loss convergence during the first 100 training steps.

B.3 Training Cache Size Ablation

Furthermore, in our main experiments with a larger inference cache, we reuse small-cache-trained weights and only increase the cache capacity at inference time. We conduct an ablation study on whether a larger cache should be used during training, with results shown in Table 10. We evaluate on RULER tasks other than S-NIAH, using a context length of 2K. The experimental setting at inference time uses a sliding window size of 132 (including 4 sink tokens) and a sparse cache size of 384. SALSA-L (large-cache training) uses the same experimental setting during training as at inference, whereas SALSA-S (small-cache training) uses a sliding window size of 68 (including 4 sink tokens) and a sparse cache size of 64 during training.

Table 10: Ablation on training cache size at 2K context length.

Model	MK1	MK2	MK3	MQ	MV	VT	CWE	FWE	SQA	Avg
SALSA-L	**92.6**	30.4	0	35.35	83.95	**97.92**	47.48	63.87	59.25	56.76
SALSA-S	91.6	**58.4**	0	**92.75**	**94.1**	69.36	**86.9**	**75.67**	**63.33**	**70.23**

B.4 S-NIAH with Larger Inference Cache

We increased the sparse cache size and evaluated on S-NIAH tasks with longer contexts; the results are reported in Table 11. At inference, the cache configuration uses a sliding window of size 260 (including 4 sink tokens) and a sparse cache of size 768. The weights remain those trained with a sliding window of size 68 (including 4 sink tokens) and a sparse cache of size 64, consistent with the experimental setup in the main text.

Table 11: S-NIAH results on Llama-3.1-8B with larger inference cache.

	S-NIAH-1				S-NIAH-2			S-NIAH-3		
	2K	4K	8K	16K	2K	4K	8K	1K	2K	4K
SALSA	100	100	40.4	0	100	98	1.8	99.6	94.4	7.6