STATE SPACE ATTENTION: LEARNING CONTEXT-DEPENDENT MEMORY RETENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

The key–value cache used by standard attention grows linearly with context length. Bounding this cache raises a central question: what should a sequence model retain, and how can it learn to make those decisions to support long-context reasoning? We introduce State Space Attention (SSA), an attention architecture that learns to maintain a bounded set of past representations and reads them using standard attention. A learned scorer repeatedly rescores all chunks in light of newly arriving context and the other contents of memory. We train the scorer and reader jointly, gradually moving from fully differentiable, softly weighting candidate memories to retaining a hard, fixed number. Moreover, the scorer receives training feedback based on whether retaining otherwise discarded information would improve the model's predictions. Across our evaluated retrieval and reasoning benchmarks, SSA maintains full-attention Transformer-level performance while its active inference memory remains independent of sequence length. We compare SSA with recurrent models, bounded-memory attention methods, and full-attention Transformers across retrieval, variable tracking, multi-hop reasoning, and language modeling. We further pretrain 400M-parameter models on 15B tokens, and measure the associated computational and memory costs along with language modeling performance. Together, these experiments examine the capabilities and limitations of learned memory retention for long-context sequence modeling.

1 INTRODUCTION

Long-context reasoning requires a model to use information that may have appeared far earlier in a sequence. Transformers make this easier compared to previous natural language processing architectures by retaining key–value representations of every preceding token and allowing later queries to access it directly in computation (Vaswani et al., 2017). This flexibility has, however, a growing memory cost: the cache expands with the sequence, and while each new token can attend to an increasingly large history, it also comes with increasingly larger memory use. Alternatives like State space models (SSMs) and recurrent linear attention offer a different tradeoff (Gu & Dao, 2024; Dao & Gu, 2024; Katharopoulos et al., 2020). They update a fixed-size state as tokens arrive, making inference memory independent of context length. Their memory is bounded, but recovering precise information from that state can be difficult, because successive inputs interfere within the same fixed-size representation (Jelassi et al., 2024).

In this paper, we ask whether and how a model can retain the contextual reasoning capabilities of attention under the same constant-memory constraint. A bounded state need not be a compressed one: a model can keep exact token representations, as long as it caps how many it keeps. Seen this way, a Transformer with context length R already has a bounded state, managed by a fixed first-in, first-out rule that keeps only the most recent R tokens. We propose to relax this constraint, and instead let the model choose which R of the seen context to keep. This adds a dimension of learning on top of basic attention: in addition to learning how to read its history, the model must learn what to keep before it knows which future questions it will need to answer.

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.

We introduce State Space Attention (SSA) as an architecture with a complete training recipe to implement this approach. It replaces the growing attention cache with a fixed-size memory and a learned retention policy, allowing the efficient processing of contexts $\gg R$ with adaptive retention. As the sequence advances, a *scorer*, a transformer, compares incoming information with stored content and decides what to discard. The *language backbone* (also referred to as the *reader*), also a transformer, then reads the retained history together with recent context. This allows the model to process information at contexts up to $24\times$ in our experiments effectively to solve tasks. Both the scorer and the language backbone are trained jointly and end-to-end, from scratch. This was listed as an important future direction by recent work (Yang et al., 2026), although a similar approach for a less-expressive scorer was demonstrated (He & Garner, 2025).

SSA has no separate recurrent memory or external archive at inference and is closer to attention than SSMs; its persistent state consists of the retained representations and the bounded information needed to manage them. Thanks to this, SSA is stackable – it is meant to be a slot-in replacement for standard attention, meaning it is possible to create hybrid models with it. It does, however, also natively include a lossy compression mechanism through the deeper layers' KV values, which were produced with different contexts and thus include information from them. This is expanded upon in §4.3, along with measurements consistent with this.

The main difficulty in training is credit assignment for the scorer. A useful memory can be and often is discarded long before the prediction that reveals its value, and a hard selection decision does not provide an ordinary gradient. Straight-Through Estimation (STE) can provide gradients for values that happen to be retained (Bengio et al., 2013), but assignment of credit for discarded information remains a part that needs to be solved. Moreover, a model that has not learned to use memory cannot give a reliable retention signal, while the reader and the scorer must be trained together. We address these problems with a training recipe that develops the reader through language pretraining and soft scorer training, where the scorer's decisions can be trained in a "low-risk" environment where it can't fully expel information, introducing discrete retention gradually. When discrete retention is introduced, we provide a recovery mechanism that tests whether preserving discarded information would have corrected a training error through counterfactual credit assignment, thus still providing training signal to the scorer. Through this recipe SSA is able to learn in a wide variety of reasoning tasks at varying context length requirements.

We provide experiments that address three questions: whether a bounded attention reader can solve the task with a perfect policy, whether a learned policy can learn to preserve useful information across a long sequence, and whether the training procedure reliably develops both abilities. SSA matches the full-attention baseline on long-context cued retrieval and multi-hop reasoning tasks. Comparisons with fixed eviction rules, where we compare against both fixed eviction that is as memory constrained as the SSA, and fixed eviction that can house the entire needed context (i.e a basic transformer) show that learned retention matters heavily when relevant information must survive beyond the recent window. Comparisons with recurrent models also show that the choice of memory representation matters: even at similar state budgets, SSA is able to not collapse when SSMs do. These findings support selective memory as a practical route to constant-memory reasoning within the studied tasks.

Thus, our contributions are a bounded attention architecture, a procedure for training its reader and retention policy jointly, an analysis of their capability and optimization tradeoffs, and a comparison with current leading constant memory architectures for long context reasoning. Our controlled benchmarks establish strong task performance at small model scales; we scale our models for language modeling and harder benchmarks, and pretrain 400M-parameter models on 15B tokens.

2 Related work

Fixed-state sequence models. Linear attention and state space models replace a growing context cache with an online state update (Katharopoulos et al., 2020; Gu & Dao, 2024; Dao & Gu, 2024). DeltaNet and Gated DeltaNet develop associative recurrent memories with learned updates (Yang et al., 2024; 2025). Transformer-XL and Compressive Transformers instead propagate caches or compressed representations between segments (Dai et al., 2019; Rae et al., 2020), while Infini-attention combines local attention with a compressive associative state (Munkhdalai et al., 2024). SSA shares the fixed inference-state objective, but represents its long-range memory as selected

key–value chunks read by softmax attention, thus changing memory storage from lossy only to exact and lossy; Transformer-XL also keeps exact representations, but only of the most recent segment, whereas SSA keeps a learned subset. This also bridges the understanding between KV cache pruning methods and constant memory architectures.

Cache selection and learned eviction. Methods such as H_2O, TOVA, Scissorhands, SnapKV, and StreamingLLM reduce the cache of a pretrained Transformer using attention statistics or positional rules, targeting the growing memory hurdle of modern attention (Zhang et al., 2023; Oren et al., 2024; Liu et al., 2023; Li et al., 2024; Xiao et al., 2024). Dynamic Memory Compression learns to compress caches through continued training (Nawrot et al., 2024). Expire-Span (Sukhbaatar et al., 2021), dynamic context pruning (Anagnostidis et al., 2023), and Selective Attention (Leviathan et al., 2025) learn which memories to drop, and Landmark Attention (Mohtashami & Jaggi, 2023) learns to retrieve blocks from a full cache. Learned token eviction has also been studied in hybrid linear and sparse-attention models (He & Garner, 2025). HOLA (Hippocampal Linear Attention) combines a recurrent delta-rule state with a bounded exact cache whose retention rule uses the magnitude of the recurrent update (Cui, 2026). IndexMem combines a learned cache indexer with latent memory for evicted content (Yang et al., 2026). SSA studies a complementary setting: the selected attention memory is the sole long-range memory (though it can be combined with other methods), and its reader and retention policy are trained jointly from scratch. We do not claim that learned eviction itself is new, we claim our exact approach and training combination is. Unlike these methods, SSA combines a hard, fixed-size memory in both training and inference, rescoring of every resident against each new candidate, and counterfactual credit for evictions as a new signal path.

Sparse reading and memory evaluation. Quest and Native Sparse Attention reduce the amount of history read by a query, targeting the quadratic compute hurdle of modern attention (Tang et al., 2024; Yuan et al., 2025). Selecting what to read and selecting what to retain are different constraints: restricting reads does not by itself bound the stored history. Copying, associative retrieval, and multi-hop induction provide controlled ways to examine these distinctions (Jelassi et al., 2024; Sanford et al., 2024). Our optimization analysis is also related to delayed learning of intermediate attention operations in multi-step tasks, as it shows up in our training (Hoffmann et al., 2024). We use these tasks to distinguish reader learning from information loss caused by retention decisions.

Cache selection and sparse reading address memory and compute separately; SSA, like SSMs, addresses both.

3 State Space Attention

3.1 A bounded attention state

Let an input of length T be divided into chunks of C tokens. At chunk step t, SSA maintains at most M retained chunks and a local window of w preceding chunks (Figure 1). Including the current chunk, the reader has access to at most

$$R = (M + w + 1)C \tag{1}$$

tokens, independent of T. Chunk membership is shared across layers and heads in the basic recipe, but SSA can be configured to have layer-specific and head-specific chunk membership; and in the basic recipe each layer stores its own keys and values for the chunks held. Queries use ordinary causal attention over the accessible set (the set that contains the R tokens) $\mathcal{S}_t$:

$$p_{qk} = \frac{\exp(a_{qk})}{\sum_{j\in\mathcal{S}_t}\exp(a_{qj})}, \qquad o_q = \sum_{k\in\mathcal{S}_t} p_{qk} v_k, \qquad a_{qk} = \frac{q^\top k + q^\top g_{\mathrm{slot}(k)}}{\sqrt{d_h}}. \tag{2}$$

Here d_h is the head dimension and g is a learned segment embedding for memory slots. Keys preserve their original rotary positions (Su et al., 2024), and the current chunk is masked causally. Empty slots are excluded from the softmax.

When a chunk leaves the recent local window, it becomes a candidate for long-range memory. If memory is full, a scorer regrades all resident chunks and the new candidate and removes the lowest-scoring member. Thus, the candidate itself can be removed, leaving the residents unchanged. Residents' rescoring again at subsequent decisions allows retention to respond to newly observed context.

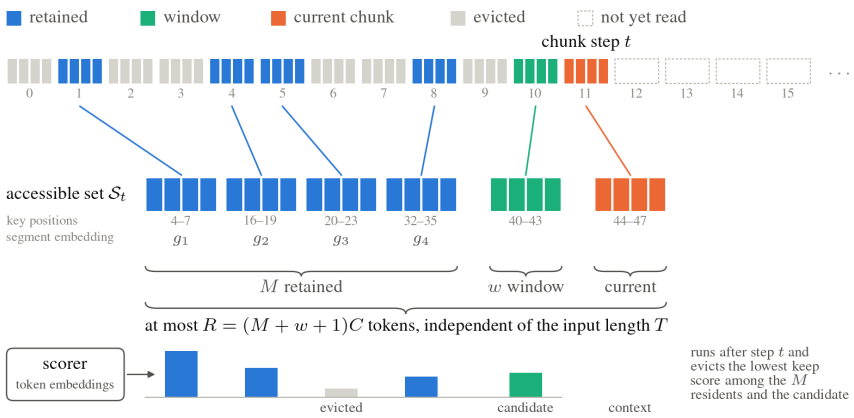


Figure 1: The bounded attention state at chunk step t, for a toy configuration with $M = 4$, $w = 1$, and $C = 4$. Top: the input, with each chunk colored by its role at step t. Middle: the accessible set $\mathcal{S}_t$, shared by all layers and heads: the M retained chunks in slot order, the w window chunks, and the current chunk, which is read causally. Retained chunks keep their original key positions and add their slot's segment embedding g_j. Bottom: The scorer reads token embeddings, rescores the candidate and the M residents with the w most recent chunks as context (here chunk t), and evicts the lowest-scoring chunk (here chunk 5).

The scorer uses token embeddings, positional and segment information, and the recent window; it does not require the reader's hidden states, speeding up training. We also tested scorers that read the reader's hidden states instead of token embeddings, and saw no gain in performance for a large drop in training speed (App. C). Its output is shared across the model. The scorer is small and is meant to be small: two layers of width 160 in the 3M family (0.87M parameters, beside a 3.18M-parameter reader).

For L layers of width d, the reader stores at most $2LdR$ key–value elements per sequence. Token identifiers, positions, and scorer metadata also have bounded size. As such, persistent inference memory is constant in sequence length, and the attention work per token is bounded for a fixed architecture and memory budget, while keeping the quadratic attention formula. This statement, however does not imply constant training activation memory as you need to hold onto past, "discarded" KV caches for backpropagation, and in our experiments we observe at scale memory use that is essentially equivalent to that of a basic transformer during training (Table 10). Evicted keys and values cannot be directly reread, thus we provide a recovery mechanism providing extra signal to train the scorer, as training the scorer is the primary problem. Information from them may nevertheless remain in contextualized representations that were retained, i.e the K/V cache of the layers, even without the recovery mechanism, providing SSA with a lossy information highway as well.

3.2 Learning discrete retention

A hard eviction changes which information later predictions can use. Its effect can therefore be not only immediate but also delayed and discontinuous, where something that was truly unnecessary for steps 1-100 might become necessary on step 101. It is here we run into the problem of teaching the model to keep information like this in context, for steps 1 through 100. We train the scorer using a continuous relaxation early in training, and after transforming to hard training, a straight-through gradient for hard reads along with counterfactual feedback for discarded chunks.

Soft retention warmup. Let $s_c^{(e)}$ be the keep score for chunk c at eviction event e. Negated scores define drop probabilities $p^{(e)} = \mathrm{softmax}(-s^{(e)}/\tau)$. During warmup, cumulative survival weights reduce the contribution of chunks to the attention calculation, rather than removing them:

$$\omega_c^{(e)} = \prod_{e' \in H_c,\, e' \le e} (1 - \lambda p_c^{(e')}), \qquad a_{qk}^{\mathrm{soft}} = a_{qk} + \log \omega_{c(k)}^{(e)}. \tag{3}$$

Equivalently, soft retention multiplies each chunk's attention weight by its survival weight before renormalizing:

$$p_{qk}^{\text{soft}} = \frac{\exp\left(a_{qk} + \log \omega_{c(k)}^{(e)}\right)}{\sum_j \exp\left(a_{qj} + \log \omega_{c(j)}^{(e)}\right)}, \tag{4}$$

where j ranges over all preceding chunks, including those the hard policy would have evicted. H_c contains the events in which c participates. Increasing the eviction strength λ and lowering the temperature τ sharpens selection. This relaxation allows useful chunks to receive credit before retention becomes discrete. It can only retain additional training-only representations; all reported evaluations enforce the fixed memory budget and were run under the constant memory model, including the values reported during training even when the model was receiving soft signal through completing the evaluations in hard mode.

The soft phase also lets the backbone correct the scorer: a down-weighted chunk can still be read, so a strong enough pull toward it sends the scorer a strong signal that the chunk should have been kept.

Hard retention and read gradients. Hard training executes discrete evictions, with stochastic selection from the drop scores for exploration, while evaluation uses the lowest keep score as the chunk to evict. A centered straight-through estimator (Bengio et al., 2013) preserves the hard forward read and differentiates a normalized read weight in the backward pass. The local derivative for a chunk is

$$\frac{\partial \mathcal{L}}{\partial \log \omega_c} = \sum_q \sum_{k \in c} p_{qk}\, g_q^\top (v_k - o_q), \qquad g_q = \frac{\partial \mathcal{L}}{\partial o_q}. \tag{5}$$

This expression measures a chunk's contribution relative to the current attention output. It provides credit through surviving reads, but cannot directly evaluate a chunk that is no longer available. Appendix A gives the derivative and explains the scope of the surrogate.

Counterfactual recovery during training. For selected training failures, recovery replays the example while forcing a previously evicted chunk to remain available. The rest of the policy continues to act under the same capacity constraint. If the resulting prediction passes the recovery acceptance criterion, the scorer receives survival credit and a margin loss favoring that chunk over its counterfactual replacement (App. A). This supplies feedback where read gradients are necessarily absent in the absence of the chunk from the read. Recovery naively uses all discarded chunks. We also provide a faster, cheap search option using the method from Quest (Tang et al., 2024), which ranks candidates with 0.73 AUC against the exhaustive search; this is provided for deployment and downstream use purposes, we do not use it in our experiments, so that it does not become a failure point for the training of the architecture.

Recovery is disabled at inference by default and adds no external archive to the constant-memory model. It can, however, serve as a deployment knob: its thoroughness interpolates between constant memory, with possible irretrievable forgetting, and quadratic memory with guaranteed retrieval, given a sufficiently trained reader.

Counterfactual credit as an estimator of soft credit. Let $A_c = -\partial \mathcal{L} / \partial \log \omega_c$ (equation 5). With the eviction trajectory fixed, the soft model's scorer gradient is the gradient of a credit-shaped surrogate,

$$\frac{\partial \mathcal{L}}{\partial \theta}\Big|_\omega = \frac{\partial}{\partial \theta} \sum_c \text{sg}(A_c) \sum_{e \in H_c} \left[-\log(1 - \lambda p_c^{(e)}) \right]. \tag{6}$$

At $\lambda = 1$, the survival term of the recovery loss (App. A) is this surrogate with A_c replaced by $\mathbf{1}[c \text{ accepted}]\, w_c$, $w_c \in [0, 1]$, and the pairwise margin is its two-point, victim-only version. $w_c \geq 0$, and the pairwise margin is its two-point, victim-only version. The two signals estimate different quantities. A_c is a local sensitivity: it holds the membership and all other read weights fixed. Recovery measures the hard model's discrete counterfactual, the loss change when c is kept and another chunk is evicted instead, which is the relevant quantity at $\lambda = 1$. As $\tau \to 0$, $\omega_c \to \{0, 1\}$ and the soft gradient fades for chunks that are already evicted, exactly where recovery still measures value, and the anneal hands credit assignment over to recovery.

3.3 TRAINING THE READER AND EXECUTING THE POLICY

For retrieval over prose, we first train next-character prediction and then introduce retrieval tasks of increasing difficulty. Language pretraining develops the reader before it must learn task-specific query selection. Soft retention develops the policy before hard eviction removes its direct gradient paths. Recovery supplies additional credit under hard retention. These stages address different learning problems; the multi-key variable tracking (MKVT) experiments in §4.3 examine the contribution of the language phase. In general, we keep the soft phase on but recovery off for language pretraining, and turn both on for task-specific post-training; the MKVT suite also tests recovery during language training (Table 14). We hypothesize that the scorer largely will be data-starved after pretraining, as basic language modeling does not stress long-context retrieval circuits, requiring significant long-context data to warm up during post-training.

Training need not execute the full reader sequentially for every eviction. Because the scorer uses features independent of reader activations, a sequential membership sweep by the scorer can first record the retained sets. The recorded slates can then be scored with gradients, ending up with essentially an eccentric attention matrix, and the reader can process the sequence in parallel subject to those sets. This separates sequential memory bookkeeping from the larger attention computation, which is fully parallelizable. The sweep still has a sequential dependency, recovery adds task-dependent overhead, and the non-conformity of computations means we cannot use all efficient kernels modern transformers have access to; thus, in practice, during 400M-parameter pretraining on four H100 GPUs each, SSA trains $3.6\times$ slower per update than a Transformer in the soft phase and $5.9\times$ in the hard phase without recovery. However, we believe more optimization headroom remains.

4 EXPERIMENTS

We evaluate memory selection, reasoning, and optimization separately. Long-context cued copying tests whether relevant content can survive beyond a recent window. Multi-hop induction tests the reader's ability to compose information stored in its state, and emphasizes the importance of quadratic attention over the state through its results. MKVT (Multi-Key Variable Tracking) combines prose with query-dependent retrieval and exposes the interaction between pretraining and retention. We compare SSA with full-attention Transformers, Gated DeltaNets (GDN), HOLA, laLTE, and other fixed eviction policies where available. HOLA is a recent preprint(Cui, 2026); we include it as the most recent and competitive design we were able to find, and scale it (§4.5).

The experiments use distinct task protocols, detailed in Appendix B. We report their metrics separately: greedy generation exact match for cued copying, teacher-forced whole-sequence exact match for multi-hop induction, and teacher-forced whole-answer exact match for MKVT. More details are in the appendix for brevity. A backbone-matched Transformer shares SSA's reader dimensions but omits its scorer. In comparison, the larger transformer configuration matches the total parameter count of the SSA, essentially giving the transformer a larger language backbone. Both of these are kept for fairness to the transformer baseline. Unless noted, SSA shares chunk membership across layers and heads. In the 3M suite it stores keys and values in FP32 and retains $M = 14$ chunks of $C = 8$ tokens with $w = 1$ ($R = 128$); in the 20M suite it retains $M = 30$ ($R = 256$) and the 400M runs $M = 34$ chunks of 16 tokens ($R = 576$). Layer- and head-wise membership, BF16 storage with 30 chunks, and hybrids with GDN layers are compared in Tables 8 and 9. Crucially, we make sure to keep the parameter size and memory size (in bits) of SSA as close to competitors as possible while keeping both less than or equal to theirs, thus ensuring a comparison where SSA cannot be structurally favored (in the case of the backbone matched transformer, the language backbone is less than or equal to).

Our SSM of choice in testing is the GDN (Yang et al., 2025), due to the prevalence of GDN and its variants in deployed models (Qwen Team, 2025; Kimi Team, 2025; Qwen Team, 2026; Qiu et al., 2026). We are aware of GDN2 (Hatamizadeh et al., 2026), but chose to compare with GDN due to it being the industry standard for now still. We also compare to HOLA (Cui, 2026) as an architecture that aims to patch this weakness of SSMs rather than approaching it from an angle of fixing fixed-eviction-policy transformers, and to laLTE (He & Garner, 2025), which learns token eviction in a hybrid linear-and-sparse-attention model.

As expected from its lineage, SSA's learning errors and optimization problems closely follow the Transformer's (§4, Apps. B and E); learned retention adds a few failure points of its own, which our training recipe addresses. Learning-rate selection, seed counts, and other details are in the appendix.

4.1 Retaining information across long contexts

In cued copying, the model must reproduce a marked payload embedded in a long sequence. The input distinguishes relevant content before the final answer is requested. SSA's 128-token readable state is small relative to the 3072-token context, so success requires preserving information beyond the recent window.

SSA reaches the accuracy of the full-attention comparison (98.70% vs. 100%) while reading a state $24\times$ smaller than the context (Table 3 in App. B). The fixed eviction rules perform substantially worse (0.00% for drop-oldest, 2.86% for random eviction) with the same reader and capacity. The contrast shows that the learned policy preserves task-relevant content that a recent window or random subset usually loses. GDN, HOLA, and laLTE also perform well, demonstrating that this task can be solved with either compressed or selectively retained memory. We believe this is especially the case here as the content between the key and the asking of the key is noise and does not need to be tracked, and combined with answer only loss, a recurrent state can keep the key exactly. Thus, it establishes the feasibility of bounded retention, rather than a general advantage over recurrent models or vice versa.

4.2 Preserving multi-hop reasoning

The k-hop induction task composes previous-occurrence lookups in a random string (Sanford et al., 2024). We sample the hop count from zero through sixteen and predict the target sequence as a suffix. This experiment uses the 20M model family, as the 3M family was found to be insufficiently large (Table 4). The complete sequence fits SSA's 256-token readable state, so it is a control for reasoning with the bounded quadratic attention reader, not a test of retention under eviction pressure.

Table 1: Multi-hop induction: whole-sequence exact match (%). Fresh data resamples the training set every epoch. Details in App. B.

Model	Fixed data	Fresh data+dropout, mean [seed range]
Transformer (same backbone)	94.35	99.75 [99.70–99.80]
Transformer (larger)	82.75	99.55 [99.40–99.70]
Gated DeltaNet	0.10	34.40 [24.00–44.80]
HOLA	6.50	73.40 [70.50–76.30]
SSA	70.05	99.65 [99.40–99.90]

SSA approaches both Transformer controls (Table 1), and we believe the gap is mainly due to a bit worse optimization dynamics, as their read mechanisms are very close. GDN and HOLA remain substantially less accurate on this task. Their core inference states are approximately the same size as SSA's (§4.4), with SSA being strictly smaller, suggesting that the representation and use of memory, in addition to its size, affect reasoning performance heavily in certain tasks, consistent with the limits of fixed-size recurrent states on tasks that require exact access to earlier context (Jelassi et al., 2024). laLTE was not run at the 20M scale; at the 3M scale it slightly trails SSA on this task (Table 4), but does well on others.

4.3 Learning query-dependent retrieval in prose

For MKVT, we use a character-level adaptation of RULER's multi-key needle-in-a-haystack task (Hsieh et al., 2024), which we call MKVT, with key–value statements embedded in TinyStories prose (Eldan & Li, 2023). Distractor bindings require the model to distinguish candidate values using the query. We evaluate four tasks at a 1024-character context, varying the number of bindings and eligible query keys, with three seeds per task (App. E). **Seed details go in the appendix** We count a case as solved when fixed-target whole-answer exact match reaches 99%.

Retrieval learning can fail without memory loss. Across 24 Transformer recipes, failed checkpoints often produced a candidate value without responding to the query, even though the Transformer retains the full context; this matches the delayed emergence of attention operations in multi-step tasks (Hoffmann et al., 2024). Extending the language phase solves all development cases (Table 11); details and controls are in App. E.

SSA separates successful reading from successful retention. Applying extended language pretraining to SSA produces early retrieval learning in most cases. In the measured 48-run suite, 44 runs reach 90% first-stage answer exact match within five retrieval epochs. At the end of training, 40/48 runs reach $\geq 99\%$ fixed-target whole-answer exact and 46/48 reach $\geq 97.5\%$ (Table 14); with the same standard attention and unpaired queries, SSA solves 11/12 cases with plain extended pretraining versus 10/12 for the Transformer, and 9/12 versus 12/12 with randomized names.

To characterize the remaining errors, we measure whether any chunk containing part of the queried statement is directly readable at the query. The reader answers accurately when relevant source information is accessible; remaining errors are concentrated where all statement chunks have been discarded with fresh data. This associates the residual gap with retention.

One crucial point is the use of information carried indirectly through other contextualized chunks, and how it shows up in our data. We propose the notion of the n-hop accuracy metric, where 0-hop accuracy implies the necessary source chunk was in memory at the time of output, 1-hop implies one of the chunks in memory had the necessary source chunk in memory when they were processed, and so on and so forth. This is the stated "lossy information highway" SSA has access to. For example, 2.9% and 1.3% of examples have one and two hop chunks, and SSA answers 63.1% and 46.8% of them exactly (App. E).

Task learning and language retention are distinct. Expectedly, improved retrieval and task performance does not guarantee preserved language modeling, and we observe a slight drop in language modeling performance. For the completed Transformer arms with extended pretraining on unmodified stories, mean held-out prose loss increases from about 0.370 nats per character at the end of language training to 0.500–0.512 after retrieval training. We therefore evaluate language loss separately from retrieval accuracy.

4.4 Memory and computational cost

For a standard multi-head Transformer with the same reader dimensions, cached keys and values require $2BLdTb$ bytes at sequence length T, where B is the inference batch size and b the bytes per element. SSA replaces T with the fixed readable capacity R. This bounds persistent reader state independently of sequence length.

For the 20M family in FP32, SSA's memory state is a constant 6.00 MiB, slightly below the 6.11 MiB of GDN's and HOLA's recurrent matrices and convolution history, whereas a Transformer with SSA's reader dimensions needs 24 MiB of key–value cache at 1024 tokens and 240 MiB at 10,240. At BF16, the same bytes hold roughly twice as many tokens for SSA and the transformer. We measure memory in bytes as a neutral, downstream metric.

SSA therefore combines a state budget comparable to the recurrent baselines with an attention read over retained representations. Training has additional costs: differentiable warmup retains extra representations, parallel reads materialize activations across the example, and recovery repeats computations. An exploratory recovery-free timing study finds that SSA's overhead is large at small model sizes and is amortized at larger sizes and longer contexts (10,240 tokens, Appendix D).

During 400M pretraining on four H100 GPUs (global batch 524,288 tokens, 2K context), a Transformer update takes 2.5 s, while SSA takes 9.1 s in the soft phase and 14.9 s in the hard phase, without recovery.

4.5 Scaling Experiments

We scale to the 400M scale. We pretrain five models on 15B tokens from SlimPajama: Transformers at 2K and 8K context, GDN, HOLA, and SSA, all with 24 layers, width 1024, and about 400M parameters. Table 2 reports held-out SlimPajama negative log-likelihood. We pretrain at the 400M

scale as a relatively cheap but still larger scale experiment, where we get into "Small Language Model" territory (Liu et al., 2024; Lu et al., 2024). SSA is 0.03 to 0.07 nats behind the baselines at 2K. At 8K, both models pretrained at 2K degrade: the 2K Transformer from 2.615 to 5.039 nats and SSA from 2.656 to 4.334, while GDN, HOLA and the 8K Transformer improve slightly with the longer context (2.549–2.558). SSA's and the transformer's degradation likely come from positions: retained chunks keep their original rotary positions, so at 8K the reader sees query–key distances up to four times longer than any seen in pretraining.

Table 2: 400M models after pretraining on 15B SlimPajama tokens (one seed): held-out SlimPajama negative log-likelihood in nats per token (about 1M reserved test tokens), evaluated at 2K and 8K context. Transformer (8K) is pretrained at 8K context and all others at 2K; SSA is pretrained without recovery.

Model	Parameters	NLL at 2K	NLL at 8K
Transformer (2K)	400.1M	2.615	5.039
Transformer (8K)	400.1M	2.623	2.557
Gated DeltaNet	399.9M	2.590	2.549
HOLA	399.8M	2.591	2.558
SSA	399.8M	2.656	4.334

Here R for the SSA was 576 for all contexts. Despite that, it tracks the Transformer at 2K context, and is better than it at 8K context (4.334 vs 5.039 nats; one seed). GDN's lead at 2K, matched by HOLA, which builds on the same delta-rule state, is in line with prior comparisons at this parameter count and token budget on SlimPajama (Yang et al., 2024).

5 Discussion and limitations

The experiments show that accurate contextual retrieval and multi-step computation need not require a growing cache. Three conditions matter: relevant information must fit within the state, the policy must learn to retain it, and the reader must learn to use the query. Failure on a task does not by itself distinguish these limitations.

The MKVT study illustrates the distinction, and the need for data scaling particularly clearly. Longer language pretraining improves retrieval learning even in full-attention models, where eviction cannot explain failure. In SSA, high conditional accuracy when source information is readable shifts attention to the retention policy, but more data helps it learn as well. We anticipate that the joint training of the scorer and the language backbone might require more data in general than only training a language backbone with a fixed eviction policy (i.e a normal transformer), and acknowledge this as a limitation in a data-constrained world. Jointly evaluating reader accuracy and source availability helps distinguish these bottlenecks and informs the transition to hard selection.

Capability experiments use small models and controlled tasks; larger models appear only in exploratory experiments. Further evaluation on open-ended language understanding, like Q&A, and tasks whose relevant information exceeds the memory budget remain untested.

6 Conclusion

We introduce State Space Attention, which preserves strong retrieval and reasoning performance on the studied tasks using a learned, bounded attention memory, and occupies a corner of literature we believe remains unoccupied. Its training procedure addresses two requirements: learning to use the query and learning to retain what it requests. The results support learned selective memory as a route to contextual reasoning with constant inference state, and establish the eviction policy as a new degree of freedom for transformers: the standard transformer is the fixed first-in-first-out case, and a learned policy can take its place.

AI USE STATEMENT

We used generative AI tools to refine wording and suggest edits, generate figures from descriptions, identify related work and references during the brainstorming phase, assist with code implementation based on theory, and review the proof concerning equivalence between the soft and hard phases. AI-generated code was reviewed by the authors, who take responsibility for the paper's final content and claims. All ideation was done by the authors.

REPRODUCIBILITY STATEMENT

We submit the repository and as many experimental details as feasible and full experimental manifests, along with seeds and all other relevant details to reproduce the content. We also provide a "reproduction" folder to make this easier to use. We observe seed sensitivity on some tasks; every result we report was reproduced in targeted reruns of the same settings, and we do not report results we could not reproduce.

REFERENCES

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers. In *Advances in Neural Information Processing Systems*, 2023.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International Conference on Learning Representations*, 2024. arXiv:2312.04927.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Wanyun Cui. A hippocampus for linear attention: An exact memory for what the recurrent state forgets. *arXiv preprint arXiv:2607.02303*, 2026.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. arXiv:1901.02860.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024. arXiv:2405.21060.

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023. URL `https://arxiv.org/abs/2305.07759`.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024. arXiv:2312.00752.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

Mutian He and Philip N. Garner. Alleviating forgetfulness of linear attention by hybrid sparse attention and contextualized learnable token eviction. *arXiv preprint arXiv:2510.20787*, 2025.

David T. Hoffmann, Simon Schrodi, Jelena Bratulić, Nadine Behrmann, Volker Fischer, and Thomas Brox. Eureka-moments in transformers: multi-step tasks reveal softmax induced optimization problems. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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*, 2024.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *International Conference on Machine Learning*, 2024. arXiv:2402.01032.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020. arXiv:2006.16236.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Selective attention improves transformer. In *International Conference on Learning Representations*, 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*, 2024. arXiv:2404.14469.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *International Conference on Machine Learning*, 2024.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2305.17118.

Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D. Lane, and Mengwei Xu. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*, 2024.

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

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 37396–37412. PMLR, 2024. URL https://proceedings.mlr.press/v235/nawrot24a.html.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Zihan Qiu, Zekun Wang, Xiao Li, et al. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency, September 2025. URL https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. arXiv:1911.05507.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. In *International Conference on Machine Learning*, 2024. arXiv:2402.09268.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 2024. arXiv:2104.09864.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. Not all memories are created equal: Learning to forget by expiring. In *International Conference on Machine Learning*, 2021.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, 2024. arXiv:2406.10774.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. arXiv:2309.17453.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2406.06484.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. arXiv:2412.06464.

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. IndexMem: Learned KV-cache eviction with latent memory for long-context LLM inference. In *International Conference on Machine Learning*, 2026. arXiv:2605.25475.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing 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 Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

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*, 2023. arXiv:2306.14048.

A TRAINING GRADIENTS AND MEMORY ACCOUNTING

The read-weight derivative. For one attention head, introduce a chunk bias $z_c = \log \omega_c$ into the logits of all keys in chunk c. Let $P_{q,c} = \sum_{j \in c} p_{qj}$. Differentiating the softmax gives

$$\frac{\partial p_{qk}}{\partial z_c} = p_{qk}\big(\mathbf{1}[k \in c] - P_{q,c}\big), \qquad \frac{\partial o_q}{\partial z_c} = \sum_{k \in c} p_{qk}(v_k - o_q). \tag{7}$$

Contracting with the upstream gradient gives equation 5; contributions add across heads and layers.

During hard training, a straight-through weight has forward value one and a backward path through the scorer's drop probability, for example $\widetilde{\omega} = 1 - p + \mathrm{sg}(p)$, which is the standard method of applying STE. Applying it inside a normalized attention read produces the centered gradient above. Applying it after the softmax without renormalization instead yields an uncentered contribution proportional to $g_q^\top v_k$. These are different estimators. Of course, neither supplies a direct read gradient for chunks absent from the hard forward pass.

Counterfactual credit. Recovery runs only in the hard phase. At each step, the $\lfloor B/8 \rfloor$ examples with the highest answer loss are buffered (eight for $B = 64$). Every eight steps and at the end of each epoch, the buffer is replayed with the current scorer and reader, dropout off. For a buffered example, let p^* be the answer position with the highest next-token loss and W the answer targets among the C positions starting at that failed prediction. For a membership record $\mathcal{M}$,

$$\ell(\mathcal{M}) = \frac{1}{|W|} \sum_{i \in W} -\log p_\theta(x_i \mid x_{<i}; \mathcal{M}). \tag{8}$$

Candidates $\mathcal{K}$ are the chunks with index below $\lfloor p^*/C \rfloor - w$ that are not resident at chunk step $\lfloor p^*/C \rfloor$, i.e. everything discarded before the failure. For $c \in \mathcal{K}$, $\mathcal{M}_c$ is the recorded membership with one substitution: at the event that evicted c, and at any later event whose recorded victim is the chunk substituted out, the most droppable present chunk under the recorded scores is evicted instead. Let $\ell_0 = \ell(\mathcal{M})$, $\ell_c = \ell(\mathcal{M}_c)$ and $P = \{c \in \mathcal{K} : \ell_c < 0.9\,\ell_0\}$. If $|\mathcal{K}| \geq 8$ and $|P| > \max(6, \lfloor 0.05|\mathcal{K}| \rfloor)$, the example is rejected, since a retention failure should have a specific cause. Otherwise, if $P \neq \emptyset$, $c^* = \arg\min_{c \in \mathcal{K}} \ell_c$ is accepted. For an accepted c^*, let H_{c^*} contain its recorded slate appearances, e^* its eviction event, and r^* the chunk evicted in its place. With $p^{(e)} = \mathrm{softmax}(-s^{(e)}/\tau_{\mathrm{credit}})$ and $\tau_{\mathrm{credit}} = 1$,

$$\mathcal{L}_{\mathrm{rec}}(c^*) = -\sum_{e \in H_{c^*}} \log(1 - p_{c^*}^{(e)}) + \mathrm{softplus}\left(\frac{s_{r^*}^{(e^*)} - s_{c^*}^{(e^*)}}{\tau_{\mathrm{credit}}}\right). \tag{9}$$

Each accepted example b has relative improvement $\rho_b = \mathrm{clip}\big((\ell_0 - \ell_{c_b^*})/\ell_0, 0, 1\big)$ and weight $w_b = \rho_b/\bar{\rho}$, with $\bar{\rho}$ the mean over examples accepted in the same replay. The replay adds $\frac{64}{N} \sum_b w_b \mathcal{L}_{\mathrm{rec}}(c_b^*)$ to the loss, where N is the number of training examples since the previous replay. QUEST-style candidate ranking reduces search cost but does not replace this check (Tang et al., 2024).

Persistent state. For ordinary multi-head attention, SSA's reader stores $2BLdR$ floating-point elements. The policy also stores bounded token identifiers and positions for the same retained chunks. In the approximately 20M configuration, the reader uses six layers of width 512, eight-token chunks, thirty long-range chunks, and a one-chunk window, giving $R = 256$. FP32 reader state for batch one is 6 MiB; explicit token identifiers and compact metadata add approximately 2.3 KiB. A sequential inference implementation can realize this bound; a parallel training implementation may store representations and membership records for the full training sequence, which is why training is not cheaper like inference.

B BENCHMARK PROTOCOLS AND ADDITIONAL COMPARISONS

Cued copying. The task embeds two four-character payloads in a 3072-token context. Experiments use 2000 training examples, a maximum of 200 epochs, batch size 64, and a curriculum

that increases eviction pressure. The bounded models use a four-layer, width-256 backbone; SSA uses fourteen retained chunks, one window chunk, and eight-token chunks, for a 128-token readable state. The Transformer comparison shares the reader backbone. Table 3 reports the mean and seed range of completed runs at the learning rate with the better seed mean from $\{10^{-4}, 3 \cdot 10^{-4}\}$; SSA uses shared membership with FP32 storage. laLTE runs use a small patch that allows training on the curriculum's short early stages (at most 64 tokens), which the released implementation for the paper rejects.

Table 3: Cued copying with a 3072-token context: greedy generation exact match (%). Details in App. B.

Model / retention policy	Mean exact	Seed range
Transformer (same backbone)	100.00	100.00–100.00
Gated DeltaNet	100.00	100.00–100.00
HOLA	100.00	100.00–100.00
laLTE	100.00	100.00–100.00
SSA	98.70	97.66–100.00
Drop oldest (SWA at matched memory)	0.00	0.00–0.00
Random eviction	2.86	2.34–3.12

Multi-hop induction. The source string has length 100 over an alphabet of four symbols, with adjacent symbols different. A hop finds the preceding occurrence of the current symbol and advances one position; a missing predecessor produces a null target. Hop counts are sampled uniformly from zero through sixteen. We follow the target recursion of Sanford et al. (2024), but place the complete target in an autoregressive answer suffix instead of predicting targets at their original input positions. The resulting sequence fits the 256-token state used in this experiment.

Training presents 8000 examples per epoch for 240 epochs, with batch size 64, gradient clipping at one, and constant learning rate 10^{-4} or $3 \cdot 10^{-4}$. The fixed-data condition reuses its training examples; the fresh-data condition resamples them each epoch while keeping validation fixed. SSA and the backbone Transformer have six layers, width 512, and feed-forward width 1376. The larger Transformer uses width 576 and feed-forward width 1396. GDN and HOLA retain width 512 and feed-forward width 1921 from their source recipes. The fresh-data intervention preserves each model's remaining training settings. Fresh-data runs use dropout 0.1 for SSA and the Transformers.

At the 3M scale, no model learns the task under the same fixed-data protocol (Table 4). However, we see that the large transformer ranks first, then SSA, then laLTE, then the smaller transformer. Notably, drop oldest is only slightly below SSA.

Table 4: Multi-hop induction at the 3M scale ($N = 100$, hop counts 0 to 16, fixed training set, no dropout): whole-sequence exact match (%), teacher-forced and greedy, mean [seed range] over three seeds at the learning rate from $\{10^{-4}, 3 \cdot 10^{-4}\}$ with the better teacher-forced mean. The Transformer rows have a single run per learning rate.

Model	Teacher-forced	Greedy
Transformer (same backbone)	20.90	17.19
Transformer (larger)	30.30	28.91
Gated DeltaNet	0.00 [0.00–0.00]	0.00 [0.00–0.00]
HOLA	0.73 [0.00–2.20]	0.52 [0.00–1.56]
laLTE	25.47 [22.40–30.60]	19.01 [17.19–22.66]
GDN3 + full attention	16.60 [12.80–23.90]	11.46 [6.25–18.75]
SSA (FP32, 14 chunks)	27.13 [23.10–32.00]	19.53 [16.41–24.22]
SSA (BF16, 30 chunks)	26.57 [22.20–33.50]	19.53 [16.41–25.00]
Drop oldest (FP32, 14 chunks)	26.10 [24.10–29.40]	21.09 [18.75–25.00]
Random eviction (FP32, 14 chunks)	10.40 [7.40–13.20]	5.73 [3.91–7.03]

Additional retention tasks. Associative recall requires retrieving the value of a queried key among 64 bindings; four bindings are eligible for querying. Copy join requires reproducing two

Table 5: Fresh-data multi-hop induction, both tested learning rates. Teacher-forced whole-sequence exact match (%), mean [range] of two completed seeds. These cells underlie learning-rate selection in Table 1.

Model	10^{-4}	$3 \cdot 10^{-4}$
Transformer (same backbone)	99.75 [99.70–99.80]	97.65 [96.00–99.30]
Transformer (larger)	99.55 [99.40–99.70]	99.25 [98.80–99.70]
Gated DeltaNet	24.05 [20.90–27.20]	34.40 [24.00–44.80]
HOLA	70.30 [65.20–75.40]	73.40 [70.50–76.30]
SSA	99.65 [99.40–99.90]	99.15 [98.90–99.40]

payloads in their specified order among noise, for more details please check the attached repository. Table 6 compares learned and fixed policies with the same bounded reader. These experiments expose task dependence: learning retention greatly improves cued copy, improves copy join over dropping the oldest chunk, and associative recall remains below the recurrent baselines' perfect accuracy in this suite.

Table 6: Additional retention comparisons: greedy generation exact match (%), mean of completed seeds (three, or two for copy join) at the better of two learning rates. Each task uses identical reader capacity across the three policies.

Task	Learned retention	Drop oldest	Random eviction
Associative recall	92.19	69.27	80.99
Cued copy	98.70	0.00	2.86
Copy join	100.00	36.46	94.53

Associative recall uses 32,000 training examples and 240 epochs; copy join uses 10,000 examples and 240 epochs for these three policies.

The 3M suite. Each task uses three seeds per learning rate for cued copy, associative recall, alias chains, k-hop and RULER variable tracking, two or three for copy join, two for MKVT, variable tracking and WikiText, one or two for multi-query associative recall (MQAR)(Arora et al., 2024), and one for WikiText variable tracking and the n-gram suffix task. Alias chains, MKVT, variable tracking and WikiText variable tracking use a single learning rate. Exact settings are all available in the attached repository.

At this scale, the full-attention Transformer with SSA's reader dimensions fails the same tasks: alias chains (39.58%), k-hop (19.53%), MQAR with 64 pairs (21.65%), MKVT $(4, 24)$ (0.78%), and variable tracking (0.00%), so these gaps reflect the small backbone rather than retention. The exceptions are MKVT $(8, 8)$, and one seed each of MKVT $(4, 16)$ and 32-pair MQAR, which the Transformer solves and SSA does not. Table 8 compares SSA's membership and precision variants at equal state size in bytes, and Table 9 hybrids with GDN layers. Layer- or head-wise membership, and BF16 storage with 30 chunks, reach 94–100% on MKVT $(8, 8)$ and 16-pair MQAR, where shared FP32 membership with 14 chunks reaches 49.61% and 72.23%.

C HIDDEN-STATE SCORER ABLATION

We compared the token-embedding scorer with scorers that read the reader's hidden states early on. A hidden-state scorer's inputs depend on memory contents, so parallel training needs fixed-point iteration over the membership mask, at two to three times the cost per step. We did not see any tangible performance improvement, so decided to retire this version of the architecture. Code for experiments with this version are also available in the repository.

Table 7: laLTE versus SSA (shared, FP32, 14 chunks) in the 3M suite: mean at the better learning rate. MQAR is teacher-forced answer-character accuracy, WikiText is character perplexity (lower is better), and all other rows are greedy generation exact (%). Task definitions for all 3M-suite tasks are in the attached repository.

Task	laLTE	SSA
Cued copy	100.00	98.70
Associative recall	100.00	92.19
Copy join	100.00	100.00
Alias chain (3 hops)	89.84	48.70
k-hop ($N = 100$)	19.01	19.53
MQAR, 8 pairs	99.77	99.18
MQAR, 16 pairs	99.20	72.23
MQAR, 32 pairs	90.79	30.44
MQAR, 64 pairs	97.98	24.22
MKVT $(4, 4)$	0.00	100.00
MKVT $(4, 8)$	0.78	99.61
MKVT $(8, 8)$	0.00	49.61
MKVT $(4, 16)$	1.56	2.34
RULER variable tracking	81.25	100.00
Variable tracking	3.91	8.20
WikiText variable tracking	17.19	1.56
n-gram suffix (256)	7.03	42.19
WikiText (character PPL)	2.57	2.74

Table 8: SSA variants in the 3M suite: chunk membership shared across layers and heads (the SSA column of Table 7), independent per layer, or independent per layer and head, with FP32 keys and values and 14 retained chunks or BF16 and 30 chunks. Mean at the better learning rate; metrics as in Table 7. †One or more runs in the cell were stopped by the recovery liveness guard (no accepted candidate in 25 hard-phase epochs resulting in early termination) and are excluded. –: no completed run.

	FP32, 14 chunks			BF16, 30 chunks		
Task	Shared	Layer	Head	Shared	Layer	Head
Cued copy	98.70	99.22	99.22	97.40	98.18	100.00
Associative recall	92.19	95.05	–	99.22	99.48	–
Copy join	100.00	100.00	100.00	100.00	100.00	100.00
Alias chain (3 hops)	48.70	44.53	50.52	71.09	38.28	30.47
k-hop ($N = 100$)	19.53	22.14	21.88	19.53	19.27	19.01
MQAR, 8 pairs	99.18	93.66	99.57	84.12	99.10	99.35
MQAR, 16 pairs	72.23	96.77	93.84	99.79	99.69	99.29†
MQAR, 32 pairs	30.44	30.71	30.33	30.74†	41.13	65.37
MQAR, 64 pairs	24.22	24.18	23.91	20.30	14.16†	19.88†
MKVT $(4, 4)$	100.00	100.00	99.61	99.61	99.61	99.61
MKVT $(4, 8)$	99.61	99.22	99.61	100.00	99.61	99.61
MKVT $(8, 8)$	49.61	99.61	99.22	94.92	100.00	98.44
MKVT $(4, 16)$	2.34	44.53	44.14	49.61	58.59	80.08
RULER variable tracking	100.00	100.00	100.00	100.00	100.00	99.74
Variable tracking	8.20	7.81	3.52	7.81	11.72	17.19
WikiText variable tracking	1.56	1.56	1.56	1.56	1.56	1.56
n-gram suffix (256)	42.19	42.97	73.44	0.00	0.00	0.00
WikiText (character PPL)	2.74	2.73	2.76	2.65	2.66	2.68

D EXPLORATORY TRAINING-COST MEASUREMENTS

A recovery-free timing study measures a complete training step, including the SSA scorer sweep, attention forward and backward passes, and AdamW. It uses FP32, batch size 16, activation checkpointing, and a 144-token SSA read capacity. Table 10 reports the longest measured context.

Table 9: Hybrids in the 3M suite: three GDN layers followed by one SSA layer, with one membership set for the SSA layer or one per head, and the same stack with a full-attention fourth layer (growing memory). Mean at the better learning rate; metrics and markers as in Table 8.

	SSA, FP32, 14 chunks		SSA, BF16, 30 chunks		Full
Task	Layer	Head	Layer	Head	attention
Cued copy	100.00	100.00	100.00	100.00	100.00
Associative recall	100.00	100.00	100.00	100.00	100.00
Copy join	100.00	100.00	100.00	100.00	100.00
Alias chain (3 hops)	94.01	90.89	85.16	72.14	88.54
k-hop ($N = 100$)	13.54	16.41	13.28	14.32	11.46
MQAR, 8 pairs	99.95	99.94	99.91	99.98	99.98
MQAR, 16 pairs	99.88	99.94	99.94	100.00†	99.82
MQAR, 32 pairs	99.79	99.81†	99.80	99.96	99.77
MQAR, 64 pairs	73.71	59.33	99.93	99.92†	99.94
MKVT $(4, 4)$	61.33	62.50	99.22	99.22	48.44
MKVT $(4, 8)$	52.34	53.91	99.61	99.61	48.44
MKVT $(8, 8)$	57.03	52.73	55.08	55.47	48.83
MKVT $(4, 16)$	5.47	6.25	73.05	66.80	1.56
RULER variable tracking	100.00	100.00	100.00	100.00	100.00
Variable tracking	47.66	48.05	96.88†	3.91†	48.05
WikiText variable tracking	2.34	1.56	96.09	1.56	21.09
n-gram suffix (256)	92.97	80.47	50.00	61.72	97.66
WikiText (character PPL)	2.72	2.72	2.69	2.68	2.52

Table 10: Exploratory training cost at 10,240 tokens. Time is milliseconds per 1000 tokens; memory is measured peak training allocation in GiB. Recovery is disabled.

	Time		Peak training memory	
Model scale	SSA	Transformer	SSA	Transformer
3M	11.80	3.49	7.80	5.13
25M	24.20	15.00	11.96	11.66
85M	41.08	36.99	20.13	19.79
314M	90.11	97.21	38.43	38.02
1B	239.55	255.31	75.53	74.99

The measurements illustrate two distinct effects. The scorer adds a substantial overhead at small scale, whereas the bounded attention computation becomes relatively less costly at larger scale. Peak training memory can nevertheless be similar because the training step stores activations and optimizer state. Neither observation changes the constant persistent-state property of sequential inference, and neither includes the additional cost of counterfactual recovery. SSA shows signs of a wall-clock advantage at larger scales at this context length, which could help alleviate effects of the soft-training and recovery, but this is yet to be stress tested; at the 2K context of our 400M pretraining, SSA remains slower (§4.4).

E Multi-key variable tracking

E.1 Data, objectives, and training schedules

MKVT inserts statements such as `VAR abc = 41` into TinyStories prose and queries them using `?abc=`. Keys are three lowercase letters drawn from the full 26^3 pool; keys beginning with `a` are eligible for querying. Values are distinct two-digit strings within an example. We use four configurations $(K, B) \in \{(4, 16), (4, 24), (8, 8), (16, 16)\}$, where K is the number of eligible keys and B the total number of bindings. Each recipe is run on seeds 0, 1 and 2 of each configuration, giving twelve task–seed cases. The 1024-character target includes its statements and answer suffix. A queried statement must occur more than seventeen pre-insertion document characters before the

Table 11: Completed Transformer MKVT experiments. Solved cases have teacher-forced whole-answer exact match $\geq 99\%$ on 1024 fixed-target validation examples. Name randomization applies only during language training. The auxiliary source head uses task annotations and is reported as a separate supervised control.

Language characters	Retrieval attention	Paired queries	Randomized names	Source head	Solved cases
73M	NormSoftmax	Yes	No	No	6/12
73M	NormSoftmax	Yes	Yes	No	3/12
73M	NormSoftmax	Yes	No	Yes	12/12
998M	NormSoftmax	Yes	No	No	12/12
998M	Standard	Yes	No	No	12/12
998M	Standard	No	No	No	10/12
998M	NormSoftmax	Yes	Yes	No	12/12
998M	Standard	Yes	Yes	No	12/12
998M	Standard	No	Yes	No	12/12

document end; other eligible statements need not satisfy this constraint. Thus statement placement is not fully independent of query eligibility. This was done to force the need for retention.

Models use a 97-character vocabulary present in earlier experiments. Training and validation stories are disjoint, but adjacent procedural seeds generate overlapping example streams. The fixed final-task validation set has 1024 examples per run. Whole-answer exact match requires both answer characters to be correct under teacher forcing; the separate generation check uses greedy decoding on 128 examples. The training objective adds separately averaged answer and non-answer cross-entropies with unit weights, preventing the short answer from being diluted by the much longer prose context.

Table 12: MKVT language and retrieval budgets. The longer language phase adds training before the same retrieval budget.

	Short language phase	Extended language phase
Epochs at each of lengths 256, 512, 1024	5	68
Language examples per epoch	8192	8192
Language character presentations	73,400,320	998,244,352
Retrieval epochs	135	135
Total epochs	150	339

Retrieval proceeds through $(L, K, B) = (128, 2, 2), (256, 4, 4), (512, \min(K, 8), 8)$ and then the final task. The $(4, 24)$ task adds an intermediate $(1024, 4, 16)$ stage. The first retrieval stage is capped at 28 epochs, or 21 for $(4, 24)$, and at least 51 epochs are reserved for the final stage. Earlier stage transitions allocate the saved epochs to the final task; total retrieval budgets are equal even though allocations across stages can differ.

The Transformer has 19,028,992 parameters and SSA has 22,499,712, so they are backbone matched. Transformer runs use batch size 64, dropout 0.1, AdamW, gradient clipping at one, and a cosine learning rate from $3 \cdot 10^{-4}$ to 10^{-5}. The extended language phase adds 189 epochs at the initial learning rate so that the retrieval schedule is preserved. Paired batches contain 32 contexts with two eligible queries each; unpaired batches contain 64 independent contexts. All evaluations are unpaired. NormSoftmax, when used, is restricted to retrieval training. Randomized-name pretraining replaces repeated story names consistently within each document; retrieval prose is unmodified.

SSA uses unpaired queries, dropout 0.1, constant learning rate 10^{-4}, and recovery during retrieval. The extended schedule places 136 soft-retention epochs within language training, followed by hard retention; retrieval begins at epoch 205. The four arms cross name randomization with recovery during the language phase. The longer hard-language interval also reduces the sampling temperature at retrieval onset relative to the shorter schedule.

E.2 RETRIEVAL ONSET AND DIAGNOSTIC CONTROLS

Retrieval learning can fail without memory loss. An ablated study of 24 Transformer recipes found substantial seed sensitivity at small scale: the best shared recipe solved six of twelve cases, and no recipe solved all three seeds of any task. Failed checkpoints often produced a candidate value without responding to which key was queried. In a paired diagnostic, one failed checkpoint answered only eight of 64 queries correctly across eight shared passages, whereas a successful checkpoint answered all 64. The failed model returned the same answer for different queries about each passage. Since the Transformer retains the full context, this failure cannot be explained by eviction. It indicates a failure to learn query-dependent selection, consistent with the delayed emergence of attention operations observed in other multi-step tasks (Hoffmann et al., 2024).

Language pretraining improves retrieval learning. We extend the language phase while preserving the retrieval budget and its learning-rate schedule. A second intervention randomizes repeated proper names consistently within each language-training story, intended to teach the model to respond to these randomizations. It changes the language examples, without using binding locations or other task specific annotations. The retrieval syntax and evaluation tasks remain unchanged.

Extended pretraining solves all development cases with paired queries under either standard attention or NormSoftmax (Hoffmann et al., 2024) (which was tried as a method to solve this problem) (Table 11). Adding randomized names also solves all cases with standard attention and unpaired queries. Most of the improvement comes from the extended language budget; randomized names hurt at the short budget (3/12 vs 6/12) and add two cases with standard attention and unpaired queries at the extended budget (12/12 vs 10/12).

Table 13 reports first-stage retrieval onset for three extended-pretraining variants with randomized names, each with twelve cases. The short-pretraining name control changes names without increasing the language budget and solves $(4, 16)$ seed 1 and $(8, 8)$ seeds 0 and 2.

Table 13: Retrieval onset with extended language pretraining and randomized names. Onset means first-stage teacher-forced answer exact match $\geq 90\%$. Final success uses the separate fixed-target metric.

Retrieval configuration	By epoch 2	Before stage exit	Final solved
NormSoftmax, paired	11/12	12/12	12/12
Standard attention, paired	11/12	12/12	12/12
Standard attention, unpaired	10/12	12/12	12/12
All three	32/36	36/36	36/36

E.3 SSA RESULTS AND SOURCE AVAILABILITY

Table 14 reports the completed extended-pretraining suite at epoch 339.

Table 14: SSA MKVT at the end of training (epoch 339). Each arm contains twelve task–seed cases. Exact is fixed-final-task teacher-forced whole-answer exact on 1024 validation examples.

Language intervention	$\geq 99\%$	$\geq 97.5\%$	Lowest exact (%)
Extended pretraining	11/12	12/12	98.0
+ Randomized names	9/12	12/12	97.6
+ Recovery in language	9/12	11/12	83.9
+ Both	11/12	11/12	86.1
All arms	40/48	46/48	83.9

Forty-four runs reach 90% hard first-stage answer exact within one to five retrieval epochs.

Source availability is measured using recorded chunk memberships. A statement is directly readable when any overlapping chunk remains in memory or the local window at the query. A longer path through retained contextualized chunks denotes possible indirect access, not demonstrated causal use.

A statement spans two or three chunks. Retaining the chunk with the value can be enough to answer even when most statement characters are absent. A character-fraction metric that excludes the local window can therefore understate usable retention. The chunk availability diagnostic captures this distinction. In the earlier sample, cases with one or two intermediate chunks comprise 2.9% and 1.3% of examples, with conditional answer exact of 63.1% and 46.8%. Most successful retrieval is associated with direct source availability.