RAM-Net: Linear-Time Sequence Modeling with Sparsely Addressable State

Kaicheng Xiao[†] **Haotian Li**[†] **Liran Dong** **Guoliang Xing**
The Chinese University of Hong Kong
{xk023, lh023, dl123, glxing}@ie.cuhk.edu.hk

https://github.com/MuonCat/RAM-Net

Abstract

Linear attention offers an efficient alternative to full attention with a fixed-size recurrent state. However, this state is shared by all tokens, so information from distinct tokens becomes superposed within it and produces inter-token interference that degrades long-range fine-grained recall. To address this issue, we propose RAM-Net, which replaces dense access to a shared state with sparse address-based access. RAM-Net organizes the recurrent state as a fixed-size array of independent slots and uses an Address Decoder that maps each key or query into a sparse address, selecting a small subset of slots to write to or read from at each step. This design directs tokens with non-overlapping addresses to disjoint slots, suppressing inter-token interference, while keeping per-step state access dependent only on the number of selected slots rather than the total state size. Empirically, RAM-Net outperforms strong recurrent baselines on fine-grained long-range retrieval and achieves competitive perplexity and commonsense reasoning. It does so while accessing fewer state elements per step than all baselines, e.g., 7.6× fewer than Mamba2.

1 Introduction

Transformers [45] have become the dominant architecture for sequence modeling, largely owing to the ability of full attention to capture long-range dependencies. This ability rests on two design choices. First, full attention stores every past token as a separate entry in a key-value (KV) cache, which keeps the information of each token separable. Second, it weights these entries for each query through a softmax over query-key similarities, which tends to concentrate each read on the few entries that match the query. However, such high-fidelity retrieval comes at a cost, as the KV cache grows linearly with sequence length and the total computation grows quadratically, creating a fundamental bottleneck for long-context modeling [31].

Linear attention removes this growth by compressing the history into a fixed-size recurrent state. Early designs approximate the softmax similarity with kernel feature maps and accumulate each token into a single state matrix through an outer-product update [18, 8]. Subsequent variants refine this update with data-dependent decay, gating, or delta-rule corrections [50, 32, 36, 49, 11]. Nevertheless, these designs give up both choices of full attention. Tokens are superposed in one shared matrix rather than stored separately, and each step reads the entire state through linear aggregation rather than selecting a few entries. Consequently, contributions from different tokens become entangled, and this inter-token interference degrades fine-grained long-range recall [2, 38]. Moreover, since every step accesses the entire state, enlarging the state to hold more information raises the per-step cost proportionally.

[†]Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

However, we observe that a fixed-size state does not by itself require giving up separable storage or selective access. This observation points to a third attention paradigm beyond full attention and linear attention. In this paradigm, the state remains bounded but is divided into many separate units, and each step accesses only a few of them. We realize this paradigm as the Random Access Memory Network (RAM-Net), which organizes the recurrent state as a fixed-size array of independent storage units called *slots*. At each step, the key produces a *write address* that selects a few slots to write into, and the query produces a *read address* that selects a few slots to read from. All other slots remain unchanged. This address-based access resembles random access memory in computer systems and gives the architecture its name. Since each address selects only a few slots, such sparse addressing brings two properties that answer the two problems of linear attention. (i) **Confined interference.** Tokens with non-overlapping addresses are written to disjoint slots, so interference arises only at the slots they jointly address. (ii) **Decoupled state size.** Each step accesses only the selected slots, so the total state can grow without increasing the per-step state access.

We evaluate RAM-Net at the 340M and 1.3B scales against Transformer++ and strong recurrent baselines. On long-context retrieval, RAM-Net achieves the highest S-NIAH average at both scales. On language modeling, it achieves the lowest WikiText perplexity at 340M and remains competitive at 1.3B on both perplexity and commonsense reasoning. Notably, since sparse addressing accesses only a small subset of the state at each step, RAM-Net obtains these results with less per-step state access than every baseline, e.g., at least $7.6\times$ less than Mamba2 (Table 1). This reduction also translates into practical speed, as RAM-Net Top-8 achieves $2.0\times$ the decoding throughput of the fastest baseline (Fig. 5).

2 Background

Attention enables a sequence model to associate past information with the current output through a state $\mathbf{S}_t$ that stores historical context. At each step, two operators act on this state: a *write* operator $\mathcal{W}$ that integrates the new input into $\mathbf{S}_t$, and a *read* operator $\mathcal{R}$ that retrieves from $\mathbf{S}_t$ to produce the output [36, 3]. Concretely, given an input row vector $\mathbf{x}_t \in \mathbb{R}^{1\times d_m}$, the model computes a query $\mathbf{q}_t \in \mathbb{R}^{1\times d_k}$, key $\mathbf{k}_t \in \mathbb{R}^{1\times d_k}$ and value $\mathbf{v}_t \in \mathbb{R}^{1\times d_v}$ via learnable projections. The key and value drive $\mathcal{W}$ to update the state, while the query drives $\mathcal{R}$ to retrieve from it:

$$\mathbf{S}_t = \mathcal{W}(\mathbf{S}_{t-1}, \mathbf{k}_t, \mathbf{v}_t), \quad \mathbf{o}_t = \mathcal{R}(\mathbf{S}_t, \mathbf{q}_t). \tag{1}$$

Within this formulation, attention architectures differ in two design aspects. (i) **State granularity**: whether $\mathbf{S}_t$ is divided into separate units that each store part of the history or maintained as a single undivided block. (ii) **Access pattern**: how much of $\mathbf{S}_t$ the operators $\mathcal{W}$ and $\mathcal{R}$ access at each step, and how the accessed contents are weighted. We next revisit full attention and linear attention along these two aspects.

2.1 Full Attention

Full attention [45] retains the entire history of keys and values (see Fig. 1a). At step t, its state $\mathbf{S}_t = (\mathbf{K}_t, \mathbf{V}_t)$ stores all past keys and values, with $\mathbf{K}_t \in \mathbb{R}^{t\times d_k}$ and $\mathbf{V}_t \in \mathbb{R}^{t\times d_v}$, giving a state size of $t \cdot (d_k + d_v)$. The write operator simply appends the current key and value, and the read operator scores the query against every cached key with a softmax-normalized similarity, then aggregates the corresponding values:

$$\begin{aligned} \mathbf{K}_t &= \mathrm{Concat}(\mathbf{K}_{t-1}, \mathbf{k}_t), \quad & \mathbf{V}_t &= \mathrm{Concat}(\mathbf{V}_{t-1}, \mathbf{v}_t), \\ \mathbf{r}_t &= \mathrm{Softmax}(\mathbf{q}_t \mathbf{K}_t^\top), & \mathbf{o}_t &= \mathbf{r}_t \mathbf{V}_t. \end{aligned} \tag{2}$$

Along the two aspects above, full attention adopts (i) a **per-token-entry** state granularity, where each token's $(\mathbf{k}_s, \mathbf{v}_s)$ occupies a dedicated entry in $\mathbf{S}_t$, and (ii) a **softmax-based dense** access pattern, where each query is scored against every cached key via softmax and aggregates the corresponding values. Full attention achieves high retrieval fidelity through the joint effect of the two choices: per-token entries keep each token's information separable, and softmax concentrates on the most similar entries. However, the state grows linearly with sequence length and incurs substantial memory and compute cost in long-context inference.

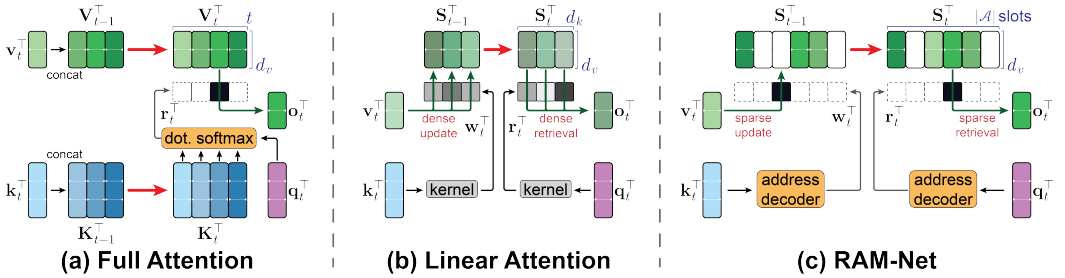


Figure 1: Comparison of attention architectures. (a) Full attention maintains a KV cache and retrieves through competitive softmax over all cached keys, supporting high retrieval fidelity at the cost of memory that grows with sequence length t. (b) Linear attention compresses history into a fixed shared recurrent state and retrieves it through dense linear aggregation, thereby causing inter-token interference at retrieval. (c) RAM-Net keeps a fixed slot array and uses sparse address-based access: an Address Decoder maps $\mathbf{k}_t$ and $\mathbf{q}_t$ into sparse write and read addresses $\mathbf{w}_t$ and $\mathbf{r}_t$, so only selected slots are updated or retrieved.

2.2 Linear Attention

Linear attention compresses the state to a fixed size $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$, as shown in Fig. 1(b), giving a state size of $d_k \cdot d_v$. Vanilla linear attention [18] achieves this by replacing the $\exp(\mathbf{q}_t \mathbf{k}_s^\top)$ in softmax with the inner product $\phi(\mathbf{q}_t)\phi(\mathbf{k}_s)^\top$ under a kernel feature map $\phi(\cdot)$. This approximation factorizes the softmax dependency between query and key, and allows the history to be accumulated into a single matrix. Concretely, the write applies an outer product $\mathbf{w}_t^\top \mathbf{v}_t$ with $\mathbf{w}_t = \phi(\mathbf{k}_t)$, and the read aggregates $\mathbf{S}_t$ via $\mathbf{r}_t = \phi(\mathbf{q}_t)$:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{w}_t^\top \mathbf{v}_t, \quad \mathbf{o}_t \propto \mathbf{r}_t \mathbf{S}_t = \sum\nolimits_{s \le t} \phi(\mathbf{q}_t)\phi(\mathbf{k}_s)^\top \mathbf{v}_s. \tag{3}$$

Modern variants generalize this update to $\mathbf{S}_t = \mathbf{\Lambda}_t \mathbf{S}_{t-1} + \mathbf{w}_t^\top \mathbf{v}_t$, where the transition matrix $\mathbf{\Lambda}_t$ introduces data-dependent decay, gating, or delta-rule corrections [11, 32, 50, 49], refining how past contributions are weighted across time while still accumulating them into a single shared matrix.

Examined along the two design aspects above, linear attention adopts (i) an **unpartitioned matrix** state granularity, where the outer-product update $\mathbf{w}_t^\top \mathbf{v}_t$ writes every token's value into the same $d_k \times d_v$ matrix with no per-token entries, and (ii) an **additive dense** access pattern, where each step reads from the entire matrix via $\mathbf{r}_t \mathbf{S}_t$, which expands into an additive aggregation $\sum_s \phi(\mathbf{q}_t)\phi(\mathbf{k}_s)^\top \mathbf{v}_s$ over all past values. Crucially, retrieval interference emerges from the joint effect of the two choices: the unpartitioned matrix superposes contributions from different tokens into the same storage, and the additive read sums all of them into the output with limited ability to suppress the irrelevant ones [36, 38].

Overall, full attention and linear attention occupy two contrasting positions in this two-aspect design space. Full attention stores tokens as separate entries and weights them through a softmax, which in practice concentrates on a few high-similarity entries [7]. As a result, it achieves high retrieval fidelity, but its state grows with sequence length. In contrast, linear attention bounds the state by superposing all tokens into a single matrix and reading it through additive aggregation. This design reduces cost but loses both the separability and the selective access of full attention. The contrast points to a third paradigm, which keeps the state bounded as in linear attention yet divides it into separable units and accesses only a few of them at each step. In principle, such a paradigm can retain the efficiency of linear attention while approaching the retrieval fidelity of full attention. The next section develops RAM-Net within this paradigm.

3 RAM-Net

3.1 Overview: Sparse Addressing on a Slot Array

The core idea of RAM-Net is twofold: it organizes the fixed-size state into many independent units; at each step, the current key and query each select a sparse subset of these units to write into and read from, respectively.

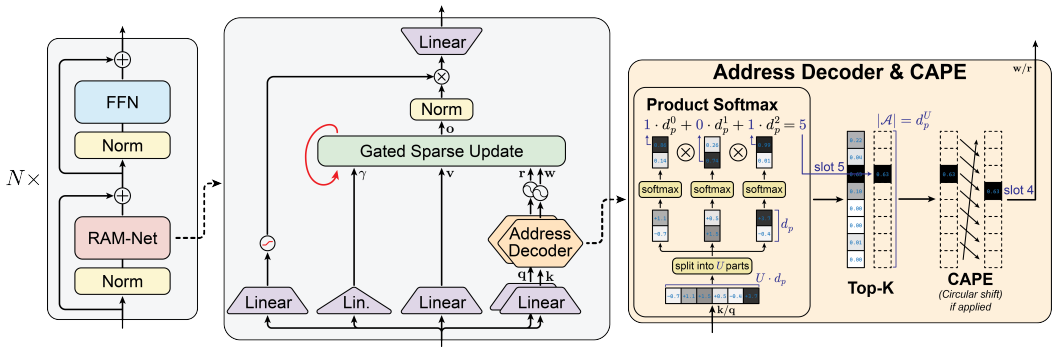


Figure 2: Overview of RAM-Net. The Address Decoder maps the key $\mathbf{k}_t$ and query $\mathbf{q}_t$ to sparse weights over the slots through Product Softmax and Top-K truncation. Cyclic Address Positional Embedding (CAPE) then optionally shifts their slot indices by token position, producing the write and read weights $\mathbf{w}_t$ and $\mathbf{r}_t$. Next, Gated Sparse Update writes $\mathbf{v}_t$ into the selected slots, and the read output $\mathbf{o}_t$ is normalized, gated, and projected. For visual clarity, the Address Decoder panel illustrates $K = 1$ and $|\mathcal{A}| = 8$ slots, with CAPE shifting the selected slot 5 to slot 4.

Along the two aspects in §2, RAM-Net makes the following choices. (i) For state granularity, RAM-Net organizes the state as an **addressable slot array**. The array contains $|\mathcal{A}|$ slots, whose integer addresses form the finite address space $\mathcal{A} = \{0, \ldots, |\mathcal{A}| - 1\}$. At step t, the slot at address a stores a content vector $\mathbf{s}_t[a] \in \mathbb{R}^{1\times d_v}$ that aggregates the values written to it so far (§3.4), and stacking all slots gives the state $\mathbf{S}_t \in \mathbb{R}^{|\mathcal{A}|\times d_v}$, as illustrated in Fig. 1(c). (ii) For the access pattern, RAM-Net adopts **softmax-based sparse** access. At each step, the key $\mathbf{k}_t$ produces a write weight vector $\mathbf{w}_t$ over $\mathcal{A}$, and the query $\mathbf{q}_t$ produces a read weight vector $\mathbf{r}_t$. Each vector has only $K \ll |\mathcal{A}|$ nonzero entries. The positions of these entries select the slots to write $\mathbf{v}_t$ into or read from, and their magnitudes $w_t[a]$ and $r_t[a]$ specify the write and read strengths at slot a. These weights come from a softmax-based mapping over $\mathcal{A}$ followed by Top-K truncation, as detailed in §3.2.

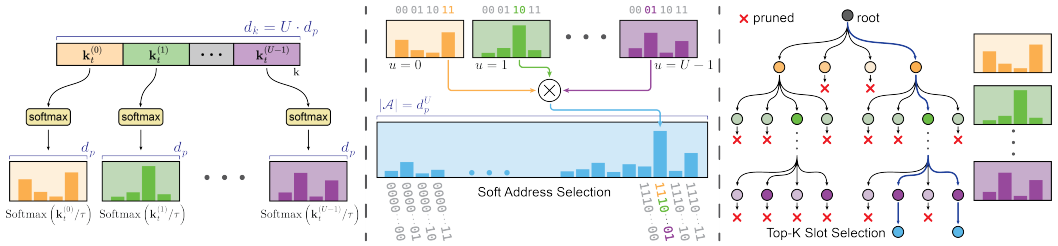
3.2 Address Decoder

This section presents the Address Decoder, which maps each key or query to sparse weights over the slots to write to or read from at the current step (Fig. 2). To obtain selective access as in full attention, a direct realization would project keys and queries to $|\mathcal{A}|$ dimensions, apply a softmax over all slots, and retain only the top-K weights. However, this construction faces two obstacles as $|\mathcal{A}|$ grows. First, the projections and the $|\mathcal{A}|$-dimensional softmax add parameters and computation that grow linearly with $|\mathcal{A}|$. Second, after Top-K truncation, only the K selected entries out of $|\mathcal{A}|$ receive a direct learning signal at each step, so the address selection becomes hard to optimize as $|\mathcal{A}|$ grows (Appendix B.1). To resolve both obstacles, the Address Decoder combines a *Product Softmax* construction with a Top-K truncation step, as illustrated in Fig. 3.

Product Softmax. Product Softmax $\rho_U(\cdot)$ builds the distribution over all $|\mathcal{A}|$ slots from U small softmax distributions. Specifically, as illustrated in Fig. 3(a), the key $\mathbf{k}_t$ (and similarly $\mathbf{q}_t$) is split into U sub-vectors $\mathbf{k}_t = [\mathbf{k}_t^{(0)}, \ldots, \mathbf{k}_t^{(U-1)}]$, each of dimension $d_p = d_k/U$. We set $|\mathcal{A}| = d_p^U$, so that each address a can be written as U **digits in radix** d_p, $a = \sum_{u=0}^{U-1} a^{(u)} d_p^u$ with $a^{(u)} \in \{0, \ldots, d_p - 1\}$. Each sub-vector $\mathbf{k}_t^{(u)}$ scores the d_p choices of digit $a^{(u)}$, and a softmax turns these scores into a digit distribution $\mathbf{p}_t^{(u)} = \mathrm{Softmax}(\mathbf{k}_t^{(u)}/\tau)$, where $\tau > 0$ is a temperature. As shown in Fig. 3(b), the probability $\rho_U(\mathbf{k}_t)[a]$ of each address a is the product of its digit probabilities, so the full distribution $\rho_U(\mathbf{k}_t)$ is the Kronecker product $\otimes$ of the digit distributions:

$$\rho_U(\mathbf{k}_t)[a] = \prod_{u=0}^{U-1} p_t^{(u)}[a^{(u)}], \qquad \rho_U(\mathbf{k}_t) = \mathbf{p}_t^{(U-1)} \otimes \cdots \otimes \mathbf{p}_t^{(0)}. \tag{4}$$

Top-K Truncation. To obtain K-sparse weights, we apply a Top-K truncation operator $\mathcal{T}_K(\cdot)$, which retains the K largest probabilities and sets the rest to zero. The complete address decoding is



(a) Sub-vector Softmax (b) Kronecker Combination (c) Top-K Prefix Tree Search

Figure 3: Address Decoder. (a) The key $\mathbf{k}_t$ or query $\mathbf{q}_t$ is split into U sub-vectors of size d_p, and each passes through its own softmax to give one digit distribution. (b) The Kronecker product of the digit distributions gives the address distribution over $|\mathcal{A}| = d_p^U$ slots. (c) A Top-K tree search keeps the K most probable partial addresses at each level, returning the exact Top-K slots without materializing the $|\mathcal{A}|$-dimensional distribution.

therefore defined as:

$$\mathcal{D}_{K,U}(\mathbf{k}_t) = \mathcal{T}_K\left(\rho_U(\mathbf{k}_t)\right). \tag{5}$$

In practice, we fuse Product Softmax and Top-K truncation, so that the full $|\mathcal{A}|$-dimensional distribution is never materialized. Expanding the Kronecker product factor by factor yields an implicit prefix tree of depth U, with complete addresses at the leaves and each prefix weighted by the product of digit probabilities along its path from the root (Fig. 3(c)). At each level, we extend each retained prefix with all d_p possible next digits and keep the K most probable extensions. This pruning is exact because suffix probabilities are independent of the prefix: for any completion of a discarded prefix, appending the same suffix to the K retained prefixes yields K addresses that are at least as probable. As each level examines only Kd_p candidates, the cost grows linearly in $U = \log_{d_p} |\mathcal{A}|$ rather than in $|\mathcal{A}|$ (see §4 for details).

The Address Decoder thus resolves both obstacles of a direct $|\mathcal{A}|$-dimensional softmax. For the cost obstacle, with a fixed radix d_p, the slot count $|\mathcal{A}| = d_p^U$ grows exponentially in the number of digits U, while the key and query dimension $d_k = Ud_p$ grows only linearly, and the Top-K prefix tree search avoids any $|\mathcal{A}|$-dimensional computation. For the optimization obstacle, each digit is shared by many slots, so the learning signal from a selected slot also reaches every slot that shares its digits. Consistently, our ablation shows that a larger U improves associative recall accuracy, with larger gains when $|\mathcal{A}|$ is large (Appendix B).

3.3 Cyclic Address Positional Embedding

The Address Decoder computes addresses from content alone, so identical content maps to the same slots regardless of its position. To inject positional information, we propose *Cyclic Address Positional Embedding (CAPE)*, which cyclically shifts the decoded addresses according to the token position, as illustrated in the CAPE block of Fig. 2. Specifically, CAPE moves each selected address a to $\pi_t(a) = (a - t) \bmod |\mathcal{A}|$ while keeping its weight, which changes the slots being accessed but not their contents. The final write and read weights are thus given by

$$w_t[\pi_t(a)] = \mathcal{D}_{K,U}(\mathbf{k}_t)[a], \qquad r_t[\pi_t(a)] = \mathcal{D}_{K,U}(\mathbf{q}_t)[a], \qquad a \in \mathcal{A}. \tag{6}$$

Since writes at step s are shifted by s and reads at step t by t, the overlap between $\mathbf{r}_t$ and an earlier $\mathbf{w}_s$ depends jointly on their content and the relative distance $t - s$. This is analogous to RoPE [37], which obtains relative positions by rotating queries and keys, whereas CAPE does so by shifting slot addresses. Since position-aware and content-only addressing capture different patterns, RAM-Net applies CAPE to some heads and No Positional Embedding (NoPE) to the others (see Appendix A for the configuration and Appendix C for access visualizations).

3.4 Gated Sparse Update

A state update must support both writing new information and forgetting outdated content. In gated linear-attention models, forgetting is implemented through a gate applied to the entire state at each step [11, 32, 50, 49]. Applying such a gate to the slot array would access all $|\mathcal{A}|$ slots per step. This

conflicts with sparse addressing, which modifies only the slots selected by $\mathbf{w}_t$ and leaves all other slots exactly unchanged (§3.1). We therefore propose *Gated Sparse Update* (GSU), which applies a forgetting gate only to the addressed slots, so that writing and forgetting both remain sparse.

GSU uses a *per-slot test-time training* (TTT) update [38], taking a gradient step on the squared-error objective $\ell_t(\mathbf{s}) = \frac{1}{2}\|\mathbf{s} - \mathbf{v}_t\|_2^2$ with an adaptive learning rate. Each slot a maintains a content vector $\mathbf{s}_t[a]$ and a scalar mass $m_t[a]$ representing its inertia against new writes. At each step, GSU first scales the previous mass by a retention factor $\lambda_t[a]$ and adds the current write weight $w_t[a]$ from Eq. (6):

$$\begin{aligned} \lambda_t[a] &= \sigma(\mathrm{logit}(1 - w_t[a]) - \gamma_t)\,, \\ m_t[a] &= \lambda_t[a]\, m_{t-1}[a] + w_t[a], \end{aligned} \tag{7}$$

where σ is the sigmoid function and $\mathrm{logit}(x) = \log \frac{x}{1-x}$. The retention factor uses $1 - w_t[a]$ as its baseline, so a slot discards more of its mass when written more strongly and keeps all of it when not written. A forgetting modulator γ_t adjusts this baseline in logit space. It is computed from $\mathbf{x}_t$ by a learned per-head projection, giving one scalar per token and head shared across slots (see Fig. 6 in Appendix A). Then, GSU uses the updated mass to compute the learning rate $\eta_t[a]$ and update the slot content:

$$\begin{aligned} \eta_t[a] &= \frac{w_t[a]}{m_t[a] + \epsilon}, \\ \mathbf{s}_t[a] &= \mathbf{s}_{t-1}[a] + \eta_t[a]\big(\mathbf{v}_t - \mathbf{s}_{t-1}[a]\big), \end{aligned} \tag{8}$$

with $m_0[a] = 1/|\mathcal{A}|$, $\mathbf{s}_0[a] = \mathbf{0}$, and a small constant $\epsilon > 0$ for numerical stability.

These updates give GSU three properties. (i) **Exact sparsity.** For an unaddressed slot, $w_t[a] = 0$ gives $\lambda_t[a] = 1$ by continuity and $\eta_t[a] = 0$. Therefore, both its mass and content remain exactly unchanged. Unaddressed slots thus need not be accessed at all, and each step accesses only the slots selected by $\mathbf{w}_t$, as required by sparse addressing. (ii) **Bounded content.** Since $\lambda_t[a]\, m_{t-1}[a] \geq 0$, the mass satisfies $m_t[a] \geq w_t[a] \geq 0$, and with $\epsilon > 0$ the learning rate satisfies $\eta_t[a] \in [0, 1)$. Each content update is therefore a convex combination of the previous content and $\mathbf{v}_t$. Every slot thus stays within the convex hull of its initial content and the values written to it, so its norm never exceeds the largest norm among these values, regardless of how often it is updated. (iii) **Adaptive learning rate.** The learning rate $\eta_t[a]$ is adapted at two levels. First, each slot's learning rate depends on its current write weight $w_t[a]$ relative to its historical mass $m_{t-1}[a]$. A stronger write increases the learning rate, whereas a larger historical mass provides greater inertia and suppresses the update. Second, the input-dependent modulator γ_t controls how much of this historical mass is retained through the retention factor $\lambda_t[a]$. This allows the model to balance preserving history against adopting new content for each token and head. As $\gamma_t \to -\infty$, $\lambda_t[a] \to 1$ and the slot becomes a write-weighted running average of the values written to it. As $\gamma_t \to +\infty$, $\lambda_t[a] \to 0$ and $\eta_t[a] \approx 1$, so each write nearly overwrites the slot.

Read-out. The read weights retrieve $\mathbf{o}_t = \mathbf{r}_t \mathbf{S}_t = \sum_{a \in \mathcal{A}} r_t[a]\mathbf{s}_t[a]$ from the selected slots. Finally, $\mathbf{o}_t$ passes through a head-wise RMSNorm [54], a sigmoid output gate, and the output projection, giving $\mathbf{y}_t = \big(\sigma(\mathbf{x}_t W_g) \odot \mathrm{RMSNorm}(\mathbf{o}_t)\big) W_o$, as shown in Fig. 2.

4 Implementation

Efficient Address Decoding. For each token and head, the write and read addresses are decoded separately by the Top-K tree search in §3.2. We first sort the d_p logits and compute their softmax for each of the U digit distributions entirely in registers, at a total cost of $\mathcal{O}(U d_p \log d_p)$. The search then proceeds level by level, using a wavefront scan to combine the K retained prefixes with the d_p choices for the next digit and select the K most probable extensions, at a cost of $\mathcal{O}(K d_p)$ per level. The total per-step cost is therefore $\mathcal{O}(UKd_p + U d_p \log d_p)$, far below the $\mathcal{O}(|\mathcal{A}|)$ cost of an exhaustive search. Since an address probability is a product of U digit probabilities, all computations are carried out in log space, which turns the product into a sum and avoids numerical underflow.

Slot-Parallel Execution. Although each step accesses only a few slots, the selected slots are scattered across the state and differ from token to token, which makes the accesses irregular on GPUs. During training and prefill, we therefore organize the computation by slot rather than by token. First, the write and read addresses of all tokens are decoded in parallel. Next, all read and write events

Table 1: Zero-shot performance on language modeling and commonsense reasoning.

Model	Wiki. ppl ↓	LMB. acc ↑	MMLU acc ↑	ARC-e acc ↑	ARC-c acc ↑	OBQA acc_n ↑	SciQ acc ↑	BoolQ acc ↑	COPA acc ↑	PIQA acc ↑	Hella. acc ↑	Wino. acc ↑	Avg. acc ↑	Act. State[1] per step ↓
340M														
Transformer++	26.81	33.1	23.1	57.7	24.7	32.0	82.9	55.9	66.0	65.8	33.3	52.2	49.4	100.7M
Gated DeltaNet	26.85	32.4	23.4	57.9	25.0	35.0	83.3	58.9	67.0	66.8	**33.9**	50.2	50.1	4.4M
DeltaNet	27.61	32.0	22.9	58.1	24.9	31.6	80.0	61.2	**71.0**	65.5	33.2	50.0	49.8	3.4M
Mamba2	26.23	**33.9**	23.1	59.2	24.2	33.8	**83.4**	59.6	**71.0**	**67.7**	**33.9**	50.4	50.6	13.0M
GLA	30.03	30.4	23.2	56.9	25.0	32.4	81.6	56.8	66.0	66.4	32.7	50.7	49.2	3.1M
GSA	29.17	32.7	23.0	59.3	25.8	**35.8**	83.0	57.4	66.0	66.4	33.3	49.8	50.0	3.1M
MoM	28.27	30.2	22.9	56.8	25.9	31.8	82.5	56.3	66.0	65.4	32.8	51.0	49.1	5.2M
RAM-Net														
Top-8	26.89	28.7	23.6	59.1	24.7	34.4	82.5	57.4	67.0	66.9	33.4	**54.1**	50.3	**0.4M**
Top-16	26.23	29.7	23.5	**59.4**	**26.1**	33.0	82.8	58.6	69.0	67.2	33.8	51.4	50.5	0.8M
Top-32	**25.87**	30.9	**23.8**	58.9	**26.1**	32.2	82.8	**61.6**	70.0	67.2	33.7	52.7	**50.9**	1.7M
1.3B														
Transformer++	17.08	46.9	23.0	70.2	34.0	39.2	88.7	59.1	73.0	71.3	42.8	58.7	56.0	201.3M
Gated DeltaNet	**15.85**	**50.1**	**25.5**	72.4	**38.7**	42.0	**91.9**	**63.6**	76.0	73.5	45.2	58.2	**58.7**	17.0M
Mamba2	15.96	49.9	24.5	71.9	36.3	42.6	88.9	55.1	**81.0**	73.5	**45.6**	58.2	57.8	26.0M
RAM-Net														
Top-8	16.49	43.0	23.6	72.0	36.3	41.8	88.2	60.3	76.0	73.0	44.6	57.9	57.4	**0.8M**
Top-16	15.90	47.4	23.6	71.8	37.6	40.8	89.6	58.3	79.0	72.9	44.9	**59.1**	57.8	1.7M
Top-32	16.02	46.7	24.8	**72.8**	36.6	**43.6**	90.6	57.2	79.0	**73.7**	44.9	57.8	58.1	3.3M

[1] denotes the state size accessed per step; Transformer++ is measured at context length 2048. This equals the total state size for all models except MoM and RAM-Net, whose total state sizes are 8.6M for MoM at 340M and 26.6M/53.2M for RAM-Net at 340M/1.3B.

are sorted by slot, so that each slot receives a time-ordered list of the events that access it, and slots without events are skipped. Finally, different slots and the d_v channels within each slot are processed in parallel. Within each slot, the events are applied in time order as required by the recurrence, and each read result is accumulated back to its token. The total work thus scales with the number of events, which is proportional to the sequence length times K, rather than with the full state size. During autoregressive decoding, each new token simply updates and reads its own K selected slots.

5 Experiments

Baselines and Experimental Setup. We compare RAM-Net against Transformer++ [43], Mamba2 [11], GLA [50], GSA [57], DeltaNet [36], MoM [12], and Gated DeltaNet [49], with all baselines using the implementations provided by the Flash-Linear-Attention library [48]. All models are trained using the Flame framework [56] on the FineWeb-Edu dataset [25] with the Llama2 tokenizer [44] and a context window of 4,096 tokens. We train models at two scales: 340M parameters on 10B tokens, and 1.3B parameters on 100B tokens. For both scales, we use AdamW optimizer [23] with weight decay of 0.1, gradient clipping of 1.0, cosine decay schedule, and a global batch size of 0.5M tokens. The peak learning rate and warmup tokens are set to 1.0×10^{-3} and 500M for the 340M scale, and 4.0×10^{-4} and 1B for the 1.3B scale, respectively (see training details in Appendix A). The ablation studies are reported in Appendix B.

Language Modeling and Commonsense Reasoning. We evaluate zero-shot language modeling and commonsense reasoning with the Language Model Evaluation Harness [14] on WikiText-103 [26], LAMBADA (LMB) [29], MMLU [15], ARC-Challenge/Easy [10], OpenBookQA [27], SciQ [46], BoolQ [9], COPA [33], PIQA [6], HellaSwag [53], and WinoGrande [35]. The results are summarized in Table 1. First, RAM-Net achieves the lowest WikiText perplexity at 340M and performs on par with the strongest baseline Gated DeltaNet at 1.3B. Second, its commonsense reasoning remains competitive. RAM-Net Top-32 obtains the highest average accuracy at 340M and the second highest at 1.3B. Third, increasing Top-K consistently improves average accuracy at both scales, suggesting that accessing more slots per step strengthens commonsense reasoning. Notably, RAM-Net achieves these results while accessing far fewer state elements per step than every baseline.

Synthetic Associative Recall. We evaluate RAM-Net on multi-query associative recall (MQAR) [2] using Zoology, a synthetic benchmark that tests whether a model can store key-value pairs and retrieve the correct value from a query key under controlled conditions. To focus on architectural recall capacity, we train all models from scratch for this task. Beyond the baselines above, we additionally include linear attention [18] and sliding window attention (SWA) [5]. We follow the Zoology evaluation protocol and report average accuracy over the standard MQAR settings. As shown in Fig. 4, RAM-Net (Top-8/16) lies on the upper envelope against total state size, reaching high accuracy at smaller total states than most linear recurrent baselines. Moreover, RAM-Net reaches this accuracy while accessing only a small fraction of its state per step, about $10\times$ less than baselines of

Table 2: Zero-shot performance comparison on S-NIAH benchmark. All models are trained with a 4K context window.

	S-NIAH-1					S-NIAH-2					S-NIAH-3					
Model	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K	Avg.
340M																
Transformer++	99.7	97.7	81.0	0.0	0.0	**100.0**	99.7	74.7	0.0	0.0	81.7	62.7	31.3	0.0	0.0	48.6
Gated DeltaNet	99.3	99.3	98.7	88.0	44.0	99.7	96.0	37.3	20.3	**10.3**	46.0	26.7	2.3	0.7	0.3	51.3
DeltaNet	**100.0**	**100.0**	**100.0**	99.7	98.7	94.3	95.7	18.0	23.0	7.0	48.7	21.7	7.0	2.7	**1.3**	54.5
Mamba2	89.3	91.7	77.0	24.7	6.7	97.3	72.7	12.0	11.7	4.0	20.7	10.3	5.3	1.3	0.7	35.0
GLA	68.7	23.7	5.7	0.3	0.3	64.7	20.3	6.3	3.3	1.0	5.0	0.3	0.0	0.0	0.0	13.3
GSA	75.3	54.7	31.7	13.7	7.3	69.0	49.7	11.0	7.0	1.7	0.0	0.0	0.0	0.0	0.0	21.4
MoM	**100.0**	99.7	98.7	89.7	59.0	97.0	76.0	26.3	10.0	0.7	19.3	8.0	1.7	0.3	0.3	45.8
RAM-Net																
Top-8	**100.0**	**100.0**	99.7	99.7	99.7	99.3	99.3	83.7	20.7	5.3	69.3	69.3	18.3	5.7	1.0	64.7
Top-16	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.7	88.0	23.3	2.7	70.7	58.0	25.7	7.3	**1.3**	65.1
Top-32	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**94.0**	**56.3**	4.0	**95.3**	**88.0**	**44.7**	**9.7**	1.0	**72.9**
1.3B																
Transformer++	**100.0**	**100.0**	95.0	0.0	0.0	**100.0**	**100.0**	**90.5**	4.0	0.0	**98.0**	**91.6**	**66.0**	0.4	0.0	56.4
Gated DeltaNet	**100.0**	99.7	99.0	61.7	29.0	**100.0**	99.3	77.0	30.0	14.3	72.7	62.7	20.3	9.7	4.3	58.6
Mamba2	**100.0**	**100.0**	**100.0**	99.7	71.0	99.0	95.0	65.7	17.0	5.7	89.7	69.7	15.0	6.3	2.3	62.4
RAM-Net																
Top-8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.7	99.7	82.7	42.3	9.3	63.0	54.0	16.0	8.3	0.3	65.0
Top-16	**100.0**	99.7	99.7	99.7	99.7	**100.0**	97.7	85.7	**49.0**	**18.7**	74.7	63.3	39.0	**18.3**	**6.3**	**70.1**
Top-32	99.3	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	98.0	80.7	33.3	10.7	78.0	73.3	21.0	9.0	1.7	67.0

comparable accuracy. These results suggest that sparse addressing makes more effective use of the total state, and that restricting each step to a small part of it does not compromise recall.

Long-Context Retrieval. We evaluate long-context retrieval on the single needle-in-a-haystack (S-NIAH) tasks from RULER [16][‡] with context lengths from 1K to 16K, up to four times the 4K training length (Table 2). RAM-Net achieves the highest average accuracy at both scales. On S-NIAH-1, where the needle is hidden among repeated filler sentences, all RAM-Net variants remain near-perfect up to 16K tokens. In contrast, Transformer++ fails beyond its training length, and most recurrent baselines degrade. We attribute this robustness partly to sparse addressing, as content-only heads write the repeated filler into the same few slots and thus leave the slots holding the needle largely undisturbed. Unlike S-NIAH-1, S-NIAH-2/3 use diverse natural text as filler and are therefore harder. On these tasks, Top-32 performs best up to 8K tokens at 340M; at 1.3B, Transformer++ leads within its training length, whereas Top-16 performs best beyond it, although all models remain weak at 16K. Finally, Top-16 and Top-32 both achieve higher average accuracy than Top-8 at both scales, suggesting that accessing more slots per step benefits retrieval.

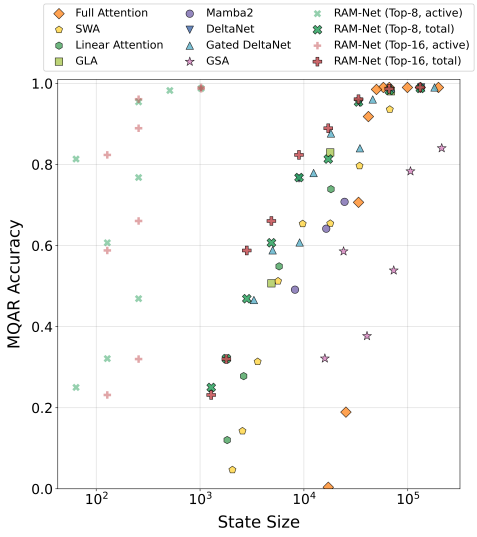


Figure 4: MQAR accuracy versus state size. Active state in RAM-Net denotes the effective state accessed per step.

Real-World Recall. We further evaluate in-context recall on the recall-intensive tasks [4][‡], which require extracting information from real documents. As shown in Table 3, Transformer++ achieves the highest average at both scales, since it stores every context token in its KV cache and can retrieve from any of them directly. Among recurrent models, RAM-Net Top-32 attains the highest average at both scales, and at 340M it narrows the gap to Transformer++ from 8.2 points for the best baseline to 3.1 points.

[‡]All models are pretrained only with next-token prediction, so we evaluate S-NIAH and the recall tasks as cloze-style completion [16, 4].

Table 3: Zero-shot performance on recall-intensive tasks. FDA, SWDE, and NQ are evaluated at context lengths 512 and 1K; the remaining tasks use the full document.

Model	FDA 512 ↑	FDA 1K ↑	SWDE 512 ↑	SWDE 1K ↑	NQ 512 ↑	NQ 1K ↑	SQuADv2 full ↑	TriviaQA full ↑	DROP full ↑	Avg.
340M										
Transformer++	**50.5**	**37.8**	**44.2**	**37.8**	**23.5**	**24.8**	**35.3**	44.7	17.0	**35.1**
Gated DeltaNet	22.1	15.1	37.8	17.0	21.8	22.7	33.0	47.7	17.7	26.1
DeltaNet	22.4	16.1	32.0	9.5	20.6	20.2	27.7	37.3	**21.7**	23.1
Mamba2	29.8	22.1	36.7	18.7	22.3	20.6	32.3	44.3	15.3	26.9
GLA	11.0	7.4	25.5	11.9	17.6	18.9	25.7	40.7	15.7	19.4
GSA	7.4	5.4	25.2	9.9	16.0	15.5	25.0	40.0	17.7	18.0
MoM	14.7	9.0	30.6	18.7	18.9	16.8	28.0	42.0	16.3	21.7
RAM-Net										
Top-8	29.8	20.4	30.3	16.7	17.6	18.1	24.3	34.7	16.7	23.2
Top-16	38.8	27.8	40.5	19.4	21.4	17.2	29.7	38.3	18.0	27.9
Top-32	46.8	36.8	42.5	24.8	21.8	21.4	26.7	**48.3**	18.7	32.0
1.3B										
Transformer++	**79.9**	**70.6**	**48.6**	**48.6**	**36.1**	**35.7**	**49.7**	56.3	22.3	**49.8**
Gated DeltaNet	54.2	41.8	43.9	40.8	34.9	32.4	39.7	**61.3**	22.3	41.3
Mamba2	48.5	34.4	39.5	29.9	34.5	34.5	38.0	57.0	21.3	37.5
RAM-Net										
Top-8	65.9	56.5	37.1	36.4	27.7	26.5	34.7	57.3	**23.3**	40.6
Top-16	60.5	49.5	41.2	36.7	29.8	31.1	30.0	56.0	16.3	39.0
Top-32	68.6	58.9	42.9	36.4	**36.1**	29.0	34.7	56.3	21.3	42.7

Decoding Throughput. We benchmark autoregressive decoding throughput of the 340M models on a single NVIDIA RTX 3060 as shown in Fig. 5. Each decoding step is captured in a CUDA graph to exclude Python and kernel-launch overhead. At batch size 1, decoding is bound by reading the model weights, and all models run at similar speed. As the batch grows, reading and updating the recurrent state becomes the bottleneck. Since RAM-Net Top-8 accesses the least state per step, it achieves 2.0× the throughput of the fastest baseline at batch size 64, although its total state is the largest among the recurrent models in Fig. 5. This shows that RAM-Net decouples decoding speed from total state size. Its large total state supports the strong retrieval in Table 2, while its sparse access keeps decoding fast. The remaining trade-off is set by K, as a larger K improves accuracy but lowers throughput.

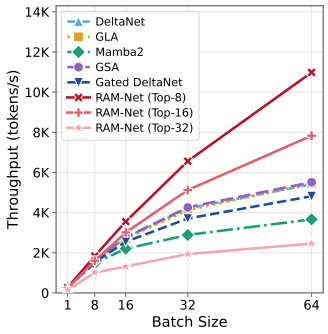


Figure 5: Decoding throughput on a single RTX 3060.

6 Related Work

Linear Recurrent Models and State Expansion. Linear recurrent models replace the growing KV cache of full attention with a fixed-size recurrent state [18, 39]. Early variants, such as Linear Transformer [18] and RetNet [39], aggregate history through kernel feature maps or fixed decay rules that do not depend on the input. Subsequent models make the state update data-dependent. Specifically, Mamba2 [11], GLA [50], and HGRN2 [32] use input-driven gating or decay, whereas DeltaNet [36], Gated DeltaNet [49], RWKV-7 [30], and Kimi Linear [42] edit the state through delta-rule error correction. TTT [38] further views the state as fast weights trained online on a self-supervised loss.

Another line of work targets the limited capacity of fixed recurrent states. MoM [12] routes each token to a few of several memory states, and SSE [28] similarly selects a few partitions of an expanded state, so that different tokens update different regions. However, each selected region remains a dense matrix that is updated and read as a whole. In contrast, GSA [57] organizes the state into finer units called slots, but updates all slots with gated decay and reads them through a softmax at every step. Instead, RAM-Net makes each individual slot the unit of selective access, as each key or query addresses only a few of many slots. The per-step access therefore stays small while the number of slots can grow, and interference is confined to co-addressed slots, leaving all other slots unchanged.

Sparse Memory Addressing. Beyond recurrent states, discrete content-based addresses have also been used to retrieve from large sets of key-value pairs. Transformer-VQ [21] quantizes each key

to a codeword and aggregates values under the codewords, but each query still reads all codewords densely. Conversely, Product-Key Memory (PKM) [20] retrieves only the top-k of millions of slots through a product search over two sub-key sets. However, PKM slots are slow weights fixed at inference, so they store dataset-level knowledge rather than the current sequence. FwPKM [58] extends PKM into a fast-weight memory that updates the selected slots online by gradient descent on a reconstruction loss. Nevertheless, FwPKM is an auxiliary module placed before an attention or linear attention layer. It updates its memory only once per chunk and has no explicit forgetting. In contrast, RAM-Net is itself an attention layer whose recurrent state is read and written through sparse addresses. Its Address Decoder factorizes each address into U digits and produces the write and read weights at every step. GSU then updates the few written slots with learned forgetting.

Sparse Attention. Sparse attention reduces the cost of full attention by restricting each query to a subset of past tokens. Early methods use predefined sparse patterns, such as the fixed and strided factorizations in Sparse Transformer [7], the sliding-window and global attention in Longformer [5], the global-local attention in ETC [1], and the local-global-random pattern in BigBird [52]. Subsequent methods make the sparse pattern content-dependent, for example through locality-sensitive hashing in Reformer [19], online clustering in Routing Transformer [34], differentiable sorting in Sparse Sinkhorn Attention [41], or Top-K selection in SparseK Attention [24]. More recently, long-context systems select KV-cache blocks in a query-aware or hardware-aligned manner [40, 17, 51], or retrieve relevant tokens through product quantization of cached keys [55]. Despite these differences, these methods sparsify only the reads, as every token is still written into a KV cache that grows with sequence length. In contrast, RAM-Net also sparsifies the writes, since each token writes only to its K addressed slots in a fixed-size state. As a result, its per-step cost stays constant in sequence length.

7 Limitations and Future Work

First, our experiments are limited to models with up to 1.3B parameters. Whether the advantages of sparse addressing persist at larger scales remains to be verified.

Second, sparse addressing involves an inherent recall trade-off. Restricting each read and write to a few slots reduces interference, but relevant content can be missed when the query is not co-addressed with it. This limitation is most evident when the context contains diverse distractors over long ranges. More flexible addressing, such as adaptive access, may mitigate this issue.

Third, GSU updates each addressed slot independently. Richer update rules that coordinate co-addressed slots, such as delta-rule-style joint corrections [36, 49], remain to be explored.

Finally, our implementation does not yet exploit the structure of the learned access patterns, in which a few slots are accessed far more frequently than others. This skew suggests system-level optimizations, such as keeping frequently accessed slots in fast memory and storing the rest in lower-cost memory.

8 Conclusion

In this work, we introduced RAM-Net, a linear-time sequence model that replaces dense access to a shared recurrent state with sparse address-based read and write over a fixed slot array. By making token interactions depend on overlap in addressed slots, RAM-Net restricts retrieval interference to a small subset of the state, while its per-token state access remains independent of the total number of slots and its addressing cost grows only logarithmically with it. Empirically, RAM-Net shows clear advantages on fine-grained retrieval benchmarks over recurrent baselines, and remains competitive on standard language modeling and commonsense reasoning tasks. These results suggest that explicit sparse addressing is a promising paradigm for building fixed-state sequence models with improved long-range recall.

Acknowledgments and Disclosure of Funding

This work was supported by the Research Grants Council (RGC) of Hong Kong under Grant CUHK 14212323 and CUHK R1018-25. The authors declare no competing interests.

References

[1] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, 2020.

[2] 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 *Proceedings of 12th International Conference on Learning Representations (ICLR)*. ICLR, 2024.

[3] 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. *arXiv preprint arXiv:2402.18668*, 2024.

[4] Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models, 2024. URL https://arxiv.org/abs/2407.05483.

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

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

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[8] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[9] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, 2019.

[10] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

[11] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[12] Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. Mom: Linear sequence modeling with mixture-of-memories. *arXiv preprint arXiv:2502.13685*, 2025.

[13] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. 23(1), January 2022. ISSN 1532-4435.

[14] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

[15] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

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

[17] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

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

[19] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[20] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Lucas D Lingle. Transformer-vq: Linear-time transformers via vector quantization. *arXiv preprint arXiv:2309.16354*, 2023.

[22] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[24] Chao Lou, Zixia Jia, Zilong Zheng, and Kewei Tu. Sparser is faster and less is more: Efficient sparse attention for long-range transformers. *arXiv preprint arXiv:2406.16747*, 2024.

[25] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[26] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

[27] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[28] Yuqi Pan, Yongqi An, Zheng Li, Yuhong Chou, Ruijie Zhu, Xiaohui Wang, Mingxuan Wang, Jinqiao Wang, and Guoqi Li. Scaling linear attention with sparse state expansion. *arXiv preprint arXiv:2507.16577*, 2025.

[29] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL https://arxiv.org/abs/1606.06031.

[30] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7 "goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

[31] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

[32] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion, 2024. URL https://arxiv.org/abs/2404.07904.

[33] Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011. URL https://people.ict.usc.edu/~gordon/publications/AAAI-SPRING11A.PDF.

[34] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9: 53–68, 2021.

[35] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

[36] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR, 2021.

[37] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[38] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

[39] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

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

[41] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International conference on machine learning*, pages 9438–9447. PMLR, 2020.

[42] Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

[43] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[44] 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.

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

[46] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions, 2017. URL https://arxiv.org/abs/1707.06209.

[47] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

[48] Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

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

[50] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, pages 56501–56523, 2024.

[51] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025.

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

[53] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

[54] Biao Zhang and Rico Sennrich. *Root mean square layer normalization*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[55] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025.

[56] Yu Zhang and Songlin Yang. Flame: Flash language modeling made easy, January 2025. URL https://github.com/fla-org/flame.

[57] Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, Peng Zhou, and Guohong Fu. Gated slot attention for efficient linear-time sequence modeling, 2024. URL https://arxiv.org/abs/2409.07146.

[58] Tianyu Zhao and Llion Jones. Fast-weight product key memory. *arXiv preprint arXiv:2601.00671*, 2026.

A Training Details

Model and Training Configuration. Tables 4 and 5 summarize the RAM-Net architecture and the training setup at the 340M and 1.3B scales. All baselines follow the default configurations of the Flash-Linear-Attention library [48, 56] and are trained with the same setup.

In each layer, some heads use CAPE and the remaining heads use NoPE. The number of CAPE heads decreases stage-wise with depth, starting from all heads in the first four layers. Writing $h^{\times l}$ for l consecutive layers with h CAPE heads each, the 340M model uses $\{16^{\times 4}, 8^{\times 4}, 4^{\times 4}, 2^{\times 4}, 1^{\times 2}, 0^{\times 7}\}$, and the 1.3B model uses $\{32^{\times 4}, 16^{\times 4}, 8^{\times 4}, 4^{\times 4}, 2^{\times 9}\}$.

Table 4: Model configurations.

Params.	**340M**	**1.3B**
Layers	25	
Hidden size	1,024	2,048
MLP intermediate size	2,816	5,632
Heads	16	32
Value channels per head	64	

Table 5: Training configuration.

Category	**Setting**	**340M params.**	**1.3B params.**
Data	Dataset	FineWeb-Edu [25]	
	Training budget	10B tokens	100B tokens
Tokenizer	Tokenizer	Llama 2 [44]	
	Vocabulary size	32k	
	Context window	4,096	
Optimization	Optimizer	AdamW [23]	
	Weight decay	0.1	
	Gradient clipping	1.0	
LR schedule	Schedule	Cosine decay [22]	
	Linear warm-up	500M tokens	1B tokens
	Peak LR	1.0×10^{-3}	4.0×10^{-4}
	Min LR	1.0×10^{-4}	4.0×10^{-5}
Batch	Per-device batch	65.5K tokens	
	Global batch size	0.5M tokens	
	Devices	NVIDIA H200	

Learnable Address Temperature. In addition to the fixed temperature τ in §3.2, each head learns a log-scale correction for every digit of its write and read addresses. Specifically, the digit logits of the key are scaled as $\exp(\beta_w^{(u)})\,\mathbf{k}_t^{(u)}/\tau$, and those of the query as $\exp(\beta_r^{(u)})\,\mathbf{q}_t^{(u)}/\tau$, where $\beta_w^{(u)}$ and $\beta_r^{(u)}$ are learnable scalars initialized to zero. The effective temperature is thus $\tau e^{-\beta}$, and the exponential parameterization keeps it positive. This allows each head to sharpen or smooth its addresses during training, starting from the base temperature $\tau = \sqrt{U}$.

Effect of the Forgetting Modulator. Fig. 6 shows how the forgetting modulator γ_t reshapes the retention factor $\lambda_t[a]$ as a function of the write weight $w_t[a]$. At $\gamma_t = 0$, the retention factor equals $1 - w_t[a]$, so the stronger a write is, the less of its previous mass the slot keeps. A positive γ_t lowers this curve, so the slot discards more of its accumulated mass and adopts new values faster. Conversely, a negative γ_t raises the curve, so the slot retains more of its mass and averages over a longer history. All curves share the same endpoints regardless of γ_t. At $w_t[a] = 0$, $\lambda_t[a] = 1$, so the modulator leaves unaddressed slots unchanged and preserves the exact sparsity property (§3.4) of GSU. At $w_t[a] = 1$, $\lambda_t[a] = 0$, so a slot written with full weight is always overwritten.

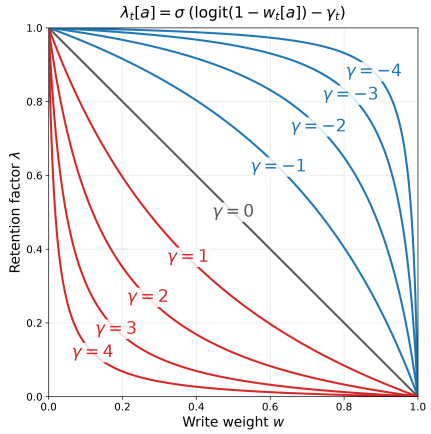


Figure 6: Retention factor $\lambda_t[a]$ as a function of the write weight $w_t[a]$ for different values of the forgetting modulator γ_t.

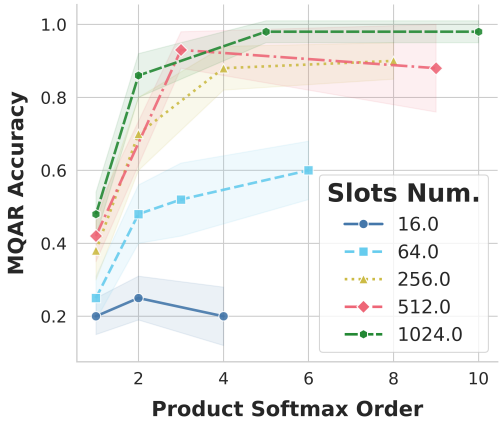


Figure 7: MQAR accuracy versus the number of address digits U for different numbers of slots $|\mathcal{A}|$, where increasing U improves accuracy.

Bias Initialization. Sparse addressing and gated updates are sensitive to initialization, so we initialize three groups of biases with fixed values. First, the output gate bias is set to 3, so that the gate starts nearly open at $\sigma(3) \approx 0.95$ and does not suppress the layer output, which lets gradients reach the addressing and update parameters from the start of training. Second, we add a constant offset $\gamma_0 = \ln 9$ to the forgetting modulator, giving $\gamma_t = \mathbf{x}_t W_\gamma + \gamma_0$. Therefore, addressed slots forget relatively quickly at initialization, which encourages early writes to spread across slots rather than accumulate in a few. Third, we add a fixed bias to the address logits so that slots start with distinct preferences rather than being symmetric. Within digit u, this bias increases linearly across the d_p choices from $-\alpha_u$ to α_u, and its magnitude α_u grows linearly across digits from 0 to r. The key and query projections share this bias, and we use $r = 1.0$ for the 340M models and $r = 1.5$ for the 1.3B models. Setting $r = 0$ removes the bias and markedly degrades retrieval at 340M (Table 6).

B Ablation Studies

B.1 Ablation on Product Softmax and Slot Count

We conduct an ablation study on MQAR [2] to examine how the Address Decoder affects retrieval capability. Specifically, we vary the Product Softmax order U and the total slot count $|\mathcal{A}|$, thereby separating the effect of Product Softmax from the effect of state size.

Order U of Product Softmax. We first study the role of the Product Softmax order U. When $U = 1$, the address is produced by applying a single softmax over all $|\mathcal{A}|$ slots followed by Top-K truncation. For $U > 1$, the model applies softmax separately to U low-dimensional sub-vectors, and their Kronecker product gives the probability of each slot before Top-K selection. As shown in Fig. 7, increasing U substantially improves MQAR accuracy across most slot counts, with the gains most pronounced when the slot count is large. This suggests that product softmax is important not only for efficient Top-K decoding, but also for making large address spaces easier to optimize.

A likely reason is that, with $U = 1$, the address is a single softmax over all $|\mathcal{A}|$ slots, and after Top-K truncation only the K selected entries receive a direct learning signal at each step. With $U > 1$, each digit is shared by many slots, so the learning signal from a selected slot also reaches every slot that shares its digits. This denser learning signal makes large address spaces easier to optimize, consistent with the larger gains at large slot counts in Fig. 7.

Slot Count $|\mathcal{A}|$. We next study the effect of the total number of slots. For a fixed or sufficiently large Product Softmax order, larger slot counts generally improve MQAR accuracy, with the best results obtained at $|\mathcal{A}| = 1024$. This trend is consistent with the role of slots as addressable storage units: increasing $|\mathcal{A}|$ enlarges the address space and reduces the chance that unrelated key-value pairs must share the same slots.

Table 6: Ablation study of RAM-Net design choices. The default setting uses $|\mathcal{A}| = 4^5$ slots, Top-$K = 8$, and the CAPE setting in Appendix A. All variants are trained with a 4K context window unless noted otherwise. Results are reported on language modeling, commonsense reasoning, and 4K S-NIAH retrieval tasks.

Variant	Wiki. ppl ↓	LMB. acc ↑	MMLU acc ↑	ARC-e acc ↑	ARC-c acc ↑	OBQA acc_n ↑	SciQ acc ↑	boolQ acc ↑	COPA acc ↑	PIQA acc ↑	Hella. acc ↑	Wino. acc ↑	S-NIAH-1 acc ↑	S-NIAH-2 acc ↑	S-NIAH-3 acc ↑
RAM-Net															
Top-8 (default)	26.89	28.7	23.6	59.1	24.7	34.4	82.5	57.4	67.0	66.9	33.4	54.1	99.7	83.7	18.3
Top-16	26.23	29.7	23.5	59.4	26.1	33.0	82.8	58.6	69.0	67.2	33.8	51.4	100.0	88.0	25.7
Top-32	25.87	30.9	23.8	58.9	26.1	32.2	82.8	61.6	70.0	67.2	33.7	52.7	100.0	94.0	44.7
Ablations															
– write Top-4, read Top-12	27.98	27.2	23.7	58.8	26.5	33.4	81.4	60.2	68.0	66.2	33.1	51.9	98.0	45.3	17.0
– write Top-12, read Top-4	26.59	29.3	23.1	59.7	25.8	33.8	82.8	62.0	70.0	66.3	33.6	51.1	100.0	48.7	7.7
– $\|\mathcal{A}\| = 4^4$ slots	26.47	28.0	23.6	59.1	26.5	34.6	82.3	55.0	67.0	67.5	33.7	53.3	100.0	41.7	13.7
– $\|\mathcal{A}\| = 4^6$ slots	27.09	27.6	24.0	60.0	25.5	34.2	82.4	60.8	67.0	67.5	33.6	52.0	100.0	10.0	14.3
– $\|\mathcal{A}\| = 4^6$ slots, 8K training	26.90	26.8	23.9	58.6	25.1	33.6	81.0	54.9	67.0	66.4	33.1	51.6	100.0	34.0	7.0
– fixed $\gamma_t = 0$	27.59	27.6	23.6	57.2	25.3	30.6	82.6	61.7	69.0	66.1	33.0	50.9	100.0	31.0	30.3
– w/o output RMSNorm	28.87	26.3	23.8	57.9	24.1	33.0	82.0	59.6	67.0	66.6	32.6	53.1	99.7	16.3	12.3
– w/o output gate	27.56	28.6	23.6	57.4	25.0	32.0	81.5	59.8	70.0	65.6	32.8	51.6	100.0	50.3	25.0
– $r = 0.0$	28.18	27.1	23.5	56.7	24.8	32.6	81.0	59.4	70.0	67.0	32.5	53.1	100.0	27.7	5.3
– $r = 1.5$	26.63	29.4	24.6	59.7	25.7	34.0	81.2	58.2	67.0	66.5	34.0	52.3	100.0	58.7	25.7
– $\tau = 1.5$	26.61	29.5	24.4	58.0	24.8	33.0	81.5	56.1	64.0	67.0	33.4	52.3	100.0	56.0	25.0
– Top-K renorm	26.83	27.8	23.1	60.8	25.6	33.6	83.3	60.2	70.0	67.9	33.8	51.0	100.0	63.7	12.0
– NoPE only	28.04	30.0	23.0	58.4	24.9	32.2	82.3	61.4	69.0	66.8	33.3	54.4	100.0	43.0	8.7
– NoPE + conv	27.13	30.2	22.8	58.9	24.7	32.8	81.0	55.4	70.0	66.0	33.2	50.6	100.0	16.3	1.0
– CAPE + conv	26.84	28.9	23.7	58.2	24.1	34.4	80.6	59.4	69.0	66.8	33.7	50.9	99.7	88.3	15.0
– uniform CAPE	26.90	28.0	23.8	57.7	24.0	33.2	81.4	61.6	66.0	67.0	33.3	51.1	99.7	22.7	10.0
– 60% CAPE	27.12	27.9	24.6	57.8	25.7	32.6	79.4	57.9	70.0	66.4	33.4	50.1	100.0	16.3	6.0
– peak LR 10^{-2}	30.86	25.0	23.4	56.9	25.3	32.8	81.7	57.6	69.0	64.5	32.4	52.2	100.0	14.7	9.0
– peak LR 10^{-4}	44.57	18.3	22.9	50.0	20.1	29.4	77.8	62.0	59.0	60.8	28.7	51.1	77.7	21.3	0.0
– slot balance	27.10	26.0	23.6	58.3	26.2	36.0	82.0	60.3	68.0	66.5	32.9	52.6	100.0	11.3	5.7

B.2 Ablation on Other Design Choices

Table 6 evaluates the remaining design choices on language modeling and S-NIAH retrieval at the 340M scale. The default configuration uses $|\mathcal{A}| = 4^5$ slots, Top-$K = 8$, $\tau = \sqrt{5} \approx 2.24$, $r = 1.0$, the learned forgetting modulator γ_t, output RMSNorm and output gate, the hybrid CAPE configuration in Appendix A, and a peak learning rate of 10^{-3}. We observe no loss spikes in any run in the table, and none of them requires a restart. In the following, we discuss the results by design aspect.

Per-step state access. Accessing more slots consistently helps. Increasing Top-K from 8 to 32 lowers Wiki perplexity from 26.89 to 25.87 and raises S-NIAH-2/3 accuracy from $83.7/18.3$ to $94.0/44.7$. At a fixed budget of 16 accessed slots, the symmetric split of eight writes and eight reads gives the best retrieval. Writing 4 and reading 12 raises perplexity by 1.09 and lowers S-NIAH-2 to 45.3, whereas writing 12 and reading 4 slightly improves perplexity but lowers S-NIAH-2/3 to $48.7/7.7$.

Slot count. With Top-K fixed, the default slot count $|\mathcal{A}| = 4^5$ performs best on retrieval. Reducing it to 4^4 lowers S-NIAH-2 to 41.7, consistent with the MQAR trend in Appendix B.1, where fewer slots force more unrelated content to share the same slots. However, increasing it to 4^6 lowers S-NIAH-2 further to 10.0 and raises perplexity by 0.20. Training this variant with an 8K context under the same token budget does not recover retrieval ($34.0/7.0$), so the degradation is not caused by an insufficient training context. A likely reason is that, with K fixed, enlarging the address space reduces the chance that a query is co-addressed with the slots holding the relevant content. The slot count therefore balances interference at small $|\mathcal{A}|$ against missed reads at large $|\mathcal{A}|$.

Forgetting modulator. The learned modulator γ_t matters for both language modeling and retrieval. Fixing $\gamma_t = 0$ reduces the retention factor to $1 - w_t[a]$, so each slot forgets only in proportion to the current write weight. This raises perplexity by 0.70 and reduces S-NIAH-2 from 83.7 to 31.0.

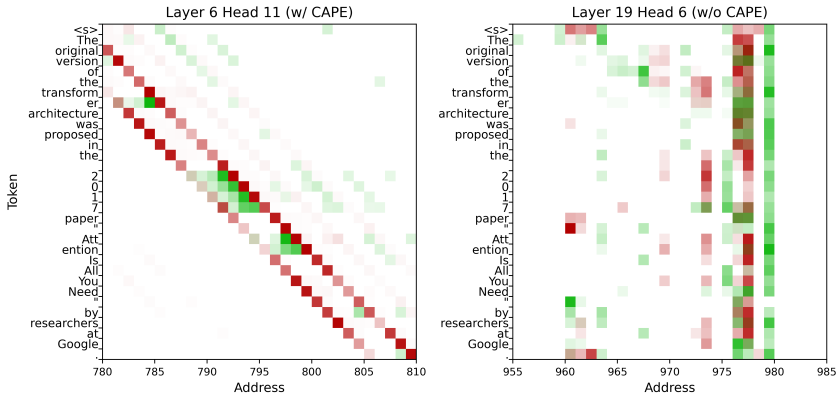
Read-out. Output RMSNorm is essential for retrieval. Removing it increases Wiki perplexity by 1.98 and reduces S-NIAH-2 from 83.7 to 16.3. This effect persists even though GSU already keeps each slot on the scale of the values, suggesting that output RMSNorm mainly stabilizes the read output, whose scale varies with the unnormalized read weights. The output gate contributes less. Removing it raises perplexity by 0.67 and reduces S-NIAH-2 to 50.3.

Addressing. The address bias, temperature, and normalization all affect retrieval. Setting $r = 0$ gives all slots the same initial addressing preference, increasing perplexity by 1.29 and reducing

S-NIAH-2/3 to $27.7/5.3$. Sharpening the addresses, either by a larger bias ($r = 1.5$) or a lower temperature ($\tau = 1.5$), slightly improves perplexity but reduces S-NIAH-2 to 58.7 and 56.0, respectively. Renormalizing the retained Top-K weights to sum to one, instead of keeping them unnormalized as in the default, reduces S-NIAH-2 to 63.7.

Positional encoding. CAPE is important for retrieval. Using NoPE in all heads reduces S-NIAH-2/3 to $43.0/8.7$. Adding a short convolution with kernel size 4 to NoPE (following Mamba2 and Gated DeltaNet) improves perplexity but further reduces retrieval to $16.3/1.0$, indicating that local token mixing cannot replace CAPE. Adding the same convolution to the default configuration changes retrieval only marginally ($88.3/15.0$). In contrast, changing the CAPE layout removes its gain. Assigning CAPE to five heads in every layer (uniform CAPE) keeps the share at about 30% but reduces S-NIAH-2 to 22.7. Raising the share to around 60% with a similar decreasing schedule (60% CAPE), namely $\{16^{\times 9}, 8^{\times 9}, 4^{\times 4}, 2^{\times 2}, 1^{\times 1}\}$, reduces it to 16.3. Both layouts fall below NoPE in all heads, suggesting that position-shifted addressing is beneficial in the early layers but displaces content-based addressing when extended to the later layers.

Optimization and regularization. The default peak learning rate follows the Flame setting [56] shared by all baselines. A larger rate of 10^{-2} degrades both perplexity and retrieval, while a smaller rate of 10^{-4} learns too slowly and lags far behind within the 10B-token budget. We also test an MoE-style load-balancing loss [13] on the write addresses, which penalizes, for each head, the product of how often each slot is selected and the address probability it receives. This loss reduces S-NIAH-2/3 to $11.3/5.7$, suggesting that the uneven slot usage learned by RAM-Net (Appendix C) is beneficial rather than a defect to be corrected.



(a) CAPE head (b) NoPE head

Figure 8: Zoomed-in state access traces on the input *"The original version of the transformer architecture was proposed in the 2017 paper 'Attention Is All You Need' by researchers at Google."* Each cell shows whether token t (vertical axis) issues a **write** (red) or **read** (green) event at slot address (horizontal axis); color intensity encodes the corresponding write or read strength. (a) A CAPE head, in which the cyclic shift produces a clear diagonal trace. (b) A NoPE head, in which addresses are determined purely by content.

C State Access Trace Analysis

To better understand what RAM-Net learns under sparse addressing, we visualize the state access traces of two representative heads on a short factual sentence in Fig. 8, and the traces of four heads over a longer passage in Fig. 9. From these traces, we highlight three observations.

CAPE and NoPE heads exhibit complementary access styles. First, the CAPE heads (see Fig. 8a and Fig. 9a,b) produce accesses concentrated along a diagonal. This pattern directly reflects the cyclic shift in CAPE, which moves the slot index of a given content by one at each step, so that consecutive writes form a stride-one trace. In contrast, the NoPE heads (Fig. 8b and Fig. 9c,d) show no such drift, as their accesses depend only on token content and therefore fall on the same set of slots regardless

of token position. This divergence suggests a functional differentiation between the two head types, with CAPE heads organizing slots by position and NoPE heads relying on content for direct lookup.

Slot allocation within NoPE heads splits into shared and selective groups. Beyond the head-level distinction above, NoPE heads further partition their slots by usage frequency. Specifically, a small group of slots receives reads and writes from nearly every token (e.g., addresses 977–980 in Fig. 8b, and the dense horizontal bands in Fig. 9d), while the remaining slots are accessed only by a small subset of tokens with matching content. Such an emergence of always-on slots is consistent with the "attention sink" phenomenon reported in full attention [47], where a few token positions absorb a large share of attention weight across queries. Notably, attention sinks anchor at fixed token positions, whereas RAM-Net's shared slots emerge from learned address patterns. Overall, these observations indicate that sparse addressing is not used uniformly. Much like an attention sink, the model spontaneously sets aside a small fraction of its slots as sequence-level shared storage and reserves the rest for content-specific storage.

Sparse addressing links related tokens through overlapping slots. Finally, the CAPE head in Fig. 8a suggests that related tokens tend to access overlapping slot regions. In the trace, such a link appears as a red write and a green read from different tokens falling in the same slot column. We highlight three representative cases. (i) *Adjacent subword tokens.* The second sub-token of *transformer* (*"er"*) reads from the slot just written by its first sub-token (*"transform"*), and the same holds for *"Att"* and *"ention"*. This suggests that the model learns to reconnect words split by the tokenizer. (ii) *Consecutive tokens forming one entity.* The four digit tokens *"2"*, *"0"*, *"1"*, and *"7"* concentrate their reads and writes in a common slot region more densely than the surrounding tokens, suggesting that the model groups tokens that jointly denote a single entity. (iii) *Non-adjacent paired tokens.* The opening and closing quotation marks of the cited title access the same slot region although several tokens separate them, so the closing mark may retrieve content written between the pair. Since this is a CAPE head, these links depend jointly on token content and relative position, consistent with the design of CAPE in §3.3.

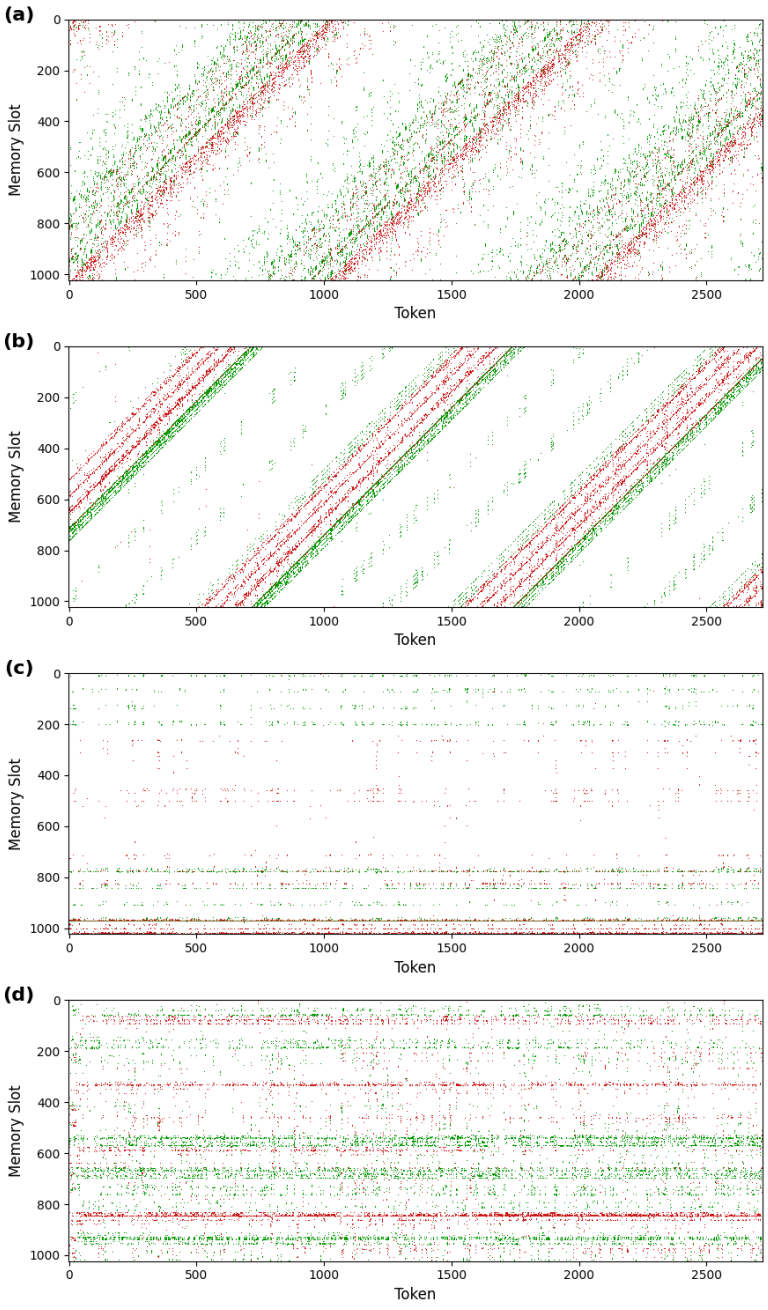


Figure 9: State access traces of four representative heads, showing **write** (red) and **read** (green) events at each slot address over tokens. CAPE heads (sub-figures (a) and (b)) produce diagonal traces, while NoPE heads (sub-figures (c) and (d)) show content-driven accesses.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have elucidated our contributions and scope in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of this paper are discussed in Section 7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: As our work is primarily empirical, it does not involve theoretical claims that require formal proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The detailed training and evaluation settings are provided in Section 5 and Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the PyTorch and CUDA implementations in the Supplementary Material. All datasets used in this paper are reported in Section 5 and publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We have detailed all the training and evaluation settings in Section 5 and Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We do not have enough resources to obtain error bars, as LLM pre-training consumes a significant amount of computational power, making it impractical to conduct multiple experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the GPU type used for conducting our experiments in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work follows the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: To our knowledge, this work poses no direct societal risks.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: To our knowledge, this work poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All of the datasets we use are publicly available, and we have properly cited all the training and evaluation datasets we used in Section 5.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Related codes of this work are released with suitable license.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLM is used only for writing.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.