SPMEM: SPARSE ATTENTION VIA MEMORY TOKENS

Anonymous authors
Paper under double-blind review

ABSTRACT

Most sparse attention methods store the entire key–value (KV) cache and are sparse only in how they *read* it. A selector must keep every entry in case a later query wants it, so the cache still grows with the context. These methods also pay for sparsity with machinery outside attention itself: a KV block compressor, a top-k selector, and a training recipe that departs from the base model's.

We present SpMem, a sparse attention that needs neither a compressor nor a selector and keeps the KV cache compressed throughout inference. On the global-attention layers, SpMem appends m memory tokens after every C real input tokens. Memory tokens are processed through a transformer forward pass, generating KV features that carry the core information of the prior input chunks. To this end, we train a model using next-token prediction with modified attention masking. During inference, real and memory tokens share one fixed retention rule, Tapered Window Attention (TWA), which keeps the most recent W tokens and halves the number retained with each further window of age. At 128K context a SpMem layer holds 16K entries against dense attention's 128K, and that number does not grow with context.

On Gemma 4 12B and 31B, against a dense model trained on the same data, SpMem stays close on knowledge and long-context reasoning and the cost falls mostly on retrieval. Keeping one of the eight global layers at full attention recovers most of that retrieval with a $4.3\times$ smaller KV cache at 12B, and puts that model at or above dense on five of seven benchmarks, MRCR-v2 among them. At 1M tokens, $8\times$ its training length, it finds every needle in single-needle NIAH once its memory cache is widened at inference, where the dense model finds 85%.

1 INTRODUCTION

Serving a transformer language model at long context is limited by memory, not arithmetic. Every token leaves a key–value (KV) pair at every attention layer, and the cache must stay resident for the rest of decoding. At a million tokens a model with full attention in every layer holds more cache than weights, which caps both the context length and the number of concurrent requests a server can hold. Recent architectures build long-context efficiency in directly, through multi-latent, linear and hybrid attention and local/global interleaving (DeepSeek-AI et al., 2025; Kimi Team, 2025; NVIDIA et al., 2025; The Microsoft AI Team, 2026). Some carry a *recurrent state* of bounded size, updated as the sequence streams by (Bulatov et al., 2022; Munkhdalai et al., 2024; Behrouz et al., 2024; Hwang et al., 2024); others keep a *compressed* history, pruning or merging KV entries while keeping random access to the past (Zhang et al., 2023; Xiao et al., 2024; DeepSeek-AI, 2024).

The most common answer has been sparse attention, and most of it is sparse only in how it *reads*. NSA (Yuan et al., 2025), MoBA (Lu et al., 2025), SPLA (Wang et al., 2026) and the parameter-free InfLLM-V2 (Zhao et al., 2026) all decide *which* entries a query attends to, so every entry must stay resident in case it is selected later. The per-step read drops to $O(\text{top-}k)$ while the cache stays $O(L)$: latency improves, but a memory-bound server can hold no more context than before. DeepSeek-V4 (DeepSeek-AI et al., 2026) moved sparsity to what is *stored* and reports a KV cache 10% of its dense predecessor's at a million tokens, though its successor has since pulled that compression back to 1–2$\times$ (Section 3). Both lines add machinery beside attention itself: a learned compressor, a top-k selector, or both.

We think most of this machinery is unnecessary, because a standard transformer can learn the sparsity it needs on its own. Recurrent memory transformers (Bulatov et al., 2022), gist tokens (Mu et al.,

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.

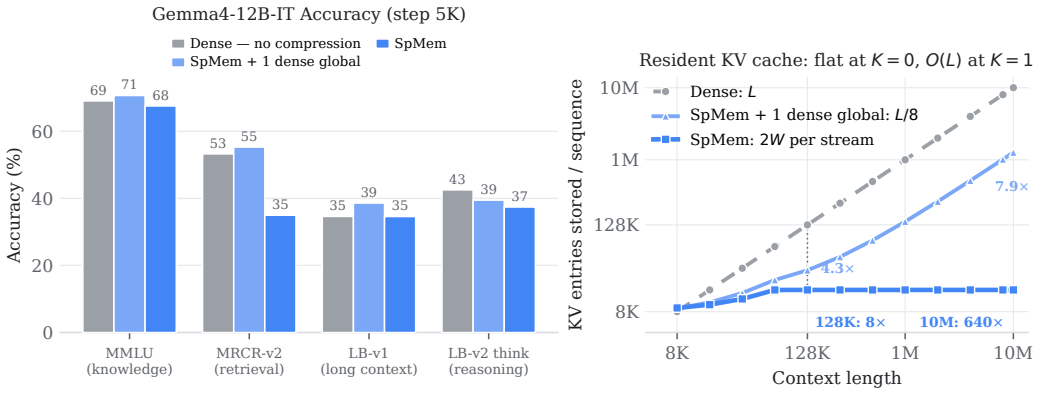


Figure 1: SpMem on Gemma 4 (Gemma Team, 2026) 12B-IT with SpMem configuration ($W = 4$K, $C = 8$, $m = 1$). **Left:** accuracy at step 5K against a dense model uptrained on the same data. With every global layer bounded, retrieval degrades; returning one global layer to full attention puts MMLU, LongBench-v1 and MRCR-v2 above dense. **Right:** resident KV entries per global layer against sequence length: SpMem stays at 16K, and with one dense global is still $4.3\times$ below dense at 128K.

2023) and their descendants insert a few dedicated positions that absorb the preceding content, and train them with the language-modeling loss alone. Viewed through the attention mask, this design is already a form of sparse attention: a query sees a bounded set of recent tokens plus a bounded set of summaries, never the full history. It needs no router, because the mask is fixed at training time, and no compressor, because the summaries are computed by the same weights as every other token.

The memory tokens' own cache, however, still grows with context. SpMem bounds both caches: real tokens and memory tokens are held under one retention rule, Tapered Window Attention (TWA), which keeps recent context in full and thins older context progressively, so neither cache grows with sequence length. The rest of the model is *unchanged*. The only new parameters are the memory-token embeddings, training is ordinary next-token prediction, and the same attention pattern is used in training and inference. Older context that the tapering drops remains reachable through the summaries, out to the memory cache's own reach (Section 2.3).

Our contributions are as follows. **(i)** We treat memory tokens as a sparse attention pattern rather than a memory mechanism, which removes the router and the compressor: the sparsity is a fixed mask shared by training and inference, and the summaries come from the model's own weights. **(ii)** We apply one retention rule, TWA, at two granularities, real tokens per token and summaries per chunk, so each compressed global layer holds a constant 16K entries at any context length. **(iii)** We show that the global layers are the long-context bottleneck, and that TWA and memory tokens each recover much of what a sliding window loses; together they match dense on LongBench-v1 at $8\times$ less cache. Keeping one of the eight global layers at full attention closes most of the remaining retrieval gap at $4.3\times$, and at a fixed budget memory tokens pay off only together with that layer.

2 Method

2.1 Overview

SpMem operates on a decoder-only transformer whose layers alternate between *local* (sliding-window) and *global* (full) attention (Beltagy et al., 2020; Gemma Team, 2024), as in several recent models (Gemma Team, 2025; 2026; OpenAI, 2025; Li et al., 2025). Local layers handle dependencies inside their window, and the global layers are the only place information crosses it and the only place whose cache grows with context: one layer in six under our base model's 5:1 interleaving, 8 of 48 at 12B.

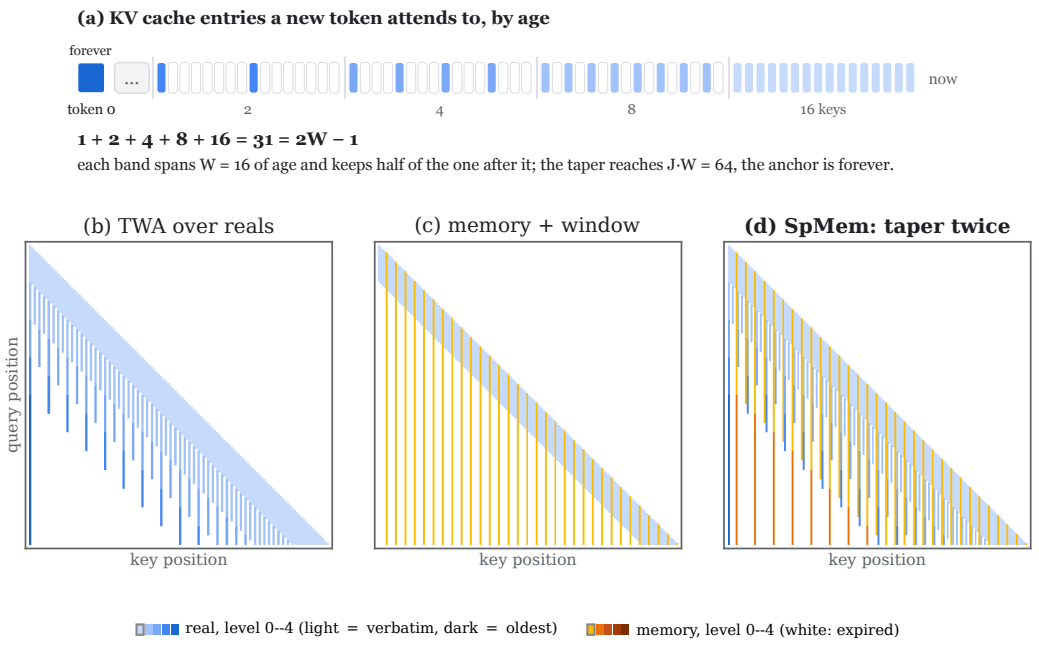


Figure 2: Tapered Window Attention is a mipmap over age, and SpMem applies it to both real and memory tokens. **(a)** The KV entries a new token can attend to, by age $q - k$: the most recent W kept, then levels that each span W of age with half the keys of the one before, $2W - 1$ entries in all. **(b)** The TWA mask over real tokens; each row is (a). **(c)** Memory tokens (orange) over a sliding window: every summary is kept, so the memory cache grows with the sequence. **(d)** SpMem: the same rule on the memory tokens, which thin and expire like the real tokens but reach r times further per slot. Drawn at $W = 16$, $C = 4$; Section 2.3 gives the shipped geometry.

The token stream is split into chunks of C tokens (we use $C = 8$), and m memory tokens are appended after each chunk (we use $m = 1$). Memory positions are marked with reserved ids of the base vocabulary and embedded from a dedicated $m \times d$ weight that the LM head never sees. They are inserted at the embedding lookup and removed before the output projection, so they exist only *inside* the transformer stack and no logits are ever computed at a memory position. Local layers keep their attention and simply see the memory tokens inside their window, which therefore holds about $C/(C + m)$ as many real tokens; nothing in SpMem depends on the local layers' type, so it applies unchanged when they hold a gated delta rule or a state-space block instead (Appendix A.4).

On the global layers, real tokens are held under **Tapered Window Attention** (TWA) rather than a sliding window. TWA has one parameter, the window W (we use $W = 4$K), and one rule. Define the *age* of key k at query q as $q - k$; it is a property of the pair, so it is defined the same way in prefill and in decoding. The rule: *the most recent W keys are kept verbatim, and resolution halves once per W of age behind them.* Level $j \geq 1$ covers ages $[jW, (j{+}1)W)$ and keeps only the keys whose position is divisible by 2^j. Figure 2 shows the rule, what it retains, and the resulting mask.

A mipmap over age. Level 0 is a ring buffer of W keys. Level j is a ring of $W/2^j$, fed by every other key evicted from level $j - 1$, so every level spans the same W of age with half as many keys as the one before it. The pyramid sums to

$$W + \tfrac{W}{2} + \tfrac{W}{4} + \cdots + 1 \;=\; 2W - 1, \qquad J = \log_2 W \text{ levels}, \tag{1}$$

the same geometric series that bounds the storage of a one-dimensional mipmap (Williams, 1983). Retained density falls as $2^{-\text{age}/W}$: dense near the query, and thinning fast behind it. The stride doubling per level is the trade a stack of dilated causal convolutions makes (van den Oord et al., 2016): the cost per output stays fixed while the receptive field grows. TWA makes that trade across ages within one layer rather than across depth, and stops after J levels (Section 2.3).

TWA contains the sliding window it replaces. Level 0 is W consecutive keys that are never thinned, so $\mathrm{SWA}(W) \subset TWA(W)$: whatever a sliding window of width W would see, TWA also sees, with a thinning sample of the older past behind it.

The same rule governs the memory tokens. Every chunk boundary emits m summaries, so L real tokens produce Lm/C of them, and a plain pool of summaries would grow without bound. SpMem gives them a TWA cache of their own, evaluating the identical rule in *memory-slot units*: recent summaries kept in full, older ones thinning on the same schedule. A memory slot stands for $r = C/m$ real tokens, so the memory cache reaches r times further than the real cache with the same number of slots. Unlike W, the memory window can be widened at inference without retraining (Section 5.4). Figure 2(c, d) shows the difference: under a sliding window the summaries accumulate indefinitely, and under SpMem they expire.

2.2 Attention masking

The same predicate governs real-token positions rpos and memory-token positions mpos. For a cache with window W and $J = \log_2 W$ levels, key k is resident for query q iff

$$q \geq k \;\wedge\; \Big(k = k_0 \;\;\vee\;\; \ell(k,q) \leq \min\big(\nu_2(k+c),\; J-1\big) \Big), \qquad \ell(k,q) = \Big\lfloor \frac{q-k}{W} \Big\rfloor, \tag{2}$$

where $\ell(k,q)$ is the mipmap level the key has aged into, ν_2 is the 2-adic valuation, $c \in \{0,1\}$ is the layer's offset, $J = \log_2 W$ indexes the last level and k_0 is its one permanent slot. A key that has aged into level ℓ is kept only if its position is divisible by 2^ℓ; level 0 imposes no condition, since $\nu_2 \geq 0$ for every key.

Two properties let a cache realize Eq. 2 exactly. It is *monotone*: ℓ only grows as q advances and the levels are nested, so once a key is dropped the predicate never asks for it again. And it is *closed-form in* (q,k), with no dependence on the cache's history, so the training mask and the serving cache are two readings of one equation rather than two implementations to keep in agreement.

Figure 2(d) draws the resulting mask. A real token attends to whatever the real cache still holds and to whatever the memory cache holds. A memory token reads the real cache as seen from the last real token of its own chunk, so it integrates exactly what an ordinary token there would; it also sees the retained memory tokens of earlier chunks, under the same taper, and its m siblings bidirectionally.

2.3 Recurrent state and memory footprint

On a global-attention layer the resident KV set is two tapered caches, and neither depends on L:

$$\underbrace{2W-1}_{\text{tapered reals}} \;+\; \underbrace{2W_{\text{mem}}-1}_{\text{tapered memory}} \;=\; O(W), \qquad r = \frac{C}{m}, \tag{3}$$

against the $O(L)$ of a dense global layer. The buffer is allocated once, from W, and never grows. At the shipped geometry ($W = 4$K, $C = 8$, $m = 1$) a global layer holds 8K real and 8K memory slots, 16K in total, against a dense 131,072: an $8\times$ reduction.

Effective receptive field. Reach also follows from the configuration in closed form. The levels run out after $J = \log_2 W$, so a key leaves its cache once its age passes $J \cdot W$, and the two caches see back

$$\underbrace{R \;=\; W \log_2 W}_{\text{reals}} \qquad \text{and} \qquad \underbrace{R_{\text{mem}} \;=\; r\, W_{\text{mem}} \log_2 W_{\text{mem}}}_{\text{memory, in reals}}, \tag{4}$$

the second because each memory slot stands for r real tokens. At the shipped geometry $R = 49$K, about 38% of a 128K context: real tokens older than that are gone, and only the memory cache covers that range. The memory cache is capped at the 128K training length, and Section 5.4 widens it at inference. A schedule that spends the same budget on reach instead of on resolution near the query is compared in Appendix A.7. Eq. 3 and Eq. 4 are the entire cost model.

The fixed budget does not shrink for short inputs, so SpMem pays off only where a dense cache has already become the binding cost, and it does not grow for long ones, so a session's memory stays fixed however long it runs.

2.4 DECODING COST

Memory tokens lengthen the stream to $L + \lfloor L/C \rfloor m$ positions but add no decoding steps: at a chunk boundary the m memory tokens enter the same forward pass as the chunk's last real token, which emits the next token's logits and writes the m summaries into the cache in one step of width $1 + m$.[1] Because decoding is memory-bandwidth-bound, that wider step is nearly free: its cost is dominated by streaming the weights and resident KV, which happens once regardless of query count. For the same reason a constant resident set decodes faster than an $O(L)$ one, with the gap widening in length and batch (Section 5.3).

3 RELATED WORK

Most prior work makes attention sparse in what it *reads* rather than in what it *stores*, and the two save different things: sparse reads cut compute and bandwidth per step but leave every entry resident, so the cache still grows as $O(L)$ while sparse storage bounds the cache itself. We place SpMem on that axis below, and Appendix A.1 treats each family in full.

Sparse in what it reads. Native Sparse Attention (Yuan et al., 2025) pairs coarse block compression with fine-grained token selection and is trained end-to-end; MoBA (Lu et al., 2025) routes each query to a mixture of key blocks; SPLA (Wang et al., 2026) keeps exact attention on selected blocks and folds the remainder into a recurrent linear state; and InfLLM-V2 (Zhao et al., 2026) shows the selector can be parameter-free. All of them keep every entry resident, so the cache stays $O(L)$. The closest relative of our *attention pattern* is LongNet (Ding et al., 2023), whose stride grows across distance bands so that resolution coarsens with age, as in TWA; it too is sparse only in what it reads: dilation thins each query's view, and every entry is still kept.

Sparse in what it stores. DeepSeek-V4 (DeepSeek-AI et al., 2026) moves sparsity to what is stored, pretraining with the design: interleaved layers compress every 4 and every 128 tokens into one KV entry with a learned compressor and a top-k selector, for 10% of DeepSeek-V3.2's KV cache at one million tokens. Its successor, DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), drops the $128\times$ path and lowers the remaining ratio to 1–2$\times$, recovering its footprint instead through cross-layer cache reuse and FP4 precision. That the sequence ratio fell while total compression rose suggests that compressing along the sequence is the hard one. Other store-side designs such as Beacon (Zhang et al., 2024) summarize spans into a pool that grows sublinearly but without bound. Training-free KV eviction (Zhang et al., 2023; Xiao et al., 2024; Li et al., 2024; Kim et al., 2025) bounds the cache with no machinery at all, but applies a fixed policy to the frozen model, so it can discard only what the original model already tolerates losing.

What we take from recurrence. Recurrent memory transformers (Bulatov et al., 2022) and gist tokens (Mu et al., 2023) established that a few dedicated positions can carry the preceding content when trained with nothing but a language-modeling loss. Replacing softmax retrieval with a fixed-size state has a measured cost, which Arora et al. (2024) chart as a recall–memory frontier. SpMem therefore keeps softmax attention and bounds the set it runs over: memory tokens are ordinary keys and values, addressed by the same attention as every other position.

4 EXPERIMENTAL SETUP

Base model and training. We apply SpMem to the instruction-tuned Gemma 4 checkpoints at 12B and 31B, decoder-only models with interleaved local and global attention. Continued pretraining runs in a JAX training stack on TPU, on sequences packed to 128K tokens, with real-token window $W = 4$K at 12B and 8K at 31B, a memory window of $W_{\text{mem}} = 4$K summaries at both, chunk size $C = 8$ and $m = 1$ memory tokens per chunk ($r = C/m = 8$).

The data come from internal web-scale sources under a Llama-style domain mixture (Touvron et al., 2023): long-form documents together with synthetic question–answer pairs, including multi-round

[1] The first token of a new chunk is therefore predicted before its own chunk's summaries exist. Nothing is lost: those summaries are computed from real tokens the predicting position already holds verbatim.

Table 1: Accuracy (%) at step 5K. **Dense** is Gemma 4 uptrained with the same data, optimizer and positional scheme as SpMem. **SpMem** bounds every global layer, so its KV cache stays the same size as L grows; $+$ 1 **global** returns one global layer to full attention, which is $O(L)$ in that layer alone. KV-cache reduction at 128K: $8\times$ and $4.3\times$ at 12B ($W = 4$K), $5.3\times$ and $3.7\times$ at 31B ($W = 8$K). Bold marks the best in each block.

	Short-context retention			Long-context reasoning		Retrieval	
Model	**MMLU**	**IFEval**	**AIME**†	**LB-v1**	**LB-v2**†	**MRCR-v2**	**NIAH**‡
				Gemma 4 12B			
Dense	69.0	59.9	33.3	34.6	**42.5**	53.2	**95.8**
SpMem	67.5	**67.7**	30.0	34.6	37.4	35.0	60.0
SpMem + 1 global	**70.7**	61.9	**43.3**	**38.6**	39.5	**55.3**	90.1
				Gemma 4 31B			
Dense	81.8	87.1	**63.3**	47.6	**45.6**	52.5	**99.0**
SpMem	**83.8**	**88.9**	53.3	**51.1**	42.2	48.1	73.6
SpMem + 1 global	83.2	88.2	53.3	50.5	43.9	**57.3**	91.6

†AIME and LongBench-v2 run in thinking mode (32K and 8K decode budgets); the other columns suppress thinking; IFEval is prompt-level accuracy. ‡Mean of RULER's eight retrieval subtasks over six length tiers, 4K–128K (Table 10).

long-context retrieval in the style of MRCR-v2, following established practice for long-context adaptation (Gao et al., 2025). The dense reference and every SpMem variant train on identical data, so the overlap with that evaluation's format affects absolute scores, not the comparisons between them. Training consumes 21B tokens at a global batch of 32 sequences. The objective is standard next-token prediction, and the optimizer reproduces the constraint Gemma 4 was itself trained under: a LaProp-style update, after which every 2D weight is projected back onto a sphere, with weight decay 0 and all 1D parameters frozen (Appendix A.9 explains why the constraint matters). 12B trains on a v5p-256 slice and 31B on v5p-512. We update all weights rather than train a LoRA adapter (Appendix A.16 gives the comparison). The dense reference is the same checkpoint uptrained with the same data and optimizer on a longer schedule, read at the same step (Appendix A.8), not the released model: continued pretraining alone moves the base model in both directions, costing 9.3 points on LongBench-v1 and gaining 13.5 points on MRCR-v2 (Appendix A.8), and every SpMem variant carries that shift.

Baselines. We compare against training-free compression from both sides of the reads/stores distinction (Section 3), applied post-hoc to the same dense checkpoint and, like SpMem, only to the global layers. FlexPrefill (Lai et al., 2025), MInference (Jiang et al., 2024) and X-Attention (Xu et al., 2025) are sparse in what they read, restricting each query to a subset of keys and leaving the cache unchanged; StreamingLLM (Xiao et al., 2024), SnapKV (Li et al., 2024) and KVzip (Kim et al., 2025) are sparse in what they store, discarding entries, and so are budget-comparable to SpMem. The contrast isolates what is gained by learning the compressed memory rather than imposing a fixed reduction on a frozen model.

Evaluation. We evaluate two things. *Retention* of the base model's short-context abilities is measured with MMLU (Hendrycks et al., 2021), IFEval (Zhou et al., 2023) and AIME 2025. *Long-context* behavior under the compressed cache is measured with LongBench-v1 (Bai et al., 2024) and v2 (Bai et al., 2025), MRCR-v2 (Vodrahalli et al., 2024) and RULER (Hsieh et al., 2024) over a range of context lengths. We report synthetic retrieval (RULER's needle-in-a-haystack, NIAH) separately from semantic long-context tasks (LongBench), because a bounded cache behaves differently on the two, and the gap between them shows what the cache gives up. For AIME, we use the 30 problems of the 2025 contest, scored by exact match on the boxed integer, so one problem is worth 3.33 points.

Table 2: Building SpMem one step at a time (Gemma 4 12B, step 5K). *Reals*/*Mem.*: how the global layers hold real and memory tokens (× none, *all* every summary). *Glob.*: global layers kept at full attention. *Budget*: resident KV entries per global layer at 128K. [§]: same-budget control for the row below. Bold: at or above dense.

Reals	Mem.	Glob.	Budget	LB-v1	LB-v2[†]	MRCR-v2	NIAH[‡]	RULER-128K
Dense	—	8	131K	34.6	42.5	53.2	95.8	78.8
Global attention replaced by a window								
SWA 8K	×	0	8.2K	32.5	35.0	17.2	47.8	15.2
TWA 16K	×	0	33K	**38.8**	37.1	44.2	77.1	34.0
SWA 4K	all	0	20K	29.5	30.6	39.4	65.9	45.2
SpMem: TWA + memory tokens								
TWA 8K[§]	×	0	16K	**38.6**	34.4	35.2	64.4	21.5
TWA 4K	TWA	0	16K	**34.6**	37.4	35.0	60.0	24.7
TWA 8K	all	0	33K	**40.2**	38.4	44.3	69.2	33.5
SpMem + 1 global layer at full attention								
TWA 8K[§]	×	1	30.7K	**39.5**	32.0	51.3	85.9	53.1
TWA 4K	TWA	1	30.7K	**38.6**	39.5	**55.3**	90.1	65.4

[†]Thinking mode. [‡]Mean of RULER's eight retrieval subtasks over all six length tiers, as in Table 1.

5 RESULTS

5.1 ACCURACY

Table 1 groups the benchmarks into short-context retention (MMLU, IFEval, AIME), long-context reasoning (LongBench-v1 and v2) and retrieval (MRCR-v2, NIAH). Bounding every global layer concentrates the cost in retrieval. At 12B, IFEval rises 7.8 and LongBench-v1 does not move (34.6 against 34.6), MMLU drops 1.5, AIME one problem and LongBench-v2 5.1, while retrieval loses 18.2 points of MRCR-v2 and 35.8 of NIAH: the cache gives up specific spans of the context first. Returning one of the eight layers to full attention changes the picture: SpMem + 1 global is at or above dense on five of the seven axes, MRCR-v2 included (55.3 against 53.2), and trails only on LongBench-v2 (-3.0) and NIAH (-5.7). That single layer holds more than half of the resident cache, so it halves the reduction, from $8\times$ to $4.3\times$. 31B shows the same pattern. SpMem alone is at or above dense on MMLU, IFEval and LongBench-v1 and loses on retrieval (MRCR-v2 48.1, NIAH 73.6). With one global layer kept it passes dense on MMLU, IFEval, LongBench-v1 and MRCR-v2 (57.3 against 52.5), is within 1.7 on LongBench-v2, and trails on AIME and NIAH.

5.2 WHAT EACH PIECE BUYS

Table 2 builds SpMem one step at a time. Every row keeps the 40 local layers' sliding-window attention and changes only how the 8 global layers hold their cache. Each row is the best measured configuration for its step; Appendix A.6 repeats the comparison with the KV budget held fixed within each pair.

Each piece recovers retrieval. Replacing the global layers with an 8K sliding window, with the 40 local layers' attention unchanged, leaves long-context reasoning largely intact (LongBench-v1 32.5 against 34.6) but collapses retrieval: MRCR-v2 falls from 53.2 to 17.2 and RULER-128K from 78.8 to 15.2, so eight of 48 layers carry most of the model's long-range retrieval. Both mechanisms recover part of it. At the sliding window's own width, TWA's thinning sample of the older past doubles MRCR-v2 (17.2 $\rightarrow$ 35.2) at twice the budget; memory tokens on a 4K window, keeping every summary, reach 39.4 at 2.5 times the budget. Together, at the shipped geometry, SpMem matches dense on LongBench-v1 (34.6 against 34.6) with $8\times$ fewer entries, but recovers only part of retrieval: NIAH 60.0 and MRCR-v2 35.0 against 95.8 and 53.2.

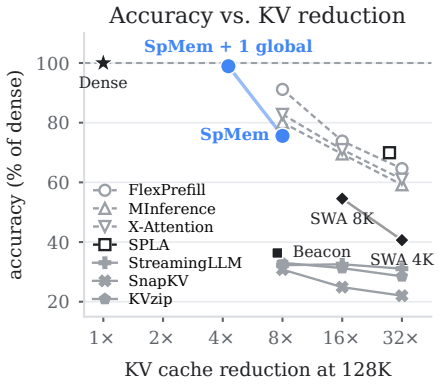


Figure 3: Accuracy against KV reduction at 128K. *Filled*: the cache shrinks; *hollow*: reads only, full cache. *Dark*: trained; *grey*: training-free. Accuracy: mean of MRCR-v2 and LongBench-v2, as % of dense.

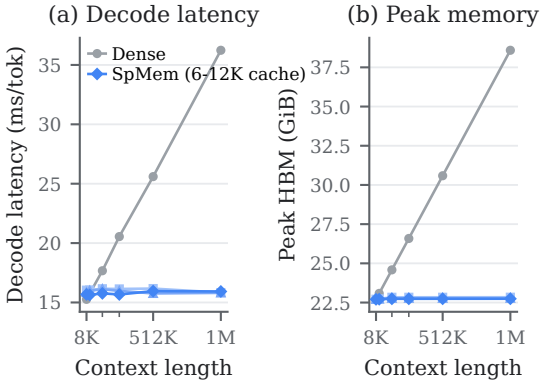


Figure 4: Inference cost against context length: **(a)** decode latency per token, **(b)** peak HBM, at three SpMem cache budgets (6.1–12.3K entries per global layer). Gemma 4 12B, bf16, one TPU v5p chip, batch 1.

One global layer closes most of the rest, and memory tokens are why. Returning one of the eight global layers to full attention takes MRCR-v2 to 55.3, above dense's 53.2, LongBench-v1 to 38.6 and NIAH to 90.1, leaving LongBench-v2 (-3.0) and RULER-128K (-13.4); the retained layer is unbounded, so this measures what one full-attention layer buys. The § rows show what makes it pay. At a fixed 16K budget with every global layer bounded, a wider 8K window and a 4K window with memory tokens trade wins, neither leading by more than 4.4. With one global layer retained, at about 30.7K, memory is ahead on MRCR-v2 ($+4.0$), NIAH ($+4.2$), LongBench-v2 ($+7.5$) and RULER-128K ($+12.3$), and within 0.9 on LongBench-v1. Memory tokens alone do not beat a wider window, but they make the retained layer far more effective. Which layer is kept matters as much as whether one is: in a sweep over positions, only the layer just above the center recovers retrieval, and a second retained layer, while helping MRCR-v2 and LongBench-v1, costs $1.47\times$ the cache, so we keep one (Appendix A.17).

5.3 Inference Efficiency

Accuracy against KV reduction. Figure 3 separates two kinds of saving. At the same $8\times$, the read-sparse methods are more accurate than bounded SpMem (80–91% of dense against 76%), but their $8\times$ is in *what each step reads*, not in what is stored. They retain every entry, so the memory footprint, and therefore the supported context and batch, does not shrink at all. On the other hand, SpMem's reduction is in storage. With one global layer at full attention it keeps 99% of dense in a cache $4.3\times$ smaller, more accurate than any read-sparse method while holding a quarter of the memory. Among methods that do shrink the cache, SpMem at $8\times$ keeps 76%, twice Beacon's 36% at a similar $7.5\times$ and well above a sliding window at either width (55% at $16\times$ for 8K, 41% at $32\times$ for 4K). Training-free eviction fares worst: StreamingLLM, SnapKV and KVzip keep 22–33% at 8–$32\times$, and on MRCR-v2 only 5–13% of dense. RULER-128K, where SpMem's retrieval gap is largest, is left out of this composite and reported in Table 2, with the full per-length breakdown in Appendix A.10.

We measure the per-step decode latency and resident memory on one TPU v5p chip in bf16 at batch 1, sweeping context from 8K to 1M (Figure 4). These runs predate TWA: they hold 6.1–12.3K entries per global layer under a sliding window, against 16K for the shipped configuration. Latency depends on how many entries are resident, not on the rule that picked them, and at batch 1 the three budgets sit within 0.5 ms of one another because the cache is under 1% of the bytes a step streams (Appendix A.5); we expect the shipped budget to fall on the same flat line. Both decode through the same standard attention; Appendix A.13 gives the protocol.

Across a $128\times$ range of context, SpMem decodes at a flat 15.6–16.2 ms/token and holds 22.7 GiB, while dense attention grows linearly with L in both. The two are within 4% of each other up to

Table 3: Single-needle NIAH at ∼1M tokens (100 prompts; last column: the 52 with the needle over 512K from the end). *Widened* swaps in a larger memory cache at inference only. *KV red.*: all eight global layers against dense.

		Memory cache				
Model	**Globals**	**entries**	**reach**	**KV red.**	**NIAH**	**>512K**
Gemma 4 12B, released	all dense	full cache		1×	11	9/52
Dense, uptrained	all dense	full cache		1×	85	45/52
SpMem	0 dense	8K	128K	64×	2	0/52
SpMem, widened	0 dense	32K	1M	26×	2	0/52
SpMem	1 dense	8K	128K	7.2×	34	12/52
SpMem, widened	1 dense	32K	1M	6.3×	**100**	**52/52**

32K, where the weights dominate every step, and separate beyond it: dense decodes 1.10× slower at 128K and 2.29× slower at 1M, where it needs 15.9 GiB more. SpMem adds no measurable prefill overhead, but its sparsity does not buy a prefill speedup at this length either.

5.4 Retrieval at one million tokens

Every needle, at a fraction of the cache. SpMem with one global layer at full attention and a widened memory cache finds all 100 needles, including all 52 placed more than 512K from the end, while each of its seven bounded layers holds 41K entries (8K real tokens and 32K memory tokens), 4% of the context; the eight global layers together stay 6.3× below dense. Both dense models keep their entire cache and still miss needles. Our uptrained dense reference finds 85, missing needles at every distance; the released Gemma 4 12B finds 11, and past its native 256K context length it misses even the needles nearest the end.

The memory cache can be widened after training, but only with a global layer. The checkpoint was trained with an 8K-entry memory cache reaching 128K. Swapping in a 32K-entry cache at inference, whose reach covers the whole 1M, takes NIAH from 34 to 100 with no change to the weights or to the real-token cache, so the memory window becomes a serving-time choice: a deployment can trade entries for reach without training a new model. The widening needs the retained global layer, as without it, NIAH stays at 2 either way. This is the same interaction Section 5.2 measures at 128K, where memory tokens pay off through the global layer. We observe this at the shipped $W = 4$K; widening does not help a $W = 16$K checkpoint (Appendix A.12).

Limitations. Finding one fact is not reasoning across a million tokens: on LongBench-v2 and BABILong at 1M every SpMem configuration stays near chance, and so does the dense model uptrained on the same data (31.6 on LongBench-v2, chance 25), so the floor is not caused by compression (Appendix A.12 gives the full 1M results).

6 Conclusion

We presented SpMem, a sparse attention built from memory tokens rather than from routers and compressors: m memory tokens inserted after every chunk on the global-attention layers, and one retention rule, TWA, applied to the real tokens and again to the summaries. It adds only m embedding rows, trains with ordinary next-token prediction, and is exactly consistent between training and inference. With every global layer bounded, each layer holds 16K entries at any context length, 8× below dense at 128K, and matches dense on LongBench-v1. Keeping one of the eight global layers at full attention recovers most of the retrieval that bounding costs, putting MRCR-v2 above dense with a KV cache still 4.3× smaller at 128K. At 1M tokens, 8× its training length, that model finds every needle in single-needle NIAH once its memory cache is widened at inference, where the dense model uptrained on the same data finds 85% and the released one 11%.

Reproducibility Statement

The method is specified completely in Section 2: the augmentation (Section 2.1), the mask (Section 2.2), the masked loss (Section 2.1), and the two-pool serving cache (Section 2.3), which realizes the training mask exactly (Section 2.2), at the geometry stated in Section 4. SpMem adds no new module and only m embedding rows to the base checkpoint, so the whole scheme is reproducible on any local/global decoder from that description plus the geometry.

The efficiency numbers are the easiest part of the paper to reproduce and the easiest to get wrong, so Appendix A.13 gives the protocol in full: the harness, the dtype and initialization, the chunked prefill, the requirement that the cache length be block-aligned for both dense and SpMem, the fact that both run on standard attention, and the fast-forward validation that makes the million-token points affordable. Two artifacts there produce large spurious gaps if left unchecked, and we report what each was worth.

The algorithm will be made publicly available: a `mask_fn`, a KV-cache implementation, and the token-embedding augmentation and wiring that attach them to a local/global decoder, with tests.

AI Use Statement

We used generative AI tools as coding and writing assistants. Use cases are: to write and refactor parts of the experiment, evaluation, and benchmarking code, including the figure-generating scripts; to run and analyze the efficiency measurements reported in Section 5.3; and to edit prose for clarity. The research questions, the method, the experimental design, and the interpretation of the results are the authors' own, and no text, figure, table, or number in this paper was accepted without an author verifying it against the underlying code or measurement. We have reviewed all AI-assisted work and take responsibility for the final content of this work.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. Colt5: Faster long-range transformers with conditional computation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2303.09752.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. In *International Conference on Machine Learning (ICML)*, 2024.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2507.10524.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2308.14508.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. arXiv:2412.15204.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.04517.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Ali Behrouz et al. Nested learning: The illusion of deep learning architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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

Aydar Bulatov, Yury Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Sakshi Choudhary, Aditya Chattopadhyay, Luca Zancato, Elvis Nunez, Matthew Trager, Wei Xia, and Stefano Soatto. Learning when to attend: Conditional memory access for long-context llms. *arXiv preprint arXiv:2603.17484*, 2026.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.16588.

Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002. doi: 10.1137/S0097539701398363.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y. Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z.

Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J. Q. Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, M. S. Di, M. Y. Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, S. H. Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W. Zhang, W. L. Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, Y. C. Yan, Y. Q. Wang, Y. W. Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, Z. F. Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025.

Yaosheng Fu, Guangxuan Xiao, Xin Dong, Song Han, and Oreste Villa. Sparda: Sparse decoupled attention for efficient long-context llm inference. *arXiv preprint arXiv:2606.04511*, 2026.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. arXiv:2410.02660.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021.

Namgyu Ho, Sangmin Bae, Taehyeon Kim, Hyunjik Jo, Yireun Kim, Tal Schuster, Adam Fisch, James Thorne, and Se-Young Yun. Block transformer: Global-to-local language modeling for fast inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.02657.

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? In *Conference on Language Modeling (COLM)*, 2024. arXiv:2404.06654.

Dongseong Hwang, Weiran Wang, Zhuoyuan Huo, Khe Chai Sim, and Pedro Moreno Mengibar. Transformerfam: Feedback attention is working memory. *arXiv preprint arXiv:2404.09173*, 2024.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and João Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, 2024. Spotlight Presentation.

Dongwon Jo, Beomseok Kang, Jiwon Song, and Jae-Joon Kim. Token sparse attention: Efficient long-context inference with interleaved token selection. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2602.03216.

Alexia Jolicoeur-Martineau, Rhea Sanjay Sukthanker, Pashmina Cameron, and Emy Gervais. Sliding-window beats linear attention. *arXiv preprint arXiv:2608.28444*, 2026.

Mahdi Karami, Ali Behrouz, Praneeth Kacham, and Vahab Mirrokni. Trellis: Learning to compress key-value memory in attention models. In *Conference on Language Modeling (COLM)*, 2025.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. Compressed context memory for online language model interaction. In *International Conference on Learning Representations (ICLR)*, 2024a.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.23416.

Minsoo Kim, Kyuhong Shim, Jungwook Choi, and Simyung Chang. InfiniPot: Infinite context processing on memory-constrained llms. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2024b. arXiv:2410.01518.

Minsoo Kim, Arnav Kundu, Han-Byul Kim, Richa Dixit, and Minsik Cho. EpiCache: Episodic kv cache management for long-term conversation on resource-constrained environments. In *International Conference on Machine Learning (ICML)*, 2026a. arXiv:2509.17396.

Minsoo Kim et al. BeaconKV: Key-value cache compression guided by beacon queries for efficient large reasoning model inference. In *International Conference on Machine Learning (ICML)*, 2026b.

Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2406.10149.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *International Conference on Learning Representations (ICLR)*, 2025. Oral Presentation.

Filipe Laitenberger, Dawid Kopiczko, Cees G. M. Snoek, and Yuki M. Asano. What layers when: Learning to skip compute in llms with residual gates. *arXiv preprint arXiv:2510.13876*, 2026.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1907.05242.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Ang Li, Sean McLeish, Haozhe Chen, Nimit Kalra, Zaiqian Chen, Artem Gazizov, Venkata Anoop Suhas Kumar Morisetty, Bhavya Kailkhura, Harshitha Menon, Zhuang Liu, Brian R. Bartoldson, Sanae Lotfi, Micah Goldblum, Pavel Izmailov, and Tom Goldstein. End-to-end context compression at scale. *arXiv preprint arXiv:2606.09659*, 2026.

Ethan Li et al. Apple Intelligence foundation language models: Tech report 2025. *arXiv preprint arXiv:2507.13575*, 2025.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2403.19887.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2502.13189.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.16300.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

NVIDIA, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Aleksandr Shaposhnikov, Alex Kondratenko, Alexander Bukharin, Alexandre Milesi, Ali Taghibakhshi, Alisa Liu, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Amy Shen, Anahita Bhiwandiwalla, Andrew Tao, Anjulie Agrusa, Ankur Verma, Ann Guan, Anubhav Mandarwal, Arham Mehta, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asit Mishra, Asma Kuriparambil Thekkumpate, Ayush Dattagupta, Banghua Zhu, Bardiya Sadeghi, Barnaby Simkin, Ben Lanir, Benedikt Schifferer, Besmira Nushi, Bilal Kartal, Bita Darvish Rouhani, Boris Ginsburg, Brandon Norick, Brandon Soubasis, Branislav Kisacanin, Brian Yu, Bryan Catanzaro, Carlo del Mundo, Chantal Hwang, Charles Wang, Cheng-Ping Hsieh, Chenghao Zhang, Chenhan Yu, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christopher Parisien, Collin Neale, Cyril Meurillon, Damon Mosk-Aoyama, Dan Su, Dane Corneil, Daniel Afrimi, Daniel Lo, Daniel Rohrer, Daniel Serebrenik, Daria Gitman, Daria Levy, Darko Stosic, David Mosallanezhad, Deepak Narayanan, Dhruv Nathawani, Dima Rekesh, Dina Yared, Divyanshu Kakwani, Dong Ahn, Duncan Riach, Dusan Stosic, Edgar Minasyan, Edward Lin, Eileen Long, Eileen Peters Long, Elad Segal, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Tramel, Erick Galinkin, Erik Pounds, Evan Briones, Evelina Bakhturina, Evgeny Tsykunov, Faisal Ladhak, Fay Wang, Fei Jia, Felipe Soares, Feng Chen, Ferenc Galko, Frank Sun, Frankie Siino, Gal Hubara Agam, Ganesh Ajjanagadde, Gantavya Bhatt, Gargi Prasad, George Armstrong, Gerald Shen, Gorkem Batmaz, Grigor Nalbandyan, Haifeng Qian, Harsh Sharma, Hayley Ross, Helen Ngo, Herbert Hum, Herman Sahota, Hexin Wang, Himanshu Soni, Hiren Upadhyay, Huizi Mao, Huy C. Nguyen, Huy Q. Nguyen, Iain Cunningham, Ido Galil, Ido Shahaf, Igor Gitman, Ilya Loshchilov, Itamar Schen, Itay Levy, Ivan Moshkov, Izik Golan, Izzy Putterman, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jatin Mitra, Jeffrey Glick, Jenny Chen, Jesse Oliver, Jian Zhang, Jiaqi Zeng, Jie Lou, Jimmy Zhang, Jinhang Choi, Jining Huang, Joey Conway, Joey Guman, John Kamalu, Johnny Greco, Jonathan Cohen, Joseph Jennings, Joyjit Daw, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kai Xu, Kan Zhu, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kevin Shih, Kezhi Kong, Khushi Bhardwaj, Kirthi Shankar, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Li Ding, Lizzie Wei, Lucas Liebenwein, Luis Vega, Maanu Grover, Maarten Van Segbroeck, Maer Rodrigues de Melo, Mahdi Nazemi, Makesh Narsimhan Sreedhar, Manoj Kilaru, Maor Ashkenazi, Marc Romeijn, Marcin Chochowski, Mark Cai, Markus Kliegl, Maryam Moosaei, Matt Kulka, Matvei Novikov, Mehrzad Samadi, Melissa Corpuz, Mengru Wang, Meredith Price, Michael Andersch, Michael Boone, Michael Evans, Miguel Martinez, Mikail Khona, Mike Chrzanowski, Minseok Lee, Mohammad Dabbah, Mohammad Shoeybi, Mostofa Patwary, Nabin Mulepati, Najeeb Nabwani, Natalie Hereth, Nave Assaf, Negar Habibi, Neta Zmora, Netanel Haber, Nicola Sessions, Nidhi Bhatia, Nikhil Jukar, Nikki Pope, Nikolai Ludwig, Nima Tajbakhsh, Nir Ailon, Nirmal Juluru, Nishant Sharma, Oleksii Hrinchuk, Oleksii Kuchaiev, Olivier Delalleau, Oluwatobi Olabiyi, Omer Ullman Argov, Omri Puny, Oren Tropp, Ouye Xie, Parth Chadha, Pasha Shamis, Paul Gibbons, Pavlo Molchanov, Pawel Morkisz, Peter Dykas, Peter Jin, Pinky Xu, Piotr Januszewski, Pranav Prashant Thombre, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Qing Miao, Qiyu Wan, Rabeeh Karimi Mahabadi, Rachit Garg, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Rich Harang, Rick Izzo, Rima Shahbazyan, Rishabh Garg, Ritika Borkar, Ritu Gala, Riyad Islam, Robert Hesse, Roger Waleffe, Rohit Watve, Roi Koren, Ruoxi Zhang, Russell Hewett, Russell J. Hewett, Ryan Prenger, Ryan Timbrook, Sadegh Mahdavi, Sahil Modi, Samuel Kriman, Sangkug Lim, Sanjay Kariyappa, Sanjeev Satheesh, Saori Kaji, Satish Pasumarthi, Saurav Muralidharan, Sean Narentharen, Sean Narenthiran, Seonmyeong Bak, Sergey Kashirsky, Seth Poulos, Shahar Mor, Shanmugam Ramasamy, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby

Thomas, Shiqing Fan, Shreya Gopal, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Simeng Sun, Smita Ithape, Somshubra Majumdar, Soumye Singhal, Stas Sergienko, Stefania Alborghetti, Stephen Ge, Sugam Dipak Devare, Sumeet Kumar Barua, Suseella Panguluri, Suyog Gupta, Sweta Priyadarshi, Syeda Nahida Akter, Tan Bui, Teodor-Dumitru Ene, Terry Kong, Thanh Do, Tijmen Blankevoort, Tim Moon, Tom Balough, Tomer Asida, Tomer Bar Natan, Tomer Ronen, Tugrul Konuk, Twinkle Vashishth, Udi Karpas, Ushnish De, Vahid Noorozi, Vahid Noroozi, Venkat Srinivasan, Venmugil Elango, Victor Cui, Vijay Korthikanti, Vinay Rao, Vitaly Kurin, Vitaly Lavrukhin, Vladimir Anisimov, Wanli Jiang, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenfei Zhou, Will Jennings, William Zhang, Wojciech Prazuch, Xiaowei Ren, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Yigong Qin, Ying Lin, Yonatan Geifman, Yonggan Fu, Yoshi Subara, Yoshi Suhara, Yubo Gao, Zach Moshe, Zhen Dong, Zhongbo Zhu, Zihan Liu, Zijia Chen, and Zijie Yan. Nvidia nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

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

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. arXiv:2508.10925.

Guanghui Qin, Corby Rosset, Ethan C. Chau, Nikhil Rao, and Benjamin Van Durme. Dodo: Dynamic contextual compression for decoder-only lms. *arXiv preprint arXiv:2310.02409*, 2023.

Qwen Team. Qwen3-next: Towards ultimate training and inference efficiency, 2025. `https://qwen.ai/blog`.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Zhenyi Shen, Junru Lu, Lin Gui, Jiazheng Li, Yulan He, Di Yin, and Xing Sun. Ssa: Sparse sparse attention by aligning full and sparse attention outputs in feature space. *arXiv preprint arXiv:2511.20102*, 2026.

Yangyang Shi, Yongqiang Wang, Chunyang Wu, Ching-Feng Yeh, Julian Chan, Frank Zhang, Duc Le, and Mike Seltzer. Emformer: Efficient memory transformer based acoustic model for low latency streaming speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021. arXiv:2010.10759.

The Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. `https://microsoft.ai/pdf/mai-thinking-1.pdf`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. Focused transformer: Contrastive training for context scaling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2307.03170.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, et al. Michelangelo: Long context evaluations beyond haystacks via latent structure queries, 2024. arXiv:2409.12640.

Bailin Wang, Chang Lan, Chong Wang, and Ruoming Pang. Rattention: Towards the minimal sliding window size in local-global attention models. *arXiv preprint arXiv:2506.15545*, 2025.

Bailin Wang, Dan Friedman, Tao Lei, and Chong Wang. Spla: Block sparse plus linear attention for long context modeling. *arXiv preprint arXiv:2601.22379*, 2026.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015. arXiv:1410.3916.

Lance Williams. Pyramidal parametrics. *ACM SIGGRAPH Computer Graphics*, 17(3):1–11, 1983. doi: 10.1145/964967.801126.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations (ICLR)*, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

Bowen Yang, Bharat Venkitesh, Dwarak Talupuru, Hangyu Lin, David Cairuz, Phil Blunsom, and Acyr Locatelli. Rope to nope and back again: A new hybrid attention strategy. *arXiv preprint arXiv:2501.18795*, 2025.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y X Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. arXiv:2502.11089.

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, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yingfa Li, Yuxuan Li, Yanghao Zhang, Wenhao Zhao, Zhiyuan Li, Yuxiang Huang, Ao Sun, Xu Han, and Zhiyuan Liu. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.24663.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Appendix

A.1 Related work in full

The long form of the introduction's survey. The dominant response, though, has been sparse attention, and most of it is sparse only in how it *reads*. Native Sparse Attention pairs coarse block compression with fine-grained token selection and is trained end to end from the start of pretraining (Yuan et al., 2025), MoBA routes each query to a mixture of key blocks (Lu et al., 2025), and SPLA keeps exact attention on selected blocks while folding the remainder into a compact recurrent linear state (Wang et al., 2026). In each case selection decides to *which* entries a query attends, so every entry must remain available in case it is selected later: the resident cache stays $O(L)$ even when the per-step read is $O(\text{top-}k)$. For a serving system whose binding constraint is memory rather than arithmetic, this lowers latency without extending the reachable context or raising concurrency. The machinery these methods add is not free either: InfLLM-V2 (Zhao et al., 2026) observes that NSA

introduces substantial extra parameters and departs from the usual pretrain-on-short, finetune-on-long workflow, with slow convergence as a consequence, and responds with a parameter-free variant that reuses the dense attention weights. That variant, however, is still a block selector, so its resident cache remains $O(L)$.

DeepSeek-V4 (DeepSeek-AI et al., 2026) moved sparsity to what is *stored*. Its hybrid attention combines Compressed Sparse Attention with Heavily Compressed Attention, and the architecture is pretrained with that design rather than grafted onto a dense model afterwards; at one-million-token context the reported KV cache is 10% of its dense-attention predecessor's. This is what has renewed interest in sparse attention as the route to long context, and it sets the bar we measure against. What such designs still require is a learned pooling compressor and a top-k selector alongside the attention itself, which is what makes them intricate to implement and, per the critique above, costly to converge.

Section 3 states the placement this paper depends on. This section is the full treatment, moved here for length.

A.1.1 Efficient Foundation Model Architectures for Long Context

A first line of work pursues long context by designing efficient foundation-model architectures and training them from scratch. Recent frontier open models bake long-context efficiency directly into the network: the DeepSeek series (DeepSeek-AI, 2024; DeepSeek-AI et al., 2025; 2026) builds on multi-latent attention to push toward million-token context, NVIDIA's Nemotron 3 (NVIDIA et al., 2025) and Microsoft's MAI models (The Microsoft AI Team, 2026) pair efficiency-oriented designs with large-scale training, and Kimi Linear (Kimi Team, 2025) and gated-linear-attention variants replace softmax attention with linear and hybrid mechanisms, as does local/global interleaved attention (Gemma Team, 2025). Based (Arora et al., 2024) is especially relevant to our framing: by combining linear attention with a short sliding window and varying the feature dimension and window size, it traverses the *recall–memory tradeoff*, showing that fixed-size recurrent states buy throughput at a measurable cost in recall. This is precisely the tradeoff a bounded-memory compressor faces, and SpMem sits on the memory side of it. A parallel push revisits recurrence at scale: xLSTM (Beck et al., 2024) equips the LSTM with exponential gating and a matrix memory, Hawk and Griffin (De et al., 2024) mix gated linear recurrences with local attention, Jamba (Lieber et al., 2025) interleaves Transformer and Mamba blocks in a mixture-of-experts hybrid, and Qwen3-Next (Qwen Team, 2025) ships an attention/linear-attention hybrid at production scale. These hybrids are the most direct alternative to SpMem, and they bound the resident cache for the same structural reason: a recurrent layer carries a fixed-size state regardless of sequence length. The difference is what the bound costs. A recurrent layer bounds memory by *replacing* softmax retrieval, and Arora et al. (2024) measure the consequence directly—fixed-state alternatives to attention struggle at recall, with the deficit governed by state size. SpMem bounds memory while leaving the operation in place: its memory tokens are ordinary keys and values, addressed by the same softmax attention as any other position, so retrieval remains content-based rather than order-based. Closest to the local/global design of our base model, RATTENTION (Wang et al., 2025) shifts the window-size Pareto of local–global attention by pairing local attention with a small residual linear-attention branch, and SPLA (Wang et al., 2026) keeps exact attention on a few selected blocks while compressing the remaining blocks into a compact recurrent linear state. SpMem does not compete with these architectures. It is a change of attention mask plus a handful of vocabulary tokens, so it composes with a local/global stack rather than replacing it, and nothing in the design requires a pretrained starting point; we demonstrate it by continued pretraining because that is what our compute budget allows, not because the mechanism depends on it.

Dilated attention is the closest relative of the *pattern*, and it is not the same claim. LongNet (Ding et al., 2023) attends on a strided subset of the past with the stride growing across distance bands, which is, as a read pattern, very close to what Tapered Window Attention does: both spend dense resolution nearby and coarsen it with age, and both get a receptive field that grows much faster than the work per query. Two things separate the designs. First, LongNet's dilation is a *read* pattern over a cache that still holds every entry, whereas TWA deletes the keys it skips, which is the harder property and the one a serving system can spend. Second, LongNet mixes several dilation rates in parallel per layer and sums the results, where TWA applies one rate per age band, so what a query sees is a partition of the past rather than an ensemble over it; that is what makes the resident

set a closed form rather than a union. The shape itself is older than either. Exponential histograms (Datar et al., 2002) summarize a stream over a sliding window with buckets whose size doubles with age, a constant number at each size, so the window is covered in logarithmic space at a resolution that coarsens with age; TWA is the same pyramid over keys rather than counts, and the mipmap of Section 2.1 is its two-dimensional ancestor. We take the resemblance as a point in the design's favor — the coarsen-with-age pattern has been arrived at more than once — and note that the contribution here is bounding what is *stored* under it in an attention cache, not the pattern.

Why we do not also change the local slots. An interleaved stack has two kinds of layer, and the hybrids above differ from Gemma 4 in the *local* one—gated delta rules or SSM blocks where Gemma 4 puts windowed attention. It is fair to ask why SpMem is not evaluated on such a base model. The structural reason is that the local slot is not where long context is decided: a windowed or recurrent local layer is bounded by construction, so the $O(L)$ term lives entirely in the global layers, and in Gemma 4 12B those are 8 of 48. Whatever fills the other 40 changes the constant, not the asymptote (Section 4). The empirical reason is that the case for swapping them is weaker than its adoption suggests. Jolicoeur-Martineau et al. (2026) run the comparison the linearization literature had skipped and find that plain sliding-window attention with attention sinks, applied *training-free*, matches post-trained linear attention on short-context reasoning (99% of the baseline average) and beats it outright on long context: at a 256-token window and 4K context, SWA recovers 17–23% of the full-attention baseline on Single-NIAH and 25% on BABILong, where the linearized models reach at most 5.8% and 5%. Two caveats keep this from proving more than it does, and both matter here. Their study covers *retrofitting* an existing model, and explicitly allows that linear attention trained from scratch may fare differently—which is what Qwen3-Next and Kimi Linear do. And they state that they did not evaluate hybrids in which some layers keep full attention, which is exactly the setting SpMem occupies. So we read it not as a verdict on linear attention but as the narrower claim we actually need: in the regime we work in—adapting a released checkpoint—leaving Gemma 4's windowed local slots alone is not a quality concession, and the layer worth changing is the global one.

A.1.2 Sparse and top-k attention

A large family reduces attention cost by attending to a *selected subset* of the history rather than folding it into a recurrent state. Multi-latent attention in DeepSeek-V2 (DeepSeek-AI, 2024) compresses each token's KV with a fixed low-rank projection, while a rapidly growing line makes the selection *learned and top-k*: Native Sparse Attention (Yuan et al., 2025) combines coarse block compression with fine-grained token selection and is trained end to end from the outset, MoBA (Lu et al., 2025) routes each query to a mixture of key blocks, Token Sparse Attention (Jo et al., 2026) sparsifies each head's queries, keys, and values to a reduced token set, and SSA (Shen et al., 2026) aligns sparse and full attention outputs to close the sparse–dense gap. InfLLM-V2 (Zhao et al., 2026) is the closest prior work on the axis SpMem cares most about: it reports that NSA's added parameters and its departure from the standard pretrain-on-short, finetune-on-long workflow cost convergence speed, and it removes the added parameters entirely by reusing the dense attention weights under a switchable dense/sparse mask. It nevertheless selects among retained blocks, so its resident cache is still $O(L)$; SpMem likewise adds no attention parameters, only an $m \times d$ memory-token embedding, but bounds what is stored rather than what is read. Others target the inference-time system cost directly: SparDA (Fu et al., 2026) prefetches the KV blocks a later layer will need, and Still (O'Neill et al., 2026) emits a compacted cache from a small per-layer Perceiver (Jaegle et al., 2021), so that each layer's compression is a separate function reading that layer's full KV rather than a single compressed state refined through depth. Block Transformer (Ho et al., 2024) makes the same choice at coarser grain, splitting the stack so that lower layers aggregate tokens into block representations and upper layers decode within them. In both, the compressing function is separate from the layers that consume its output; SpMem instead compresses through the ordinary forward pass, each layer refining what the ones beneath it left and at its own level of abstraction—the arrangement register tokens (Darcet et al., 2024) exploit in vision transformers and that Section 2.1 argues for here. Streaming eviction methods such as StreamingLLM (Xiao et al., 2024) similarly retain a sliding window plus sink tokens at inference (the training-free eviction line is discussed in Section A.1.6), while Memorizing Transformers (Wu et al., 2022) keep *every* past key–value pair in an external store and attend to an approximate k-nearest-neighbor subset, and Focused Transformer (Tworkowski et al., 2023) trains that retrieval contrastively after identifying a *distraction* problem—as the store grows, keys

with different meanings become harder to tell apart, which is a caution for any content-addressed memory including ours. Landmark Attention (Mohtashami & Jaggi, 2023) also inserts one token per block, but uses it as an index into blocks that are all still retained, preserving random access at $O(L)$ storage; SpMem's summaries instead stand in for the block once its real tokens have aged out. All of these preserve access to a compressed or selected *view* of the full history; SpMem sits at the opposite pole, keeping recent real KVs in full, thinning older ones under TWA, and leaving only its learned summaries past the taper's reach—trading random-access recall for a hard memory bound. The two lines are complementary: sparse and top-k attention change *how* a retained cache is read, whereas SpMem changes *what* is retained in the first place.

A.1.3 Conditional computation

A related axis allocates *compute* per token rather than compressing memory: Mixture-of-Depths (Raposo et al., 2024), Mixture-of-Recursions (Bae et al., 2025), GateSkip (Laitenberger et al., 2026), CoLT5 (Ainslie et al., 2023), which routes long-input tokens to heavy or light feedforward and attention branches, and—closest to SpMem's concern—L2A (Choudhary et al., 2026), which learns *when* a token needs global attention. These methods leave the KV cache's size largely intact, whereas SpMem targets the footprint directly.

A.1.4 Recurrent memory and state compression

Explicit memory predates the long-context problem: memory networks (Weston et al., 2015) attached a readable and writable store to a neural network, and the idea reappears whenever a model has to carry state past what its window holds. In streaming speech recognition, Emformer (Shi et al., 2021) distills long-range history into an augmented memory bank carried across segments, which is the same shape as SpMem's pool arriving from a different application. A separate line uses memory for *knowledge* rather than context—product-key memory layers (Lample et al., 2019) add capacity a model can look facts up in, which does not bound a KV cache and is not what we mean by memory here.

Recurrent Memory Transformer (RMT) (Bulatov et al., 2022) is a structural ancestor of SpMem. Both segment the input into fixed-size chunks, introduce special memory tokens at segment boundaries, use standard attention with no new attention weights, learn a small embedding for the memory tokens, and train with next-token prediction and no explicit memory supervision. SpMem differs in two ways that matter at scale. RMT *prepends* memory tokens, which requires non-causal access and a separate backward pass per segment, whereas SpMem *appends* them and stays cleanly causal; and RMT operates at every layer, whereas SpMem compresses only on global-attention layers, leaving local layers to handle within-chunk detail. Infini-attention (Munkhdalai et al., 2024) and Titans (Behrouz et al., 2024), the latter extended by the nested-optimization view of Behrouz et al. (2025) in which memory is organized over multiple timescales, likewise maintain a compressive recurrent state, but introduce dedicated memory-update mechanisms or additional parameters rather than reusing the base attention operation. Memorizing Transformers (Wu et al., 2022) bear on a question SpMem has to answer: at which depths memory is needed. They attach a kNN store to a single layer and ablate its index, finding the middle of the stack best and placements near the input or output worse, and report that a second retrieval layer brought no further benefit. Their memory is a lossless external store over all past keys, so removing it from a layer costs that layer retrieval help and nothing more; in SpMem the memory tokens are the *only* path beyond TWA's reach, so a global layer without them is simply local, and the finding does not transfer as a license to use fewer. What it does motivate is keeping a subset of global layers at full attention (Appendix A.17), together with a prior on which subset to choose. TransformerFAM (Hwang et al., 2024) instead adds a feedback loop that lets the model attend to its own latent activations as working memory with no new weights. Its feedback state is maintained per layer rather than carried at a sequence position through the stack, which places it on the layer-wise side of the contrast in Section 2.1; and unlike SpMem, it targets unbounded sequence processing rather than a hard KV-cache budget, and it does not evict the original tokens' KVs. Trellis (Karami et al., 2025) is a recent instance of this test-time-memorization line: it replaces the KV cache with a fixed-size memory and learns a two-pass recurrent compressor with a forget gate, updated by online gradient descent as tokens stream in. Like Titans it is trained *from scratch* and realizes compression as a native architectural mechanism with a learned, content-dependent eviction rule; SpMem instead imposes recurrence on a pretrained

checkpoint through vocabulary tokens and the base attention operation, with eviction fixed by the chunk schedule rather than learned online.

A.1.5 VOCABULARY-TOKEN COMPRESSION

Gist token (Mu et al., 2023) is a mechanistically similar prior work: both insert special vocabulary tokens into the sequence, compress a window of context into a smaller number of token representations, train with a standard language-modeling loss and no auxiliary supervision, and never predict the memory tokens themselves. The differences are in purpose and dynamics. Gist compresses a *static* prompt once, is motivated by prompt-caching convenience, and is trained via instruction tuning; SpMem compresses *recurrently* at every chunk boundary, is motivated by KV-cache memory constraints, and is trained via continued pretraining on raw text. The gist mask hides only what precedes the gist positions, so the rest of the sequence attends to its own input in full and the cache is not bounded as the context grows; the authors are explicit that their target is caching and reusing instruction prompts rather than long-range sequence modeling. In this sense SpMem can be read as *recurrent gisting at chunk boundaries under a memory budget*.

Activation Beacon (Zhang et al., 2024) is a close prior method by mechanism and deserves an explicit account. It also inserts special tokens at interval boundaries, condenses the interval's keys and values into them, and *discards the raw activations* once an interval closes, so it bounds what is stored and not merely what is read; and its beacons are sequence positions whose per-layer KVs are retained, which places it on the depth-distributed side rather than the layer-wise one. We therefore do not claim the mechanism as new. The differences are in the training. *Apparatus:* Beacon adds dedicated query, key, and value projections for beacon positions at every layer plus a beacon embedding, and modifies self-attention to route those positions through them; SpMem adds only an $m \times d$ memory-token embedding and changes only the mask. *What adapts:* Beacon holds the base model's parameters fixed and trains only the beacon parameters, which preserves short-context behavior exactly—a real advantage—but obliges the summary to be written for a reader that cannot learn to read it, whereas SpMem uptrains the whole network so that summary and reader co-adapt, at the cost of short-context behavior that must be measured rather than assumed (Table 1). *What survives, and for how long:* the two designs sit on opposite sides of two axes. Beacon retains raw tokens only inside the interval being processed, so across boundaries the beacons are the sole route to the past, whereas SpMem's tapered cache always keeps a thinning sample of earlier reals — Beacon is the stricter one here. Conversely Beacon retains every beacon it produces, so its summary pool grows with the sequence, whereas SpMem tapers the summaries too and is $O(1)$ — SpMem is the stricter one there. Because the two families differ in opposite directions, we report those axes as separate ablations rather than treating one setting of SpMem as a stand-in for Beacon (Section A.2). Beacon further samples its condensing ratio per interval so that one checkpoint serves many ratios, where SpMem fixes $r = 8$ by the chunk schedule; and its training samples are shorter than 20K tokens, with longer contexts reached by extrapolation, where SpMem is trained at the length it is evaluated at.

The soft-token lineage traces to prompt tuning (Lester et al., 2021), which learns a handful of continuous embeddings to steer a frozen model; later work repurposed the same device for *context compression*, and it is against this line that SpMem is most directly measured. AutoCompressor (Chevalier et al., 2023) full-parameter-tunes an LLM to emit summary vectors and *recursively* prepends them to subsequent segments, making it the soft-vector counterpart of SpMem's carry-forward. The differences are that it accumulates continuous summaries rather than reusing a fixed budget of vocabulary tokens, and it operates on the input side as a soft prompt rather than evicting key–value entries from the cache. The In-context Autoencoder (ICAE) (Ge et al., 2023) compresses a context into a few memory slots with a separate encoder that the decoder then reads, trained with an autoencoding *reconstruction* objective; SpMem trains under next-token prediction and never reconstructs the tokens it drops, so its summaries are optimized for downstream prediction rather than for faithful decompression. Dodo (Qin et al., 2023) makes the compression *rate* content-adaptive, using a scorer to retain a variable number of "nugget" states per context; SpMem fixes 1 tokens per boundary, trading Dodo's adaptivity for a static, schedule-driven memory budget that composes cleanly with the local/global attention split. Parallel Context Encoding (CEPE) (Yen et al., 2024) attaches a small separate encoder and cross-attention layers so a frozen decoder can attend to many context chunks encoded *in parallel*. Like LCLM below, its compression is non-recurrent and external to the base model, whereas SpMem keeps a single model and conditions each chunk's summary on all of

its predecessors. Taken together, these methods share SpMem's goal of representing context in far fewer tokens, but each places the compressor *outside* the streaming next-token objective—as a soft prompt, an autoencoder, a learned selector, or a side encoder—while SpMem folds compression into ordinary causal decoding on a checkpoint and carries the result forward as a compressed recurrent cache.

Two concurrent works scale soft-token compression and are the closest contemporaneous neighbors of SpMem. LCLM (Li et al., 2026) continually pretrains an encoder–decoder pair in which a small encoder pools each block of N tokens into a single latent token that the decoder consumes in place of the original span, supervised by next-token prediction on the uncompressed tokens only—the same masked-loss recipe SpMem uses, distributed over interleaved compressed spans. It differs on two structural points. First, LCLM compresses each window *independently* with a *separate* encoder and concatenates the resulting latents, so its compression is non-recurrent and random-access; SpMem carries its memory-token KVs forward within a single model, so each chunk's summary is conditioned on all prior summaries. The compression is also external to the stack, applied once by the encoder rather than refined layer by layer as the representation deepens (Section 2.1). Second, LCLM adds an expand-on-demand fallback that re-materializes a chunk at full fidelity, whereas SpMem commits to its lossy summaries and relies on them alone once a span leaves the real-token window. Cartridges (Eyuboglu et al., 2025) instead trains a compact KV cache *offline and per corpus* through a context-distillation objective ("self-study"), amortizing the training cost across repeated queries to the same documents. SpMem learns a single general recurrent compressor once during continued pretraining rather than an artifact per corpus, and needs no in-context teacher to distill from.

Closely related, Compressed Context Memory (CCM) (Kim et al., 2024a) compresses accumulated context into a compact memory that is updated online during generation. Two differences are structural. CCM drives its compression with a single shared compression-token embedding, whereas SpMem allocates distinct per-slot identities, so the m tokens at a boundary can specialize rather than compete for one role. And CCM holds one compressed memory, whereas SpMem's global layers read a verbatim W-real window *alongside* the pool, which is what ties the scheme to the local/global attention split of the base model and leaves recent context uncompressed by construction. CCM's reported failure mode—the model learning to *ignore* its compressed memory—is precisely the risk SpMem also faces: memory tokens degenerating into attention sinks that carry position but little content.

A.1.6 Training-Free Compression

A complementary line of work reduces attention cost *without any training*, applying a fixed policy at inference time to a frozen model, and it spans both sides of the reads/stores distinction. On the read side, FlexPrefill (Lai et al., 2025), MInference (Jiang et al., 2024) and X-Attention (Xu et al., 2025) select which blocks each query attends to during prefill while keeping the full cache resident. On the store side, eviction methods discard entries: StreamingLLM (Xiao et al., 2024) keeps attention sinks and a recent window, H2O (Zhang et al., 2023) keeps attention-score heavy hitters, SnapKV (Li et al., 2024) selects entries by the attention they receive from an observation window, and KVzip (Kim et al., 2025) prunes query-agnostically while preserving the ability to reconstruct the context. A related line targets a fixed budget under long or streaming inputs: InfiniPot (Kim et al., 2024b) iteratively distills the cache by future relevance and novelty, EpiCache (Kim et al., 2026a) manages it episodically across conversation turns, and BeaconKV (Kim et al., 2026b) compresses it with dedicated beacon queries. These methods need no gradient updates, but they never adapt the model to what they remove, so what they can discard is bounded by what a frozen model already tolerates. FlexPrefill, MInference, X-Attention, StreamingLLM, SnapKV and KVzip are the training-free baselines of Section 4.

A.2 Ablations

Table 4 varies one design choice at a time around the shipped configuration ($W = 4$K, tapered memory cache, $C = 8$, $m = 1$, every global layer bounded), all at step 5K on Gemma 4 12B. Table 6 covers C and m, and Table 15 the number and position of retained global layers.

Table 4: Ablations around the shipped configuration (Gemma 4 12B, step 5K, every global layer bounded). Each block changes one choice; *Budget* is the resident entries per global layer at 128K. Accuracy (%).

Reals	Memory	Budget	LB-v1	LB-v2†	MRCR-v2	NIAH	R-128K
Shipped							
TWA 4K	tapered	16K	34.6	37.4	35.0	60.0	24.7
Window							
TWA 8K	tapered	24K	37.8	34.0	37.2	65.8	27.3
TWA 16K	tapered	40K	38.6	36.1	**45.6**	76.9	**35.0**
Memory cache, at $W = 8K$							
TWA 8K	none	16K	38.6	34.4	35.2	64.4	21.5
TWA 8K	tapered	24K	37.8	34.0	37.2	65.8	27.3
TWA 8K	all within 64K	25K	39.1	32.0	40.4	66.9	29.1
TWA 8K	all	33K	**40.2**	**38.4**	44.3	69.2	33.5
TWA 16K	none	33K	38.8	37.1	44.2	**77.1**	34.0

†Thinking mode.

Table 5: MRCR-v2 by needle count at step 5K, for the configurations of Table 1. Accuracy (%); *Avg.* is the MRCR-v2 column of Table 1.

Model	2 needle	4 needle	8 needle	Avg.
Gemma 4 12B				
Dense	64.1	57.4	**38.2**	53.2
SpMem	39.7	39.0	26.2	35.0
SpMem + 1 global	**69.0**	**59.1**	37.7	**55.3**
Gemma 4 31B				
Dense	62.9	53.5	41.0	52.5
SpMem	51.7	54.6	38.1	48.1
SpMem + 1 global	**64.2**	**63.7**	**44.0**	**57.3**

Window. Widening the real-token window buys retrieval roughly in proportion to the cache: from $W = 4$K to 16K, MRCR-v2 rises from 35.0 to 45.6 and NIAH from 60.0 to 76.9 at $2.5\times$ the entries. Nothing about the recipe favors a small W except its budget.

Memory cache. At $W = 8$K the memory cache orders the same way: none, tapered, every summary within 64K, every summary, with MRCR-v2 at 35.2, 37.2, 40.4 and 44.3. Keeping every summary costs as many entries as doubling the window (33K against a 16K window with no memory's 33K), and the two then tie on MRCR-v2 (44.3 against 44.2); the window keeps more of NIAH (77.1 against 69.2), the summaries more of LongBench (40.2 against 38.8 on v1, 38.4 against 37.1 on v2). With every global layer bounded, then, memory tokens are an alternative to a wider window rather than a better one — the result of Section 5.2, where they pay off only once one global layer is kept.

A.3 Results moved for length

A.3.1 MRCR-v2 by needle count

Multi-round co-reference resolution is reported separately for the 2-, 4-, and 8-needle settings, whose difficulty increases with the number of interleaved key–value pairs the model must track (Table 5).

A.3.2 Choosing the summarization geometry

The chunk size C and the summaries per chunk m are the two free parameters of the memory stream, and only their ratio $r = C/m$ sets how much the sequence is compressed. Whether the ratio is the

Table 6: Summarization geometry at $K = 0$ ($W = 4$K, step 3K), every summary kept to 128K. $r = C/m$; *Budget*: resident entries per global layer at 128K. Accuracy (%).

Geometry	r	Budget	LB-v1	LB-v2†	MRCR-v2	NIAH	R-128K	MMLU	IFEval
C=8, m=1	8	24.6K	**37.4**	28.9	**36.2**	**54.1**	**25.8**	67.1	67.5
C=16, m=1	16	16.4K	32.8	**32.7**	26.3	49.9	16.4	68.9	69.9
C=32, m=1	32	12.3K	34.0	30.3	26.7	49.6	19.1	**70.4**	68.9
C=64, m=2	32	12.3K	30.9	27.2	24.4	50.2	19.3	67.5	69.3
C=128, m=4	32	12.3K	27.3	28.9	25.9	47.7	10.2	65.1	**71.5**

†Thinking mode.

whole story is an empirical question: at a fixed r, a large C asks one summary to stand for a wider span of text, while a small C emits summaries more often and gives the retained set finer granularity to thin. Table 6 walks the ladder from $C8M1$ to $C128M4$ at $K = 0$ with TWA over the reals ($W = 4$K), so the only thing that changes is the geometry. These runs keep every summary out to 128K rather than tapering them, so the memory budget is $128\text{K}/r$ and falls as r rises; the *Budget* column states it, because a comparison across r is also a comparison across cache size.

Retrieval orders with r and short-context accuracy does not. $C8M1$ leads on LongBench-v1, MRCR-v2, NIAH and RULER-128K (36.2 MRCR-v2 against 24–27 for every coarser geometry), but it also holds twice the entries of the $r = 32$ rows, so the ladder cannot separate a finer ratio from a larger cache. MMLU and IFEval move within a few points with no order, so compression at this level costs retrieval rather than general ability. At a fixed $r = 32$ the three geometries are close on MRCR-v2 and NIAH, but $C128M4$ is the worst row on RULER-128K (10.2 against 19) and LongBench-v1: four summaries written together at the end of one 128-token chunk carry less than one summary written after each of four 32-token chunks. We fix $C = 8$, $m = 1$ for the rest of the paper. It is the best geometry on every retrieval axis, and as the finest granularity it lets TWA thin the memory stream at the same resolution as the reals (Section 2).

A.4 Additional details of Experimental setup

Why this base model Frontier open models have converged on interleaving a cheap local operator with a small number of expensive global attention layers, in roughly a 3:1 to 5:1 ratio; Gemma 4 uses 5:1 (`layer_pattern_period` $= 6$, so 8 global layers of 48 at 12B and 10 of 60 at 31B), which is the geometry SpMem is built against. Yang et al. (2025) give the clearest evidence that this is not a compromise: their 3:1 stack, with RoPE sliding-window attention in the local slots and NoPE full attention in the global one, holds parity with a full-attention baseline on short-context benchmarks—ahead on seven of nine, behind on two—while pulling away on RULER at long context, where retrieval falls 22% from 8K to 256K against the baseline's 41%, and training roughly twice as fast at 128K. Parity plus a long-context gain is the whole claim; it does not need to win everywhere, and it does not. Gemma 4 (Gemma Team, 2026) and GPT-OSS (OpenAI, 2025) adopt this design. A second family keeps the same skeleton but fills the local slots with a linear-time operator—gated delta rules or state-space blocks—rather than windowed attention (Qwen Team, 2025; Lieber et al., 2025; Kimi Team, 2025).

The local operator is a separate question from ours. Whether a local slot holds windowed attention, a gated delta rule or a state-space block, its state has a fixed size, so the choice moves quality and a constant cost but not how memory grows with L; that is set by the global layers alone. The hybrids built on a linear-time local operator keep full-attention layers beside it for this reason — retrieval over the whole context still goes through them — so they carry the same $O(L)$ term that SpMem removes.

We use Gemma 4 because its local slots are plain windowed attention, the simplest case, so nothing but the global layers changes between our runs. SpMem changes only the mask and the cache of the global layers, so it applies unchanged to a hybrid with gated-delta or state-space local layers. We have not trained that combination and do not claim its numbers; Appendix A.1.1 gives the evidence that, for a retrofit, windowed local layers are not the weak choice they are sometimes taken to be.

Optimizer details The objective is ordinary next-token prediction, but the optimizer is not a generic one: it reproduces the constraint Gemma 4 itself was trained under. Weights are updated by a LaProp-style rule—second-moment EMA $\beta_2 = 0.995$, momentum $\beta_1 = 0.95$ applied to the *normalized* update rather than the raw gradient, $\epsilon = 10^{-30}$, raw-update clipping at 1.0—after which every 2D weight is projected back onto the unit sphere, and weight decay is therefore 0 (it would fight the projection). All 1D parameters (RMSNorm scales, layer scalars, router scales) are frozen at their checkpoint values. Peak learning rate is 10^{-5} on a cosine schedule with linear warmup, decaying to 10% of peak: 500 warmup steps over 5K steps for the SpMem variants, and 1K over 50K for the dense reference (Appendix A.8), at a global batch of 32 sequences of 128K tokens. The 12B variants run on a v5p-256 slice and the 31B variants on v5p-512, sharded $(\text{data}, \text{fsdp}, \text{seq}) = (-1, 8, 8)$ with an additional model $= 4$ axis at 31B.

Nothing in that recipe is specific to SpMem: it is the base model's own training constraint, applied unchanged so that the compressed and dense variants share the optimizer and objective and differ in the attention mask and the schedule length. This is what the claim that SpMem needs no special training recipe means—the *objective* is unchanged, not that the optimizer is vanilla AdamW.

A.5 Inference-efficiency measurements

Why SpMem cannot be faster at short context. Below 32K SpMem is not faster than dense, and at 8K it is 4% slower (15.9 vs. 15.3 ms; the deficit is 0.7 ± 0.2 ms over five independent runs per variant, so it is a real effect and not noise). It is also not a fixable one. A batch-1 decode step reads the model's 22.2 GiB of weights whatever the context is, and at 8K the entire KV cache is 1.9% of the bytes the step touches. The difference between the two caches—all that a cache method can possibly recover—is 0.26% of the step, or 0.04 ms. Any per-step bookkeeping above that threshold shows up as a crossing, and maintaining two pools instead of one contiguous buffer costs more than 0.04 ms in XLA-level operations. The cache's share of the step is 3.5% at 32K, where the variants tie, and 42% at 1M, where the bound is worth $2.29\times$. Two structural facts set that threshold: only 8 of Gemma 4's 48 layers are global, so the 40 sliding-window layers (320 MiB of cache) are shared by both variants and cannot be improved by a global memory bound; and dense's global cache at 8K is 128 MiB against a 22.2 GiB weight read.

The cache budget does not move latency here, and should not. The three variants sit within 0.5 ms of one another at every length, which is why Figure 4 shows what looks like a single SpMem curve. This is what Eq. 3 predicts rather than a null result. Their budgets are 12.3K, 8.2K and 6.1K entries per global layer, so the resident set moves by $2\times$ across the three while staying under 1% of the bytes a decode step streams at this batch: the term that would separate them is not the term that dominates. A larger budget buys reach in L, not latency. It separates once the cache displaces the weights: at 128K and batch 128 the same three variants order monotonically, 261.8/248.3/240.8 ms (Appendix A.11).

A.6 Each mechanism at a matched budget

Table 2 picks the best configuration at each step of the argument, and those configurations do not hold equal budgets. Table 7 is the controlled version: each pair holds the KV budget fixed and differs in whether the cache keeps a plain window or tapers.

The taper wins in both pairs. At $16\times$ it is worth $+9.6$ MRCR-v2 and $+5.1$ LongBench-v2, against a window that had the extra retrieval data. At $8\times$, with memory tokens present, it is worth more: $+15.2$ MRCR-v2 and $+11.6$ NIAH. TWA is therefore not only a better standalone cache but a better partner for the summaries. Memory tokens themselves add $+8.2$ MRCR-v2 and $+11.9$ NIAH to TWA, though that comparison crosses a doubling of the budget, from $16\times$ to $8\times$.

A.7 The octave schedule, and why we did not keep it

TWA thins the past with a schedule, and there is an obvious way to make that schedule reach further. Hold the count per level fixed and *double* the span each level covers: budget stays linear in the level count, reach becomes exponential, and a fixed cache spans the whole context instead of a prefix of

Table 7: Each mechanism turned on separately, at matched budget. K is how many of the 8 global layers are left at full attention, and *Ratio* is the realized KV reduction at 128K. The two $16\times$ rows isolate the taper alone and the two $8\times$ rows isolate it with memory tokens present. The sliding-window row at $16\times$ was trained with added synthetic retrieval data, the only run of that configuration; every other row uses the main mixture. Any $K \geq 1$ reintroduces an $O(L)$ term, so those rows measure what an unbounded layer buys rather than a bounded-memory configuration, and *Dense* is the top of the same ladder at $K = 8$. Accuracy (%) at step 5K.

Reals	Mem	K	Ratio	NIAH[‡]	MRCR-v2	LB-v1	LB-v2[†]
Taper alone, at $16\times$ *— 8,192 keys against 8,191 slots*							
SWA	×	0	16.0×	47.8	17.2	**32.5**	35.0
TWA	×	0	16.0×	**48.1**	**26.8**	32.1	**40.1**
Taper with memory tokens, at $8\times$ *— 16,384 slots against 16,382*							
SWA	✓	0	8.00×	48.4	19.8	30.8	34.7
TWA	✓	0	8.00×	**60.0**	**35.0**	**34.6**	**37.4**
$K \geq 1$*: retained global layers — an* $O(L)$ *term returns*							
TWA	✓	1	4.27×	**90.1**	**55.3**	38.6	**39.5**
TWA	✓	2	2.91×	88.1	53.1	**39.7**	36.7
Dense	×	8	1.00×	95.8	53.2	34.6	42.5

[†]LongBench-v2 in thinking mode. [‡]NIAH is the mean of RULER's eight retrieval subtasks over all six length tiers, as in Table 1.

one. We built it, and at the shipped budget it does exactly that: the same 8K slots reach 128K of context instead of 49K (Figure 5).

At a matched budget it is worse. Table 8 runs the full SpMem under each rule, applied to both caches, with everything else held fixed; the schedule that covers the whole context loses to the one that covers 38% of it on every axis but one.

Table 8: SpMem under each retention rule, applied to both caches, with a sliding window as the control. Same geometry ($C = 8$, $m = 1$), positional scheme, data and step (5K); budgets matched to two slots. Accuracy (%).

Rule	Slots	LB-v1	LB-v2	MRCR-v2	NIAH	RULER	RULER-128K	MMLU
mipmap (shipped)	16,382	**34.6**	**37.4**	**35.0**	**60.0**	**65.0**	24.7	**67.5**
octave	16,384	31.1	29.6	33.5	50.4	56.6	**25.5**	65.6
sliding window	16,384	30.8	34.7	19.8	48.4	52.4	16.4	67.2

The one axis octave wins is the one its reach is for: RULER at the full 128K, by 0.8. Everywhere else mipmap leads, by 7.8 on LongBench-v2, 9.6 on NIAH and 8.4 on RULER overall. Reach is worth something — octave clearly beats the sliding-window control on retrieval, 33.5 against 19.8 on MRCR-v2 and 25.5 against 16.4 on RULER-128K — but it is not the binding constraint. Octave buys its reach by spending every level on a wider span, which halves the verbatim window (2,048 keys against mipmap's 4,096) and thins the *near* past to pay for the far one; mipmap keeps more of what is close and simply stops, and leaves the far past to the memory cache. On these benchmarks the near past is worth more, and the ordering is mipmap, then octave, then a plain window.

A.8 Why the dense reference is our own run, not the released checkpoint

Every *dense* row in this paper is a model we uptrained ourselves, not the released instruction-tuned checkpoint, and the two differ substantially. Continued pretraining on our mixture moves the base model in *both* directions (Table 9).

The mixture includes web crawl, and uptraining an instruction-tuned checkpoint on it causes forgetting, but it also teaches multi-round retrieval, which is what the MRCR-v2 gain is. Every SpMem

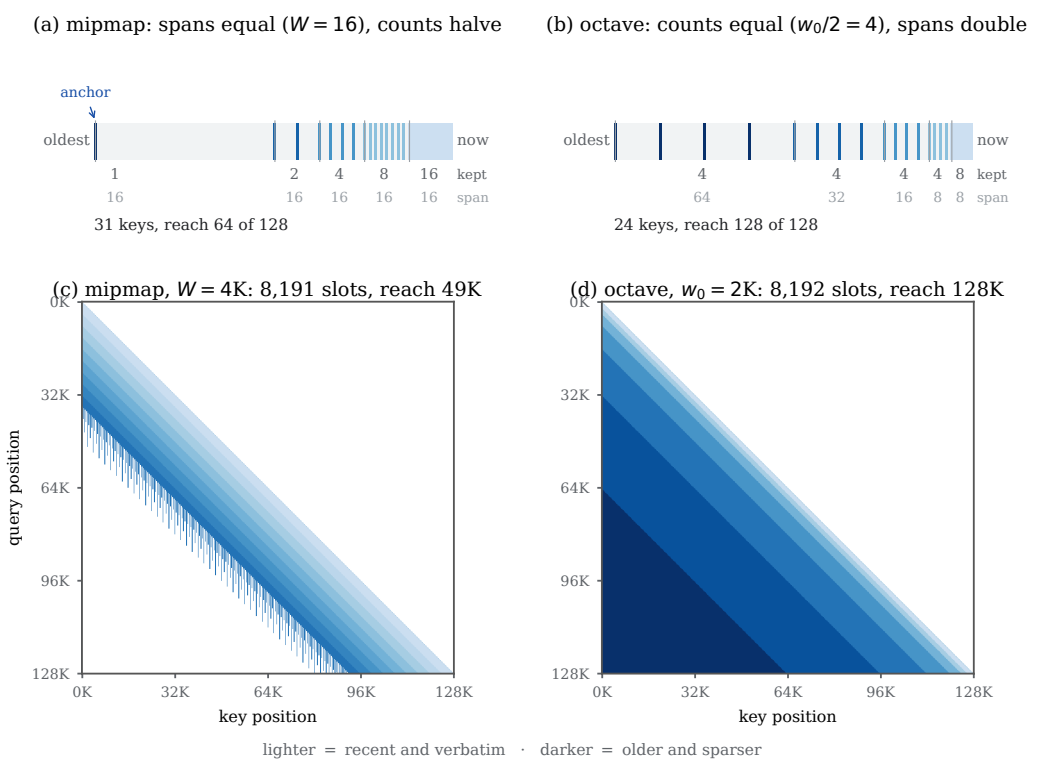


Figure 5: The two schedules at the same budget. **(a)** mipmap holds each level's span at W and halves the count per level. **(b)** octave holds the count at $w_0/2$ and doubles the span. **(c, d)** the same two rules at the shipped 8K budget over a 128K context, which is where the difference is visible: the same number of slots buys 49K of reach under one rule and 128K under the other. Panel (c) is the shipped rule; Figure 7 draws it again with the memory cache beside it. Table 8 is the part that matters — the extra reach did not pay.

Table 9: The released Gemma 4 12B-IT against our uptrained dense model (step 5K) on three long-context benchmarks. Continued pretraining costs LongBench-v1 and RULER but gains MRCR-v2.

	Released 12B-IT	**Our dense (step 5K)**	Δ
LongBench-v1	43.9	34.6	−9.3
RULER	95.3	87.5	−7.8
MRCR-v2	39.7	**53.2**	**+13.5**

variant trains on the same mixture with the same recipe, so comparing against the released model would charge the attention mask for damage the data did and credit it with gains the data gave. The uptrained dense variant is therefore the reference throughout; the released model appears only where it is useful as an absolute point of comparison, labeled as such.

The 12B dense reference is one run, read at step 5K in Section 5 and Figures 1 and 3. It shares the SpMem variants' data, optimizer, learning rate, batch and positional scheme, and differs only in its cosine schedule (50K steps with 1K warmup, against 5K with 500), so at step 5K it is earlier in its decay. The 31B dense reference is read at step 5K as well. Appendix tables that use another step say so in their caption.

A.9 KEEPING THE WEIGHTS ON THE SPHERE

Section 4 says the optimizer projects every 2D weight back onto a sphere after each step. This section says why, because the constraint is not ours and getting it wrong degrades training slowly enough to be missed.

Gemma 4's released checkpoints appear to have been trained under a weight-norm constraint. This is an inference from the weights rather than something documented: inspecting the released 31B base checkpoint, every 2D weight sits at elementwise standard deviation $\approx 1/\sqrt{\text{fan_in}}$ with no drift across depth, a fixed point that Adam-family updates do not produce on their own, since they let the weight norm grow freely. Continued pretraining with an unconstrained optimizer therefore starts the model off a manifold it was trained on, the weights drift, and quality degrades progressively rather than diverging outright. We reproduce the constraint as a post-processing step on the optimizer update:

- **Token embeddings are constrained per row.** Each token's d-vector is projected to unit L_2 norm. In the released checkpoint this is tight — coefficient of variation $\approx 2 \times 10^{-4}$ across rows, which is an exact constraint rather than a tendency. SpMem's dedicated $m \times d$ memory-token embedding takes the same per-row projection, so the memory tokens live on the same sphere as the vocabulary.
- **Every other 2D weight is constrained per matrix**, not per row: the whole matrix is rescaled to elementwise standard deviation $1/\sqrt{\text{fan_in}}$, equivalently Frobenius norm $\sqrt{\text{fan_out}}$. Per-row projection would be wrong here — the released per-row norms are *not* tight (coefficient of variation up to 0.3), so that spread is learned per-neuron structure and normalizing it away destroys information.
- **One-dimensional parameters are not constrained and are frozen.** RMSNorm scales, the per-layer magnitude gate and the router scales carry no such fixed point, some are not trained in the base recipe at all, and we do not attempt to reproduce dynamics we cannot observe.
- **Weight decay is** 0, since it would pull against the projection.

It is worth noting that fan_in is the einsum contraction axis and does not in general match the parameter's sharding factorization. `O_proj`, for instance, factorizes on axis 0 while its true fan-in is axes $(1, 2)$. The projection keys off the parameter path for this reason.

A.10 RULER PER-SUBTASK BREAKDOWN

Tables 10 and 11 break RULER (Hsieh et al., 2024) down by subtask and context length for the three configurations of Table 1, at the same steps. The 13 subtasks fall into four categories: needle-in-a-haystack retrieval (single, multi-key, multi-value and multi-query), multi-hop variable tracking, aggregation (common- and frequent-word extraction) and question answering. Table 1's NIAH column is the mean of the eight NIAH rows over all six lengths, and each block's 128K average is the RULER-128K of Table 2. Dense rows are our uptrained dense models, not the released checkpoints.

Two features of these tables stand out. Accuracy tracks dense while the context is within about twice the verbatim window and departs from it beyond, which places the deficit in the memory pathway rather than in attending under a sparse mask. And the gap to dense does *not* close with model scale: at 128K it is 54 points at 12B and 44 at 31B with every global layer bounded, and 13 against 17 with one kept. A larger model is not a substitute for a wider memory pathway, although the 31B variants also differ in window (8K), so the comparison is not at fixed mask.

A.11 INFERENCE EFFICIENCY ON THE BATCH AXIS

Section 5.3 sweeps context length at a fixed batch, because that is the axis Eq. 3 makes a claim on. The batch axis answers a different, serving-side question—how much of one chip a model can fill at a given length—and it is also where the cache budget becomes visible, since the cache only competes with the weights once it is large enough. Figure 6 sweeps batch at a fixed 128K context, with both variants on standard attention as in the main figure.

Table 10: RULER per-subtask accuracy (%) at 12B, step 5K, for the three configurations of Table 1.

Category	Subtask	4K	8K	16K	32K	64K	128K	Avg.
Dense 12B, uptrained ($O(L)$ cache)								
NIAH	Single-1	100.0	100.0	100.0	98.0	100.0	100.0	99.7
	Single-2	100.0	100.0	99.0	98.0	99.0	99.0	99.2
	Single-3	99.0	100.0	99.0	98.0	97.0	98.0	98.5
	MultiKey-1	92.0	88.0	82.0	71.0	85.0	81.0	83.2
	MultiKey-2	100.0	99.0	97.0	100.0	96.0	95.0	97.8
	MultiKey-3	99.0	98.0	100.0	100.0	98.0	91.0	97.7
	MultiValue	98.2	96.2	97.5	86.0	98.2	92.5	94.8
	MultiQuery	99.0	95.7	95.2	94.5	97.0	91.0	95.4
Tracing	Variable Tracking	99.0	83.4	50.8	41.8	11.8	31.8	53.1
Aggregation	Common-Word Extraction	99.5	98.8	95.8	88.2	66.8	29.7	79.8
	Frequent-Word Extraction	94.3	91.3	95.3	96.3	87.0	92.3	92.8
QA	QA-1 (SQuAD)	92.0	92.0	84.0	82.0	85.0	69.0	84.0
	QA-2 (HotpotQA)	68.0	64.0	57.0	65.0	61.0	54.0	61.5
Average		95.4	92.8	88.7	86.1	83.2	78.8	87.5
SpMem 12B, every global layer bounded								
NIAH	Single-1	100.0	100.0	93.0	77.0	54.0	30.0	75.7
	Single-2	100.0	100.0	80.0	43.0	29.0	14.0	61.0
	Single-3	100.0	99.0	52.0	23.0	10.0	14.0	49.7
	MultiKey-1	99.0	92.0	72.0	36.0	12.0	14.0	54.2
	MultiKey-2	100.0	100.0	91.0	66.0	47.0	8.0	68.7
	MultiKey-3	99.0	95.0	51.0	34.0	16.0	9.0	50.7
	MultiValue	99.5	99.0	83.2	48.0	22.2	7.2	59.9
	MultiQuery	100.0	98.7	82.5	49.2	18.5	11.0	60.0
Tracing	Variable Tracking	100.0	99.4	96.8	79.0	57.4	25.0	76.3
Aggregation	Common-Word Extraction	100.0	99.4	97.9	77.6	63.1	28.8	77.8
	Frequent-Word Extraction	90.7	97.0	95.0	92.7	89.0	95.0	93.2
QA	QA-1 (SQuAD)	89.0	80.0	86.0	57.0	37.0	33.0	63.7
	QA-2 (HotpotQA)	73.0	70.0	61.0	46.0	43.0	32.0	54.2
Average		96.2	94.6	80.1	56.0	38.3	24.7	65.0
SpMem 12B + 1 global layer at full attention								
NIAH	Single-1	100.0	98.0	100.0	100.0	100.0	100.0	99.7
	Single-2	100.0	99.0	100.0	100.0	97.0	95.0	98.5
	Single-3	100.0	100.0	99.0	99.0	96.0	98.0	98.7
	MultiKey-1	100.0	96.0	91.0	62.0	50.0	45.0	74.0
	MultiKey-2	100.0	100.0	98.0	91.0	83.0	60.0	88.7
	MultiKey-3	100.0	100.0	95.0	90.0	68.0	33.0	81.0
	MultiValue	98.7	99.5	98.0	93.3	80.8	69.2	89.9
	MultiQuery	100.0	100.0	96.0	90.2	78.2	75.7	90.0
Tracing	Variable Tracking	100.0	99.6	97.0	89.8	74.0	59.0	86.6
Aggregation	Common-Word Extraction	99.6	99.5	93.9	86.4	62.1	40.2	80.3
	Frequent-Word Extraction	96.0	95.7	97.3	96.7	95.0	98.7	96.6
QA	QA-1 (SQuAD)	96.0	85.0	91.0	58.0	39.0	36.0	67.5
	QA-2 (HotpotQA)	75.0	77.0	71.0	63.0	52.0	40.0	63.0
Average		97.3	96.1	94.4	86.1	75.0	65.4	85.7

Table 11: RULER per-subtask accuracy (%) at 31B, step 5K, $W = 8$K, for the three configurations of Table 1.

Category	Subtask	4K	8K	16K	32K	64K	128K	Avg.
Dense 31B, uptrained ($O(L)$ cache)								
NIAH	Single-1	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Single-2	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Single-3	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	MultiKey-1	100.0	99.0	100.0	99.0	98.0	98.0	99.0
	MultiKey-2	100.0	100.0	100.0	100.0	100.0	98.0	99.7
	MultiKey-3	100.0	100.0	100.0	98.0	97.0	80.0	95.8
	MultiValue	99.3	97.5	98.2	95.7	95.7	96.7	97.2
	MultiQuery	100.0	100.0	100.0	100.0	100.0	99.8	100.0
Tracing	Variable Tracking	100.0	100.0	99.8	100.0	100.0	100.0	100.0
Aggregation	Common-Word Extraction	100.0	100.0	99.8	92.1	86.7	18.9	82.9
	Frequent-Word Extraction	99.7	99.7	100.0	98.7	95.7	99.0	98.8
QA	QA-1 (SQuAD)	96.0	90.0	87.0	83.0	83.0	76.0	85.8
	QA-2 (HotpotQA)	70.0	70.0	70.0	65.0	65.0	58.0	66.3
Average		97.3	96.6	96.5	94.7	93.9	86.5	94.3
SpMem 31B, every global layer bounded								
NIAH	Single-1	100.0	100.0	100.0	99.0	88.0	70.0	92.8
	Single-2	100.0	100.0	100.0	68.0	39.0	20.0	71.2
	Single-3	100.0	100.0	100.0	52.0	17.0	21.0	65.0
	MultiKey-1	99.0	100.0	100.0	65.0	25.0	18.0	67.8
	MultiKey-2	100.0	100.0	99.0	94.0	77.0	25.0	82.5
	MultiKey-3	100.0	100.0	100.0	86.0	32.0	20.0	73.0
	MultiValue	98.7	98.2	98.7	68.8	26.0	12.5	67.2
	MultiQuery	100.0	100.0	100.0	69.0	31.5	15.7	69.4
Tracing	Variable Tracking	100.0	100.0	100.0	100.0	97.4	72.2	94.9
Aggregation	Common-Word Extraction	100.0	100.0	99.5	99.5	97.9	89.5	97.7
	Frequent-Word Extraction	99.7	99.0	100.0	98.7	88.7	97.0	97.2
QA	QA-1 (SQuAD)	97.0	93.0	96.0	76.0	70.0	51.0	80.5
	QA-2 (HotpotQA)	75.0	75.0	77.0	67.0	52.0	41.0	64.5
Average		97.6	97.3	97.7	80.2	57.0	42.5	78.7
SpMem 31B + 1 global layer at full attention								
NIAH	Single-1	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Single-2	100.0	100.0	100.0	99.0	100.0	92.0	98.5
	Single-3	100.0	100.0	100.0	99.0	98.0	98.0	99.2
	MultiKey-1	100.0	99.0	100.0	91.0	64.0	61.0	85.8
	MultiKey-2	100.0	100.0	100.0	96.0	79.0	27.0	83.7
	MultiKey-3	100.0	100.0	100.0	97.0	56.0	27.0	80.0
	MultiValue	98.5	98.7	99.3	97.0	92.0	64.5	91.7
	MultiQuery	100.0	100.0	100.0	99.5	91.7	72.5	94.0
Tracing	Variable Tracking	100.0	100.0	100.0	99.6	94.0	77.6	95.2
Aggregation	Common-Word Extraction	100.0	100.0	99.4	99.3	86.2	79.3	94.0
	Frequent-Word Extraction	99.7	100.0	100.0	98.7	84.0	98.7	96.8
QA	QA-1 (SQuAD)	97.0	95.0	97.0	83.0	77.0	61.0	85.0
	QA-2 (HotpotQA)	79.0	79.0	80.0	72.0	60.0	50.0	70.0
Average		98.0	97.8	98.1	94.7	83.2	69.9	90.3

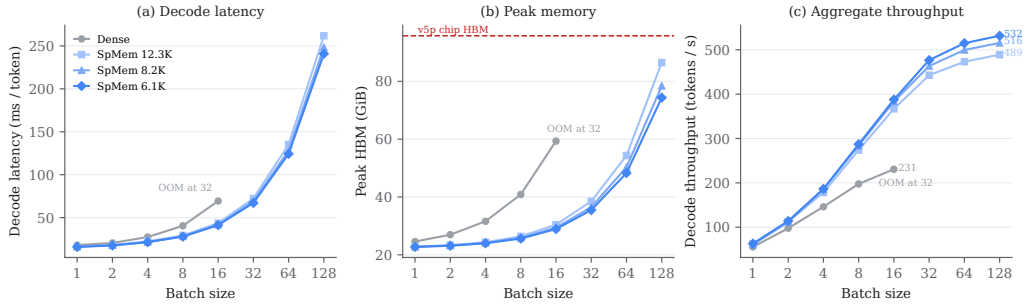


Figure 6: Inference efficiency against *batch size* at a fixed 128K context, on one TPU v5p chip (Gemma 4 12B, bf16, standard attention on both variants): (a) decode latency per generated token, (b) peak HBM against the chip's capacity, (c) aggregate decode throughput. Dense runs out of memory at batch 32; SpMem reaches 128. Compare Figure 4a, b, which fixes batch and sweeps context.

Two things it shows that the context sweep cannot. First, the budgets separate once the cache dominates: at batch 128 the three variants order monotonically, 261.8/248.3/240.8 ms/token at 12.3K/8.2K/6.1K entries, an 8.7% spread in the predicted direction. Second, the batch ceiling: peak HBM grows at 2.31 GiB per sequence for dense against 0.44 GiB for SpMem at 8.2K, so dense stops at batch 16 and runs out of memory at 32 while SpMem reaches 128. Decode throughput is batch divided by latency, and it is where the two effects compound: dense peaks at 231 tokens/s at batch 16 and cannot go further, while SpMem at 8.2K reaches 516 tokens/s at batch 128—a 2.2× gain at equal context on equal hardware, from a ceiling the dense variant cannot reach at all.

A.12 Additional evaluation at one million tokens

Section 5.4 widens the memory cache at inference on the shipped SpMem checkpoint ($W = 4$K, one global layer at full attention). Table 12 repeats that experiment on a checkpoint trained at $W = 16$K, and gives LongBench-v2 and BABILong (Kuratov et al., 2024) at 1M for both. The 1M runs use a splash-attention kernel that never materializes the mask table, which at this length no longer fits in on-chip memory.

Table 12: SpMem with one global layer at full attention, at ∼1M tokens and step 5K, for two real-token windows W. *Widened* and *all summaries* replace the trained memory cache at inference only. NIAH is single-needle over 100 prompts; the $>$ 512K column counts the 52 whose needle lies more than 512K from the end. Chance on LongBench-v2 is 25.

W	Memory cache	KV red.	NIAH	>512K	LB-v2	BABILong
4K	as trained	7.2×	34	12/52	30.9	5.4
	widened	6.3×	**100**	**52/52**	30.7	4.4
	all summaries	4.1×	98	51/52	31.6	5.8
16K	as trained	6.3×	4	0/52	31.3	7.2
	widened	5.6×	0	0/52	30.3	5.0
	all summaries	3.8×	2	2/52	29.9	7.0

At $W = 4$K, widening the memory cache takes NIAH from 34 to 100. At $W = 16$K the same widening does nothing: the checkpoint finds 0–4 needles whatever memory cache it is given, so the result of Section 5.4 holds at the shipped window and does not transfer to a wider one. LongBench-v2 stays near chance (30–32) and BABILong at 4–7% in every row, at either window.

A.13 How the efficiency numbers were measured

Efficiency claims about a sparse method are easy to inflate by accident, so this section states the measurement protocol for Section 5.3 and Figures 4 and 6 in full. Every number in them comes from one harness, one chip, and one model, and each of the choices below was forced on us by a measurement that came out wrong first.

Setup. One TPU v5p chip (95.7 GiB HBM), Gemma 4 12B, weights in bf16—the serving dtype, not the trainer's fp32 master copy, which would put 48.8 GiB of weights on the chip and distort every cache-to-weights ratio in the figure. Weights are zero-initialized: they change neither the byte counts nor the timings, and initialization drops from 142 s to 0.7 s. Latency is the median over 16 decode steps after warmup. A 128K prompt cannot be prefilled one-shot on one chip (a transient of 68.5 GiB, which rematerialization does not remove), so prefill is chunked at 8,192 reals as a serving stack would do it.

Both variants take the same attention path. The dense global layers are eligible for a fused Pallas decode kernel; SpMem's two-pool cache is not a cache type that kernel accepts. Reporting one variant with the kernel and the other without measures the kernel, not the method, so we disable it on both (the fused decode kernel and the incremental splash-attention kernel) and both variants decode through standard attention—which is in any case the ordinary path for a sliding-window cache. Each row of the released measurement files records which kernel actually ran.

Cache length must be block-aligned, on both variants. The fused decode kernel silently declines a cache whose length is not a multiple of the flash block size (512) and falls back to a reference implementation, logging one line and nothing else. A cache sized L + warmup + steps—the natural shape, and what a serving config computes—trips this at every power-of-two context. Measured at 128K, batch 1: a cache of 131,108 entries decodes at 65.6 ms/token, and the same run with the length rounded up to 131,584 decodes at 18.7 ms/token. We first used this fallback deliberately, to force both variants onto the reference path by de-aligning the buffer, and it inflated the dense variant by up to 9× and produced a spurious out-of-memory point at 1M. All reported numbers instead keep the alignment at 512 on both variants and disable the kernels explicitly.

Long contexts are reached by fast-forward, and it is validated. Prefilling 1M tokens per point is not affordable, so the harness allocates the cache for the target context, advances every `time_step` leaf to its post-prefill value, and decodes from there. Against real prefill on the points where both are possible, fast-forward agrees to within 0.3–2.4% across all four variants. Note that skipping prefill entirely is *not* equivalent and we do not do it: a dense global layer reads only up to `time_step`, so an unadvanced cache measures a shorter context than the label claims (15.95 vs. 18.66 ms/token at 128K).

What batch 1 can and cannot show. At batch 1 a decode step streams the model's 22.2 GiB of weights whatever the context is, so the KV cache is a small share of the bytes it touches, and the difference between two caches is smaller still. That difference is the entire budget a cache method has to work with (Table 13).

Table 13: The share of a batch-1 decode step a cache method can act on. *Cache share*: the dense KV cache as a fraction of the bytes a step streams. *Available*: the part of that a bounded global cache could remove, and its value in a 15.1 ms step.

Context	Cache share	Available	ms of step
8K	1.9%	0.26%	0.04
32K	3.5%	1.83%	0.28
1M	42.4%	41.2%	6.23

This is why Figure 4a ties up to 32K and separates above it, and why the short-context end is not a place to look for a speedup: eight of Gemma 4's forty-eight layers are global (one in six), so the forty sliding-window layers (320 MiB of cache) are common to both variants and untouched by a

global memory bound. The batch axis (Appendix A.11) is where the cache becomes the dominant term.

A.14 THE MASK AT THE SHIPPED SCALE

Figure 2 drew the retention rule at $W = 16$, where every cell is visible. At the shipped scale the same mask is $131{,}072 \times 131{,}072$ and no drawing shows it, so Figure 7 aggregates it: each 256×256 block of (query, key) pairs is colored by the taper level it falls in, which is exact rather than approximate because a level is $W = 4{,}096$ of age wide and a block is 256.

The figure makes one point, and it is the reason SpMem carries two caches rather than one. Panel (a) is a band, not a triangle: the rotating levels run $0 \ldots J - 1$, so a real token is gone once it is more than 49K old and the bottom-left corner of the mask is empty. Panel (b), drawn on the same real-token axis, is a full triangle. A memory slot stands for $r = C/m = 8$ reals, so the same rule reaches 8 times further; its ceiling is 393K, and the shipped cache, built for the 128K training length, keeps only the levels a 128K stream fills and covers exactly the whole context. Panel (c) puts both on one axis: past an age of 49K the blue curve is at zero and the orange one is not.

Two details are worth naming because they are visible. The deepest levels in panel (a) break into vertical stripes rather than a solid tint: level j keeps every 2^j-th position, so past $j = 8$ there is less than one surviving key per 256-wide block and most blocks are genuinely empty. That texture is the rule, not a rendering artifact. And the memory cache has 7,681 slots rather than 8,191: at $L = 128$K only 16,384 summaries have been emitted, which fill levels 0–3, so the cache is built with those four rings and the anchor. The cap stays with the trained model, which is why its memory cache still reaches 128K at 1M (Table 3) until it is widened at inference.

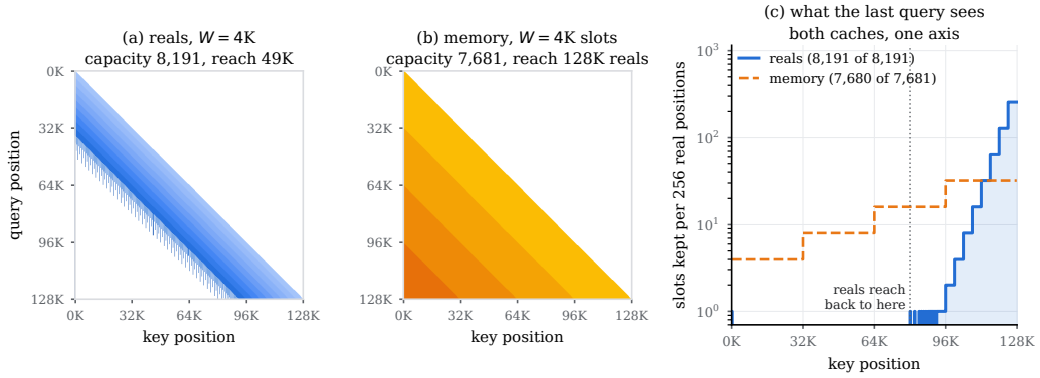


Figure 7: The shipped mask at $L = 128$K, $W = 4$K on both caches, aggregated $256\times$. Lighter is recent and verbatim; darker is older and sparser. **(a)** the real-token cache: 8K slots, and a band rather than a triangle, because the taper's reach is 49K. **(b)** the memory cache on the same axis: 7,681 slots in four levels, each standing for 8 reals, so it covers the 128K context. **(c)** the last query's row through both caches. 16K slots against a dense layer's 131,072 — a $8\times$ reduction, constant in L.

A.15 WHAT THE ATTENTION PATTERN COSTS TO TRAIN

SpMem is a serving method, and its cost model (Section 2.3) is a statement about decoding. Training is a separate question with a separate answer, and the answer is that the pattern does not make training cheaper: it makes it more expensive. We note it because a reader who sees a bounded cache will assume a cheaper step.

The cause is the memory tokens rather than the taper. Splash sorts a grid of 512×512 blocks into empty (skipped), full (no mask work) and partial (the mask materialized elementwise), so what a pattern costs is which blocks it leaves standing and of which kind. With $m = 1$ memory token per $C = 8$ reals a memory key lands in every ninth position, so no standing block is either empty

or full—all of them are partial—and the augmented stream is 12.5% longer besides. Measured on one 12B global layer at 131,072 reals (fwd+bwd, one TPU v5p chip), that is 9.97 s against causal's 6.11 s, $1.63\times$, of which the taper accounts for about 2%: the same geometry with a plain sliding window in place of TWA measures 9.83 s with an identical block composition. Scaling by global attention's share of a step projects roughly $1.36\times$ dense end to end.

A.16 LoRA versus full fine-tuning

Section 4 states that SpMem is adapted by a full fine-tune. The case against a parameter-efficient adapter is that it cannot absorb a change to the attention pattern. Table 14 makes the comparison that decides the recipe: what each adaptation method loses when SpMem is switched on.

Table 14: What each adaptation method loses when compression is turned on, at step 5K. Each method is trained twice, with full attention and with SpMem ($C = 8$, $m = 1$, one global layer kept), on an earlier configuration—NoPE global layers and a sliding window in place of TWA—which is the only one with a LoRA run, so the rows are not those of Table 1. The LoRA variant is rank 128.

	MRCR-v2		**NIAH**		**LB-v1**	
Adaptation	Full	SpMem	Full	SpMem	Full	SpMem
LoRA (rank 128)	45.0	37.8	93.9	81.3	39.9	35.9
Full fine-tuning	**55.4**	**53.0**	93.2	**84.0**	36.3	34.2
Cost, LoRA	-7.2		-12.6		-4.0	
Cost, full	$\mathbf{-2.4}$		$\mathbf{-9.2}$		$\mathbf{-2.1}$	

LoRA falls short twice. With full attention it learns less from the same data — MRCR-v2 45.0 against 55.4 — and when SpMem is switched on it loses more of what it has: 7.2 MRCR-v2 against 2.4, and 12.6 NIAH against 9.2. Full fine-tuning *with* compression ends above LoRA *without* it on MRCR-v2 (53.0 against 45.0). The adapter also stops improving: its MRCR-v2 under SpMem stays between 34 and 39 from step 3K to 5K, while the full fine-tune's rises from 28.7 at 2K to 53.0 at 5K. A frozen base keeps the original model's abilities; it does not learn to use a new attention pattern.

A.17 Retained global layers: count and position

One retained global layer closes most of the retrieval gap (Section 5.2), but every retained layer reintroduces an $O(L)$ memory overhead. Writing K for the number of retained global layers, we ask which layer to keep and whether a second is worth its cost (Table 15).

One layer, and only one position works. At $K = 1$, the position proves impactful. One layer above the center takes NIAH from 56.8 to 98.9 and MRCR-v2 from 39.6 to 50.5. The same layer at the center adds nothing (NIAH 56.0, MRCR-v2 32.0), and at either end it does harm: a first-layer global drops MRCR-v2 to 11.8 and a last-layer one drops NIAH to 33.4. The first global layer has little accumulated below it to read, and the last has no layer above it to use what it reads.

What the second layer buys. One retained global already takes NIAH to 98.9, within 0.2 of dense, so a second adds nothing there (97.6). It helps the other two axes: MRCR-v2 rises from 50.5 to 54.7 and LongBench-v1 from 32.3 to 36.5, both past dense (53.2 and 34.6). A third layer gives both back (50.7, 29.8). At 31B the second layer pays more clearly, taking MRCR-v2 from 57.3 to 64.3 against dense's 52.5 ($W = 8$K, step 5K), so the number of layers worth keeping grows with the model. We nonetheless ship the $K = 1$ configuration. A retained layer is the only part of the cache that still grows, so with K of eight kept the reduction can never exceed $8/K$: a second layer halves the ceiling, from $8\times$ to $4\times$, and already costs it at 128K ($4.3\times \to 2.9\times$).

At 31B. Table 16 repeats the count at 31B (60 layers, 10 globals) on the current architecture, with TWA and a tapered memory cache at $W = 8$K. One retained layer lifts MRCR-v2 from 48.1 to 57.3, past dense's 52.5, and NIAH from 73.6 to 91.6, at $3.7\times$ less cache than dense; a second takes MRCR-v2 to 64.3 and NIAH to 95.2, at $2.9\times$.

Table 15: Retaining K global layers at full attention on top of a sliding window + memory tokens at Gemma 4 12B; the sweep predates TWA. Positions index the 8 global layers: *first* is 0, *last* is 7, *center* is 4. The $K = 1$ rows differ in which layer is kept global. NIAH is single-key; LB-v2 in thinking mode.

Variant	K	**Position**	**NIAH**	**MRCR-v2**	**LB-v1**	**LB-v2**
$K = 0$	0	—	56.8	39.6	27.0	35.4
$f1$	1	first	55.6	11.8	29.7	34.0
$l1$	1	last	33.4	28.9	29.8	28.9
$m1{-}1^*$	1	center -1	30.9	10.6	26.5	22.8
$m1$	1	center	56.0	32.0	31.9	32.0
$m1{+}1$	1	center $+1$	**98.9**	50.5	32.3	31.0
$m1{+}2^*$	1	center $+2$	69.2	13.1	19.9	24.8
m2	2	center, $+1$	97.6	**54.7**	**36.5**	30.3
$m3$	3	center ± 1	98.7	50.7	29.8	**41.8**
Dense (no compression)	8	—	99.1	53.2	34.6	42.5

*Step 1K: these runs were stopped there because they showed no promise; the other rows are at 2.5–5K.

Table 16: Retained global layers at 31B, step 5K, $W = 8$K: the same count sweep as Table 15 on the current architecture. *KV red.*: all 10 global layers against dense at 128K. NIAH as in Table 1; LB-v2 in thinking mode.

Variant	K	**KV red.**	**NIAH**	**MRCR-v2**	**LB-v1**	**LB-v2**
SpMem ($K = 0$)	0	5.3×	73.6	48.1	**51.1**	42.2
$+\, m1{+}1$	1	3.7×	91.6	57.3	50.5	**43.9**
$+\, m2$	2	2.9×	**95.2**	**64.3**	49.8	**43.9**
Dense (no compression)	10	1×	99.0	52.5	47.6	45.6