Raven: High-Recall Sequence Modeling with Sparse Memory Routing

Arshia Afzal[*]
EPFL
arshia.afzal@epfl.ch

Aviv Bick[*]
Carnegie Mellon University
abick@cs.cmu.edu

Eric P. Xing
Carnegie Mellon University
MBZUAI

Volkan Cevher
EPFL

Albert Gu
Carnegie Mellon University
Cartesia AI

Abstract

Long-context recall in linear-time sequence models highlights a tradeoff in how they write to memory. State-based linear models, such as state-space models (SSMs) and linear Transformers, write *densely*, updating the entire state for each newly arrived token, which leads to interference and makes specific past tokens hard to recover. Sliding-window attention (SWA) exhibits the opposite behavior: it writes *sparsely* by storing explicit token representations, but only within a fixed window, so recall drops once the relevant token is evicted. Interpolating between these models, we introduce ***Raven***, a linear-time sequence model that maintains a fixed set of memory slots and, at each step, decays and updates only a selected subset via learned, *input-dependent routing*. This lets Raven mitigate SWA's position-based overwriting and hard eviction while reducing interference from dense state updates in SSMs, thereby preserving long-range content much more effectively. Across recall-intensive benchmarks, Raven is competitive with or outperforms prior linear-time baselines, achieving strong long-context recall where both SWA and SSMs sharply degrade. It remains effective when extrapolating to context lengths as large as 16× its training length, with similar gains in hybrid architectures. Our code and accompanying blog post are publicly available.

1 Introduction

Fixed-size memory models often match Transformers in language modeling but still fall short in long-range recall (Bick et al., 2025c). State Space Models (SSMs) (Dao and Gu, 2024; Gu and Dao, 2024) and linear Transformer variants (Katharopoulos et al., 2020; Yang et al., 2023, 2024b) are highly persistent—their state can, in principle, carry information *indefinitely*—but they write *densely*, updating the entire memory at every step, which makes individual items hard to preserve without interference. Sliding-window attention (SWA) exhibits the opposite tradeoff (Beltagy et al., 2020): it writes *sparsely* by retaining explicit token representations only within a fixed recent window, enabling reliable in-window retrieval but *no persistence* once older tokens are dropped. Together, these complementary limitations motivate separating *where* information is written from *how long* it persists in memory-based sequence models.

Motivated by this view, we propose ***Routing Slot Memories (RSMs)***, a class of sequence models that, at each step, route new content into one or more fixed memory slots and apply explicit decay to the updated slots. Under this view, SSMs and SWA emerge as two extremes on the routing axis: SSMs

[*]Equal contribution (alphabetical order).

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

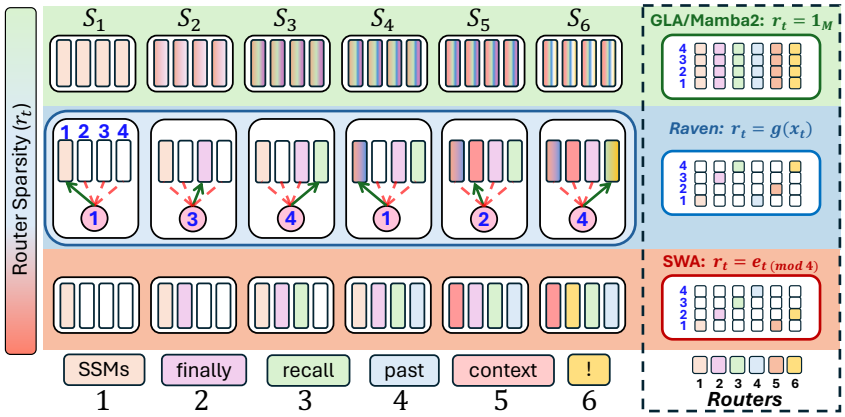


Figure 1: **Raven Overview.** Three sequence mixers using ***Routing Slot Memories*** with different routers. ***(a)*** SWA : *first-in-first-out* allocation with a **one-hot vector** $\boldsymbol{e}_t$ as router. ***(b)*** SSM : writes each token to all slots with a **dense, all-ones router** $\mathbf{1}_M$. ***(c)*** **Raven** : a **selective router** for writes. Illustration: $T = 6$ tokens, $M = 4$ memory slots, and $\text{Top}_K = 1$ for Raven.

write densely with gradual forgetting, while SWA writes sparsely and deterministically with hard eviction, as illustrated in Figure 1.

Designed to sit between these endpoints, we introduce **Raven**, an instantiation of RSMs that uses *sparse input-dependent routing to update and decay only selected memory slots*. This replaces SWA's fixed-window eviction with selective writes and gradual decay, while avoiding dense full-state updates in SSMs. The Raven block also simplifies prior linear models, requiring neither convolutional nor sliding-window attention layers for effective retrieval (Gu and Dao, 2024; Yang et al., 2023).

Raven achieves perfect recall on Needle-in-a-Haystack benchmarks where SWA performs near zero, with substantial improvements on recall-heavy tasks in standalone and hybrid settings across model scales. It is competitive with or outperforms Mamba-2, Gated Delta-Net (GDN), and Gated Linear Attention (GLA) on recall-intensive benchmarks, maintaining strong accuracy at 16× the training context length.

Further discussion of related sequence models, retrieval, length generalization, and routing methods appears in Appendix B.

2 Background

2.1 State-Space Models

Linear state-based sequence models maintain an explicit, fixed-size memory that is updated sequentially. Given an input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$, each head maintains a memory matrix $\boldsymbol{S}_t \in \mathbb{R}^{M \times d}$, where M is the value head dimension and d is the query/key head dimension (state expansion). Because each of the M rows of $\boldsymbol{S}_t$ is updated independently in all models we consider, M doubles as the number of *memory slots*; Section 3 adopts this reading throughout. At step t, the token $\boldsymbol{x}_t \in \mathbb{R}^{d_{\text{model}}}$ is linearly projected to $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^d$ and $\boldsymbol{v}_t \in \mathbb{R}^M$ [2], and updates memory via

$$\underbrace{\boldsymbol{S}_t}_{\text{Memory}} = \underbrace{\boldsymbol{S}_{t-1}\boldsymbol{A}_t}_{\text{Decay}} + \underbrace{\boldsymbol{v}_t\boldsymbol{k}_t^\top}_{\text{Write}}, \qquad \underbrace{\boldsymbol{o}_t}_{\text{Output}} = \underbrace{\boldsymbol{S}_t\boldsymbol{q}_t}_{\text{Read}}. \tag{1}$$

Here, $\boldsymbol{A}_t \in \mathbb{R}^{d \times d}$ is a (possibly input-dependent) *channel-wise* decay commonly known as forget gate (Yang et al., 2023). Some variants apply a feature map $\phi(\cdot)$ to these projections (Katharopoulos et al., 2020); for simplicity, we absorb it into $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$.

SSM variants mainly differ in the structure of $\boldsymbol{A}_t$ (see Table 2 in Yang et al. (2023)). For example, GLA uses a diagonal gate (Yang et al., 2023), Mamba-2 a scalar gate (Dao and Gu, 2024), and

[2] SSMs are often written with $(\boldsymbol{x}, \boldsymbol{B}, \boldsymbol{C})$; we use $(\boldsymbol{v}, \boldsymbol{k}, \boldsymbol{q})$ for a read/write interpretation and to stay consistent with later attention-based special cases (e.g., sliding-window attention).

DeltaNet a delta-rule update $\boldsymbol{A}_t = I - \boldsymbol{k}_t\boldsymbol{k}_t^\top$ (Schlag et al., 2021; Yang et al., 2024b). In the diagonal case $\boldsymbol{A}_t = \mathrm{diag}(\boldsymbol{a}_t)$, Equation (1) becomes

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} \odot \boldsymbol{a}_t + \boldsymbol{v}_t\boldsymbol{k}_t^\top, \qquad \boldsymbol{o}_t = \boldsymbol{S}_t\boldsymbol{q}_t. \tag{2}$$

where $\odot$ is columnwise Hadamard product (broadcast over the M rows).

2.2 Sliding-Window Attention as a Dual-State SSM

Sliding-Window Attention (SWA) stores key/value representations of the most recent M tokens and restricts attention to this window. Its two caches admit a state-space view as coupled states updated by a linear recurrence.

Let $\boldsymbol{S}_t^k, \boldsymbol{S}_t^v \in \mathbb{R}^{M\times d}$ denote the key and value caches at time t, with one memory slot per row. SWA updates the cache using a *First-In-First-Out* (FIFO) ring buffer. Let $m_t = 1 + ((t-1) \bmod M)$ denote the slot index in the FIFO buffer, and let $\boldsymbol{e}_t \in \mathbb{R}^M$ be the one-hot vector with $\boldsymbol{e}_t[m_t] = 1$. Then SWA overwrites exactly one slot per step:

$$\boldsymbol{S}_t^k = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^k + \boldsymbol{e}_t\boldsymbol{k}_t^\top, \qquad \boldsymbol{S}_t^v = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^v + \boldsymbol{e}_t\boldsymbol{v}_t^\top. \tag{3}$$

Thus, SWA computes attention over the window:

$$\boldsymbol{o}_t = (\boldsymbol{S}_t^v)^\top \,\mathrm{softmax}\big(\boldsymbol{S}_t^k\boldsymbol{q}_t\big)\,, \tag{4}$$

where the softmax is over the M slots. For $t < M$, one can either initialize $\boldsymbol{S}_0^{k,v} = 0$ and apply a validity mask in equation 4, or attend only over the filled prefix. From Equation (3), SWA applies a binary *slot-wise* decay over the M slots and removes the earliest token from memory. Stacking the two states along the feature dimension makes this connection more explicit:

$$\boldsymbol{S}_t = \begin{bmatrix}\boldsymbol{S}_t^k & \boldsymbol{S}_t^v\end{bmatrix} \in \mathbb{R}^{M\times 2d}, \qquad \boldsymbol{u}_t = \begin{bmatrix}\boldsymbol{k}_t \\ \boldsymbol{v}_t\end{bmatrix} \in \mathbb{R}^{2d},$$

which updates as

$$\boldsymbol{S}_t = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1} + \boldsymbol{e}_t\boldsymbol{u}_t^\top, \tag{5}$$

casting SWA exactly as an SSM with state size $M \times 2d$.

3 Routing Slot Memories

SSMs and SWA maintain finite-dimensional states organized into memory slots (Section 2). We formalize *where* information is written and *which slots* are preserved through **Routing Slot Memories (RSMs)**, a unified class of slot-separable linear recurrences with an explicit *routing vector*.

3.1 Memory Slots and Routing

We organize the hidden state as a matrix $\boldsymbol{S}_t \in \mathbb{R}^{M\times d}$, where each row $\boldsymbol{S}_t[i]$ is an independent memory *slot*. The general recurrence and readout are:

$$\boldsymbol{S}_t = g_t(\boldsymbol{S}_{t-1}, \boldsymbol{x}_t), \qquad \boldsymbol{o}_t = f(\boldsymbol{S}_t, \boldsymbol{x}_t). \tag{6}$$

The key structural property we require is **slot separability**: each slot updates independently of all others.

Definition 1 (Slot-Separable Update)**.** A state update is *slot-separable* if for every i:

$$\boldsymbol{S}_t[i] = g_{t,i}\big(\boldsymbol{S}_{t-1}[i],\, \boldsymbol{x}_t\big). \tag{7}$$

For linear recurrences, slot separability has a clean algebraic characterization.

Proposition 1 (Slot Separability of a Linear Update)**.** Consider the linear update

$$\boldsymbol{S}_t \;=\; \boldsymbol{D}_t\boldsymbol{S}_{t-1}\boldsymbol{A}_t \;+\; \boldsymbol{U}_t, \tag{8}$$

where $\boldsymbol{U}_t$ is independent of $\boldsymbol{S}_{t-1}$. Then

$$\text{Update is slot-separable} \quad\Longleftrightarrow\quad \boldsymbol{D}_t \text{ is diagonal.} \tag{9}$$

Diagonality of $\boldsymbol{D}_t$ makes the transition row-wise, yielding M independent slots *(see Appendix A.2)*.

Equation (8), also introduced by Zhong et al. (2025), incorporates decays on both sides of the update. Slot separability constrains how slots update; a router specifies where new information is written.

Definition 2 (Router)**.** A *router* $\boldsymbol{r}_t = r(\boldsymbol{x}_t, t) \in \mathbb{R}^M$ distributes new information across slots, with $\boldsymbol{r}_t[i]$ specifying the write intensity for slot i. Routers may depend on the current input $\boldsymbol{x}_t$ (input-dependent routing) or only on time t (time-dependent routing).

3.2 The Spectrum of Routing Slot Memories

Definition 3 (RSM)**.** A *Routing Slot Memory* is a slot-separable linear update of the form:

$$\boldsymbol{S}_t = \underbrace{(\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}}_{\text{preserved memory}} + \underbrace{\boldsymbol{r}_t \odot \left(\boldsymbol{D}_t\, \boldsymbol{S}_{t-1}\, \boldsymbol{A}_t + \boldsymbol{U}_t\right)}_{\text{updated memory}}, \tag{10}$$

where $\odot$ broadcasts row-wise, $\boldsymbol{D}_t \in \mathbb{R}^{M\times M}$ is diagonal, $\boldsymbol{A}_t \in \mathbb{R}^{d\times d}$ mixes features, $\boldsymbol{U}_t \in \mathbb{R}^{M\times d}$ is the write content derived from $\boldsymbol{x}_t$, and $\boldsymbol{r}_t \in \mathbb{R}^M$ is the routing vector.

The semantics are clean: when $\boldsymbol{r}_t[i] = 0$, slot i is frozen—its content persists exactly. When $\boldsymbol{r}_t[i] = 1$, it is fully updated. Intermediate values interpolate. A slot written at time j retains its content until the router selects it again, enabling long-lived storage with targeted updates.

3.3 Dual-State Models as RSMs

Dual-state models maintain separate key and value memories, $\boldsymbol{S}^k_t, \boldsymbol{S}^v_t \in \mathbb{R}^{M\times d}$, and differ from standard SSMs in that they apply *slot-wise* gating via $\boldsymbol{r}_t$ rather than channel-wise decay. The two main instantiations differ in a single design choice: whether the router is input-dependent.

SWA. Sliding Window Attention (Beltagy et al., 2020) is a maximally sparse RSM with the KV-cache recurrence in Equation (3), where $\boldsymbol{e}_t$ is the $(t \bmod M)$-th basis vector. The router is time-dependent and one-hot: exactly one slot is overwritten per step, with no decay:

$$\boldsymbol{A}_t = \boldsymbol{I}, \quad \boldsymbol{D}_t = \mathbf{0}, \quad \boldsymbol{r}_t = \boldsymbol{e}_t, \quad \boldsymbol{U}_t = [\boldsymbol{k}_t, \boldsymbol{v}_t]$$

The SWA router performs hard deletion, since the oldest slot is unconditionally erased. The write content is the stacked key and value vector $[\boldsymbol{k}_t, \boldsymbol{v}_t] \in \mathbb{R}^{2d}$.

GSA. Gated Slot Attention (Zhang et al., 2024) replaces SWA's fixed cyclic router with an input-dependent one:

$$\boldsymbol{S}^k_t = (\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}^k_{t-1} + \boldsymbol{r}_t\, \boldsymbol{k}_t^\top, \qquad \boldsymbol{S}^v_t = (\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}^v_{t-1} + \boldsymbol{r}_t\, \boldsymbol{v}_t^\top, \tag{11}$$

with $\boldsymbol{r}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/\tau}$. Input-dependent routing controls each slot's update strength, but typically dense writes limit protection from interference. Its RSM parameters are

$$\boldsymbol{A}_t = \boldsymbol{I}, \quad \boldsymbol{D}_t = \mathbf{0}, \quad \boldsymbol{r}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/\tau}, \quad \boldsymbol{U}_t = [\boldsymbol{k}_t, \boldsymbol{v}_t]$$

with τ being a temperature parameter for the decay. One of the main differences between GSA and ABC in the RSM view is the parameter $\boldsymbol{D}_t$, which removes state decay in ABC.

3.4 State-Space Models as RSMs

Standard diagonal SSMs are single-state models that are slot-separable and correspond to **dense routing**: $\boldsymbol{r}_t = \mathbf{1}_M$ at every step. Every slot is updated unconditionally; the only mechanism for selective retention is decay. Mamba-1 (Gu and Dao, 2024), Mamba-2 (Dao and Gu, 2024), and GLA (Yang et al., 2023) follow this template, differing only in the granularity of their decay:

$$\text{Mamba-2:} \quad \boldsymbol{S}_t[i] = a_t\, \boldsymbol{S}_{t-1}[i] + \boldsymbol{v}_t[i]\, \boldsymbol{k}_t^\top \tag{12}$$
$$\boldsymbol{D}_t = \boldsymbol{I}, \quad \boldsymbol{A}_t = a_t \quad (\textit{shared scalar} \text{ decay}),$$
$$\text{GLA/Mamba-1:} \quad \boldsymbol{S}_t[i] = \boldsymbol{S}_{t-1}[i] \odot \boldsymbol{a}_t + \boldsymbol{v}_t[i]\, \boldsymbol{k}_t^\top \tag{13}$$
$$\boldsymbol{D}_t = \boldsymbol{I}, \quad \boldsymbol{A}_t = \mathrm{diag}(\boldsymbol{a}_t) \quad (\textit{channel-wise} \text{ decay}).$$

Table 1: **Routing Slot Memories.** *Router* and *Decay* components across architectures. Row colors indicate router sparsity, with SWA and SSM marking the extremes. For SSMs, the content matrix ($\boldsymbol{u}_t$) is ***low-rank***, whereas SWA-like models use a ***sparse*** stacking of $\boldsymbol{k}_t$ and $\boldsymbol{v}_t$. $f(\cdot)$ denotes the softmax.

Model	Router ($\boldsymbol{r}_t$)	Content ($\boldsymbol{U}_t$)	Decay ($\boldsymbol{A}_t$)	Readout ($\boldsymbol{o}_t$)
LinAtt (Katharopoulos et al., 2020)	$\mathbf{1}_M$	$\boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{I}$	$\boldsymbol{S}_t\boldsymbol{q}_t$
RetNet (Sun et al., 2023)	$\mathbf{1}_M$	$\boldsymbol{v}_t\boldsymbol{k}_t^\top$	γ	$\boldsymbol{S}_t\boldsymbol{q}_t$
GLA (Yang et al., 2023)	$\mathbf{1}_M$	$\boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\mathrm{diag}\left(\sigma(\boldsymbol{W}\boldsymbol{x}_t)\right)^{1/\tau}$	$\boldsymbol{S}_t\boldsymbol{q}_t$
Mamba-2 (Dao and Gu, 2024)	$\mathbf{1}_M$	$\boldsymbol{v}_t\boldsymbol{k}_t^\top$	a_t	$\boldsymbol{S}_t\boldsymbol{q}_t$
GDN (Yang et al., 2024a)	$\mathbf{1}_M$	$\boldsymbol{v}_t\boldsymbol{k}_t^\top$	$a_t(\boldsymbol{I}-\boldsymbol{k}_t\boldsymbol{k}_t^\top)$	$\boldsymbol{S}_t\boldsymbol{q}_t$
Raven	$\boldsymbol{g}_t/\mathbf{1}^\top\boldsymbol{g}_t$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f(\boldsymbol{S}_t^k\boldsymbol{q}_t)$
GSA (Zhang et al., 2024)	$\mathbf{1}_M-\sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/\tau}$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f(\boldsymbol{S}_t^k\boldsymbol{q}_t)$
ABC (Peng et al., 2021)	$\mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t)$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f(\boldsymbol{S}_t^k\boldsymbol{q}_t)$
SWA (Beltagy et al., 2020)	$\boldsymbol{e}_t$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f(\boldsymbol{S}_t^k\boldsymbol{q}_t)$

Transposing the hidden state $\boldsymbol{S}_t^\top$ and multiplying from the opposite side yields the equivalent formulation used in other models.

Delta Networks (Yang et al., 2024b) also use dense routing ($\boldsymbol{r}_t=\mathbf{1}_M$) but replace scalar or diagonal decay with a rank-one forget gate:

$$\boldsymbol{S}_t=\boldsymbol{S}_{t-1}(\boldsymbol{I}-\beta_t\,\boldsymbol{k}_t\boldsymbol{k}_t^\top)+\beta_t\,\boldsymbol{v}_t\boldsymbol{k}_t^\top. \tag{14}$$

Since the transition acts by right-multiplication ($\boldsymbol{D}_t=\boldsymbol{I}$), rows do not mix and slots remain separable. Table 1 summarizes how all models fit within the RSM framework.

The routing/forgetting tradeoff. SSMs write densely and rely on decay for retention; SWA writes sparsely but discards information by position. GSA and ABC (Peng et al., 2021) route by content but still write densely. Raven combines the missing properties: **sparse**, **input-dependent** routing with **explicit decay** (Section 4).

4 Raven: Persistent Memory with Sparse Routing

Raven maintains separate key/value states with sparse, input-dependent writes and explicit decay. Relative to SWA, it replaces fixed-window allocation with content-dependent routing and RoPE-based position handling with decay (Afzal, 2026); relative to dense SSMs, it preserves unselected slots.

4.1 Raven Memory Update

Recurrence. Building on the slot-memory view of ABC and GSA (Peng et al., 2021; Zhang et al., 2024), our routed slot update lets different slots update at different times:

$$\begin{aligned}\boldsymbol{S}_t^k&=\exp(a_t\boldsymbol{r}_t)\odot\boldsymbol{S}_{t-1}^k+\big(\mathbf{1}-\exp(a_t\boldsymbol{r}_t)\big)\boldsymbol{k}_t^\top,\\ \boldsymbol{S}_t^v&=\exp(a_t\boldsymbol{r}_t)\odot\boldsymbol{S}_{t-1}^v+\big(\mathbf{1}-\exp(a_t\boldsymbol{r}_t)\big)\boldsymbol{v}_t^\top,\end{aligned} \tag{15}$$

with readout $\boldsymbol{o}_t=(\boldsymbol{S}_t^v)^\top\,\mathrm{softmax}\big(\boldsymbol{S}_t^k\boldsymbol{q}_t\big)$, where $\odot$ broadcasts over the feature dimension d. When $\boldsymbol{r}_t[i]$ is large, slot i is *decayed and overwritten*.

Decay. The scalar $a_t<0$ controls the rate of forgetting for the slots that are written to. Raven adopts Mamba–2's input-dependent per-head scalar decay in logarithmic scale (Dao and Gu, 2024):

$$a_t=-\mathrm{SoftPlus}(\boldsymbol{w}^\top\boldsymbol{x}_t)\,\exp(\Delta). \tag{16}$$

Here, Δ is a learnable scalar, as in Mamba-2. Since a_t enters the recurrence only through $\exp(a_t\boldsymbol{r}_t)$, unselected slots ($\boldsymbol{r}_t[i]=0$) experience no decay regardless of a_t; their content is frozen until the router selects them.

Router. The router determines which slots receive writes at each step. It must be sparse to preserve unselected slots and input-dependent to route by content rather than position. Raven achieves both

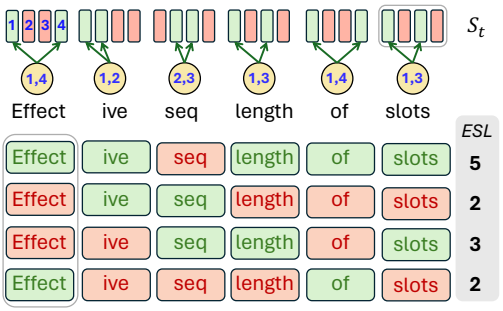


Figure 2: **Effective Sequence Length Visualization.** For $M = 4$ memory slots and $T = 6$ tokens, each slot processes a different *effective sequence length* (***ESL***), depending on the router.

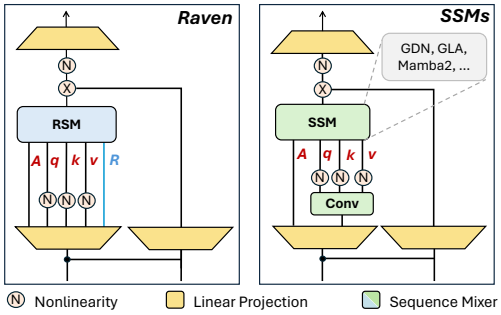


Figure 3: **Raven Neural Architecture.** ***(Left)*** Raven block vs. ***(Right)*** other linear models. Raven requires no short-range convolutions.

by adapting the DeepSeek MoE routing strategy (Dai et al., 2024; Liu et al., 2024): raw scores $\boldsymbol{m}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t) \in \mathbb{R}^M$ are sparsified by keeping only the top-K entries, then normalized:

$$\boldsymbol{g}_t = \mathrm{KeepTop}_K(\boldsymbol{m}_t) = \begin{cases} \boldsymbol{m}_t[i], & \text{if } i \in \mathrm{TopK}(\boldsymbol{m}_t), \\ 0, & \text{otherwise} \end{cases}, \qquad \boldsymbol{r}_t = \frac{\boldsymbol{g}_t}{\alpha \sum_{i=1}^{M} \boldsymbol{g}_t[i]}. \tag{17}$$

Here, $\alpha \in \mathbb{R}$ normalizes the router as the training sequence length grows (similar to τ for GLA). We choose $\alpha = 1$ for the 400M model and $\alpha = 4$ for the 800M model. This yields routing weights that sum to one over at most K selected slots. Routing design choices are ablated in Section 6.2.

Raven omits the MoE load-balancing loss (Shazeer et al., 2017): retrieval-critical tokens occupy dedicated slots, while general content shares others. Uneven allocation induces different effective sequence lengths across slots (Figure 2; analysis in Appendix C). This slot specialization (Figure 5) mirrors the concentration of Transformer recall in a few heads (Bick et al., 2025c).

4.2 Raven Block Design

Channel Mixer. Raven uses a Gated MLP (Liu et al., 2021), widely adopted in both linear (Yang et al., 2024a,b) and softmax (Qwen, 2025) transformers at medium scale (<10B). **Nonlinearities and Normalization.** Raven uses SiLU activations (as in Mamba-1 and Gated DeltaNet), followed by optional QK-RMSNorm (Henry et al., 2020). **No Short Convolutions.** Many SSMs and linear transformers convolve queries, keys, and values (Gu and Dao, 2024; von Oswald et al., 2025; Yang et al., 2024a). Raven drops these entirely, simplifying the block without sacrificing performance.

4.3 Unifying Sparse and Selective Writes

Despite their differences in decay, readout, and state structure, SWA and DeltaNet can both be written as:

$$\boldsymbol{S}_t \;=\; \boldsymbol{S}_{t-1}(\boldsymbol{I} - \boldsymbol{r}_t\boldsymbol{r}_t^\top) + \boldsymbol{v}_t\boldsymbol{r}_t^\top.$$

SWA instantiates this with a one-hot write vector $\boldsymbol{r}_t = \boldsymbol{e}_t$ (sparse, full overwrite of a single slot); DeltaNet uses $\boldsymbol{r}_t = \boldsymbol{k}_t$ (dense, distributed update across all slots). Raven uses a diagonal overwrite matrix $\boldsymbol{P}_t$:

$$\boldsymbol{S}_t \;=\; \boldsymbol{S}_{t-1}\,(\boldsymbol{I} - \boldsymbol{P}_t) + \boldsymbol{v}_t\,\boldsymbol{p}_t^\top, \tag{18}$$

with the following instantiations:

$$\begin{aligned} \textbf{SWA:} \quad & \boldsymbol{p}_t = \boldsymbol{e}_t, && \boldsymbol{P}_t = \boldsymbol{e}_t\boldsymbol{e}_t^\top, \\ \textbf{Raven:} \quad & \boldsymbol{p}_t = \mathbf{1} - \exp(a_t\boldsymbol{r}_t), && \boldsymbol{P}_t = \mathrm{diag}(\boldsymbol{p}_t), \\ \textbf{DeltaNet:} \quad & \boldsymbol{p}_t = \boldsymbol{k}_t, && \boldsymbol{P}_t = \boldsymbol{k}_t\boldsymbol{k}_t^\top. \end{aligned}$$

Raven occupies the middle ground: $\boldsymbol{P}_t$ is diagonal (not rank-one), sparse (at most K nonzero entries), and input-dependent. This enables slot-wise, selective overwrites—recovering SWA's sparsity and SSMs' decay within a single unified update.

Table 2: **In-context recall benchmarks and NIAH accuracy vs. context length and cache size.** Accuracy (%) on SWDE/FDA/SQuAD and single NIAH-1/2/3 across context lengths. Rec. mem. and Conv. mem. denote millions of cached state elements during decoding. The common baseline has 12.5M recurrent elements (approximately 0.52M per layer for 24-layer models); Mamba-2 distributes the same total across 48 layers.

Model	# Params (M)	(Rec / Conv) Mem. (M elts)	Recall tasks SWDE	FDA	SQuAD	NIAH-1 1K	2K	4K	8K	16K	32K	NIAH-2 1K	2K	4K	8K	16K	32K	NIAH-3 1K	2K	4K	8K	16K	32K
~400M / 15B tokens																							
Transformer																							
w. RoPE	340	∞ / 0.0	42.3	34.5	22.1	**100**	**100**	0.0	0.0	0.0	0.0	**100**	**100**	0.0	0.0	0.0	0.0	71.6	47.6	0.0	0.0	0.0	0.0
w. Gate (FoX)	376	∞ / 0.0	**52.5**	**64.3**	**30.1**	**100**	**100**	**32.2**	**8.0**	**4.2**	0.0	**100**	**100**	**100**	**24.0**	**11.6**	**3.2**	**95.4**	**85.6**	**64.2**	**11.6**	**7.2**	0.0
SSM																							
GLA	475	12.5 / 0.4	29.0	11.4	30.3	74.6	25.1	8.2	2.2	0.0	0.0	91.2	37.2	21.4	3.6	0.0	0.0	84.2	57.1	20.8	**10.2**	2.3	0.0
GSA	399	12.5 / **0.0**	23.8	14.5	24.9	99.2	97.1	90.0	67.4	29.6	11.0	96.6	**98.8**	28.0	5.1	1.0	0.0	60.0	30.1	13.5	1.0	0.0	0.0
GDN	475	12.5 / 0.4	29.5	8.3	31.3	99.2	**100**	**99.8**	92.0	41.8	22.1	99.2	92.0	43.6	17.8	6.2	4.0	**92.6**	**80.6**	**37.8**	5.2	**6.8**	2.5
Mamba-2	382	12.5 / 0.4	25.7	14.9	31.9	99.2	95.6	52.2	12.8	5.4	2.8	**99.8**	98.0	68.2	15.4	4.4	3.8	53.4	53.6	17.4	1.8	2.2	**3.2**
SWA	374	12.5 / **0.0**	10.0	14.4	29.7	29.8	11.0	6.2	3.4	1.2	0.0	36.2	14.4	10.2	3.8	3.2	0.0	26.2	9.2	7.4	1.4	1.8	0.0
Raven	424	12.5 / **0.0**	**34.1**	**22.7**	**35.4**	**99.8**	**100**	**99.8**	**99.8**	**99.4**	**91.4**	98.8	98.0	**98.8**	**81.6**	**23.0**	**8.8**	76.8	43.6	13.4	1.0	0.0	0.0
~800M / 32B tokens																							
Transformer						2K	4K	8K	16K	32K	64K	2K	4K	8K	16K	32K	64K	2K	4K	8K	16K	32K	64K
w. RoPE	693	∞ / 0.0	58.9	63.6	41.3	92.4	90.8	0.0	0.0	0.0	0.0	99.6	99.2	0.0	0.0	0.0	0.0	33.4	9.0	0.0	0.0	0.0	0.0
w. Gate (FoX)	694	∞ / 0.0	**64.9**	**80.0**	**41.5**	**100**	**100**	**99.0**	**28.6**	**12.4**	0.0	**100**	**100**	**67.4**	**9.0**	**7.8**	0.0	**83.2**	**63.8**	**0.2**	**0.4**	**0.2**	0.0
SSM																							
GLA	892	16.5 / 0.4	**50.1**	**35.9**	39.2	100	94.2	29.4	3.6	0.0	0.0	100	89.8	25.8	3.8	3.0	0.0	94.6	51.2	3.2	1.8	1.6	0.0
GSA	750	16.5 / **0.0**	42.8	28.9	33.6	100	100	99.8	97.6	64.0	0.0	99.4	90.0	29.8	3.6	2.4	0.0	79.0	16.6	2.4	0.2	0.0	0.0
GDN	892	16.5 / 0.4	46.0	31.9	35.9	**100**	**100**	**100**	**100**	94.8	45.2	**100**	95.4	35.2	14.2	**9.2**	**5.8**	79.4	**65.4**	10.6	6.4	**4.2**	**2.0**
Mamba-2	711	16.5 / 0.6	42.7	21.3	34.6	99.8	63.4	13.6	1.8	0.8	0.6	97.4	49.2	24.2	6.6	4.2	3.2	89.8	17.4	**14.6**	1.4	2.4	0.6
SWA	693	16.5 / **0.0**	14.9	12.2	28.8	11.0	6.2	3.4	1.2	0.6	0.0	14.4	6.4	5.0	1.2	2.4	0.0	15.2	4.8	1.4	2.4	3.4	0.0
Raven	792	16.5 / **0.0**	41.7	24.1	**39.9**	99.8	**100**	99.8	99.2	**98.2**	**91.0**	**100**	**99.4**	**90.8**	**35.2**	6.8	3.4	**90.4**	5.2	0.6	0.0	0.0	0.0

5 Empirical Validation

We evaluate Raven across three axes: retrieval ability, general language modeling, and hybrid performance.

Experimental Setup. We follow the standard training recipe, matching memory size and parameter counts across linear and quadratic baselines. Full training details and model configurations are provided in Appendix A.3. Benchmark definitions are provided in Appendix A.3.1.

5.1 Retrieval Abilities

Single Needle-in-a-Haystack (NIAH). We evaluated passkey retrieval across varying depths and sequence lengths (Hsieh et al., 2024), with 400M and 800M models trained at context lengths of 2048 and 4096 tokens, respectively. In Table 2, Mamba-2 and GDN degrade significantly beyond 8K tokens at 400M—already $4\times$ their training length. Raven maintains near-perfect accuracy ($\geq \mathbf{99}\%$) up to 16K tokens and is the *only model* at 400M to retain strong performance ($> \mathbf{91}\%$) at 32K, $\mathbf{16}\times$ its training length. Sparse routing limits memory interference, enabling extrapolation without an explicit length curriculum. At 800M, Raven retains $\mathbf{91.0}\%$ on NIAH-1 at 64K versus 45.2% for the next-best baseline (GDN), and surpasses FoX (Lin et al., 2025) on NIAH-1 (Figure 4).

Recall-Intensive Benchmarks. On SQuAD, SWDE, and FDA (Table 2), Raven consistently outperforms linear-time baselines at 400M. SWDE accuracy reaches **34.1%**, versus Mamba-2's 25.7% and GLA's 29.0%, narrowing the gap to Transformers. On FDA, Raven alone among linear models exceeds 22%, improving by $\sim$8 points over the best SSM baseline. At 800M, it remains competitive with GDN. Full-attention Transformers such as FoX retain all tokens rather than a fixed-size memory; Section 5.3 compares them with hybrids replacing half the softmax layers with linear layers.

Retrieval Without Convolutions. GLA, Mamba-2, and GDN use query/key/value convolutions for local integration and training stability (Gu and Dao, 2024; von Oswald et al., 2025; Yang et al., 2024a). Raven achieves strong retrieval without them, suggesting that input-dependent routing and per-slot decay suffice to address the recall bottleneck.

Table 3: **Zero-shot language modeling performance across models.** Left: 400M models trained on 15B tokens. Right: 800M models trained on 32B tokens.

400M / 15B tokens

Model	# Params (M)	LMB. ppl↓	LMB. acc↑	PIQA acc↑	Hella. acc↑	Wino. acc↑	ARC-e acc↑	ARC-c acc↑	Avg. acc↑
Transformer									
w. RoPE	340	42.0	31.0	64.4	30.2	51.0	44.3	18.7	39.9
w. Gate (FoX)	376	48.1	30.6	64.9	30.7	51.1	44.7	18.9	40.1
SSM									
GLA	475	42.1	30.7	64.4	30.1	52.7	43.8	19.6	40.2
GSA	399	44.1	30.3	64.9	30.7	51.5	45.6	20.5	40.5
GDN	475	40.1	31.6	65.6	31.4	50.2	45.7	19.3	**40.6**
Mamba-2	382	43.0	29.9	65.0	31.5	51.2	47.5	20.5	40.1
SWA	374	40.7	30.5	64.5	30.4	51.6	44.9	18.6	40.0
Raven	424	41.0	32.7	64.1	30.3	51.7	43.9	18.4	40.2

800M / 32B tokens

Model	# Params (M)	LMB. ppl↓	LMB. acc↑	PIQA acc↑	Hella. acc↑	Wino. acc↑	ARC-e acc↑	ARC-c acc↑	Avg. acc↑
Transformer									
w. RoPE	693	18.6	41.4	66.3	34.3	52.2	49.9	21.9	44.5
w. Gate (FoX)	694	25.3	38.2	67.8	34.4	51.7	49.9	21.5	44.1
SSM									
GLA	892	23.6	38.2	66.9	33.4	52.4	48.5	21.2	43.5
GSA	750	27.4	34.7	66.3	32.1	51.6	46.6	19.6	41.8
GDN	892	21.3	39.4	68.1	35.2	53.0	52.5	22.1	**45.1**
Mamba-2	712	24.6	36.0	68.1	35.4	52.6	52.3	22.3	44.5
SWA	693	20.8	38.8	67.9	34.2	52.8	49.3	21.2	44.0
Raven	792	26.0	38.2	67.0	33.2	50.9	49.2	21.0	43.3

Table 4: **Hybrid Models Retrieval Ability.** GDN, Mamba-2, SWA-RoPE, and Raven as linear components in full-attention hybrids with NoPE positional embeddings. Performance on NIAH-1/2/3 and highlighted SWDE, FDA, and SQuAD. The 400M and 800M models are trained on 2K and 4K sequences, respectively. Full ablations appear in Table 9.

Linear Model	No Conv.	SWDE acc↑	FDA acc↑	SQuAD acc↑	NIAH-1 1K	NIAH-1 2K	NIAH-1 4K	NIAH-1 8K	NIAH-1 16K	NIAH-1 32K	NIAH-2 1K	NIAH-2 2K	NIAH-2 4K	NIAH-2 8K	NIAH-2 16K	NIAH-2 32K	NIAH-3 1K	NIAH-3 2K	NIAH-3 4K	NIAH-3 8K	NIAH-3 16K	NIAH-3 32K
~ 400M / 15B tokens																						
GDN	✗	54.6	67.2	34.5	**100**	**100**	**100**	**100**	93.2	70.5	**100**	**100**	**100**	8.0	0.0	0.0	93.2	70.2	50.0	0.0	0.0	0.0
Mamba-2	✗	**56.3**	**68.8**	**36.0**	**100**	**100**	16.4	0.0	0.0	0.0	**100**	**100**	85.8	0.0	0.0	0.0	76.9	**80.6**	60.8	0.0	0.0	0.0
SWA-RoPE	✔	51.0	68.1	34.1	**100**	**100**	**100**	**100**	98.2	60.4	**100**	**100**	**100**	98.2	3.1	0.0	**93.4**	78.2	12.8	**60.0**	4.4	0.0
Raven	✔	51.4	64.2	31.4	**100**	**100**	**100**	**100**	98.4	**78.6**	**100**	**100**	**100**	**100**	**95.4**	**65.4**	90.0	67.0	**73.8**	**60.0**	**10.2**	**14.4**
~ 800M / 32B tokens					2K	4K	8K	16K	32K	64K	2K	4K	8K	16K	32K	64K	2K	4K	8K	16K	32K	64K
GDN	✗	64.7	77.8	**48.1**	**100**	**100**	**100**	63.4	0.2	0.0	**100**	**100**	**99.0**	2.2	0.0	0.0	96.8	93.8	76.2	0.0	0.0	0.0
Mamba-2	✗	**68.5**	72.9	42.3	**100**	**100**	0.0	0.0	0.0	0.0	**100**	**100**	5.6	0.0	0.0	0.0	88.8	87.4	5.8	0.0	0.0	0.0
SWA-RoPE	✔	63.4	68.5	17.2	**100**	**100**	**100**	**100**	97.6	**69.0**	**100**	**100**	98.4	3.0	0.0	0.0	67.8	36.0	17.6	6.2	0.0	0.0
Raven	✔	64.5	**81.3**	37.0	**100**	**100**	**100**	99.8	**99.4**	55.4	**100**	**100**	**100**	**99.8**	**84.6**	**80.8**	**97.2**	**93.6**	**85.4**	**59.2**	**36.8**	0.0

5.2 Language Modeling

Table 3 compares zero-shot language modeling at 400M and 800M parameters. At 400M parameters, Raven matches or exceeds Mamba-2 and GLA on average accuracy while achieving the best Lambada accuracy among all models, including Transformers. At 800M, Raven trails GDN on average accuracy, though GDN uses significantly more parameters (892M vs. 792M). Across both scales, sparse routing and input-dependent forgetting do not measurably degrade general language modeling.

5.3 Hybrid Raven

We evaluate Raven as a drop-in linear component paired with two full NoPE attention layers. We also ablate RoPE positional embeddings and SWA instead of full attention, evaluating four configurations in Table 9.

Hybrid Raven vs. SSMs When paired with full NoPE attention, Raven dramatically outperforms GDN and Mamba-2 hybrids on long-context NIAH. GDN+Attn degrades sharply on NIAH-2 and NIAH-3 beyond 4K, and Mamba-2+Attn collapses almost entirely past 2K on NIAH-1, whereas Raven +Attn maintains strong accuracy up to 32K on NIAH-1 and 16K on NIAH-2.

The two components play complementary roles: NoPE attention enables precise short-range retrieval, whereas Raven 's persistent slots retain long-range information across the sequence. While all other hybrid approaches fail to achieve meaningful accuracy on NIAH-2 for long sequences (32K and 16K), Raven attains over 80% accuracy on these tasks (Table 9).

Hybrid Raven vs. SWA-RoPE We also compare against hybrids combining SWA-RoPE with full NoPE attention. Raven replaces SWA's fixed ring buffer with input-dependent routing and RoPE-based position handling with decay (Section 4). As shown in Table 4, the Raven hybrid consistently outperforms SWA-RoPE on both recall-intensive benchmarks and NIAH. In particular, it substantially improves length generalization, alleviating the main limitation of SWA-based hybrids.

Table 5: **Architectural Components Effect.** Results use 256 memory slots and a sigmoid Top_{32} router, with 400M parameters trained on 15B tokens. **Lmb.**: evaluation Lambada perplexity; **Avg.**: average performance on the language modeling tasks in Table 3.

Model Ablation	Lmb. ppl ↓	Avg. acc ↑	SWDE acc ↑	FDA acc ↑	SQuAD acc ↑
Raven	43.1	**40.1**	31.5	**19.6**	**36.6**
w/o. Output Gate	48.2	39.1	29.6	14.6	30.2
w. $\boldsymbol{A}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/4}$	43.2	39.5	28.0	11.5	29.4
w. Short Conv	**42.7**	40.1	**32.2**	16.1	36.4

Table 6: **Router Design Ablations.** Recall-heavy tasks for 400M models with 256 slots and Top_{32}. Blue highlights the default Raven setup.

Router Type	Gumbel Noise	Router Projection	SWDE acc↑	FDA acc↑	SQuAD acc↑
Sigmoid	✔	Linear	**34.9**	13.8	**34.3**
Sigmoid	✗	Linear	29.7	17.7	29.5
softmax	✔	Linear	25.4	18.2	27.6
softmax	✗	Linear	26.2	9.4	25.6
Sigmoid	✔	MLP	30.0	**25.7**	30.7
Sigmoid	✗	MLP	30.0	13.3	20.5
softmax	✔	MLP	22.7	21.3	29.2
softmax	✗	MLP	20.8	4.5	24.0

6 Ablations

We ablate the key design choices in Raven across four axes: architectural components, router design, memory shape, and routing sparsity.

6.1 Block Components

We ablate three architectural components common in linear and softmax transformers: output gating, decay parametrization, and short-range convolutions. Results are summarized in Table 5.

Output gating improves retrieval, as in prior models (Dao and Gu, 2024; Yang et al., 2024b) (Figure 3). The GLA/KDA temperature-scaled sigmoid decay (Team et al., 2025) underperforms Raven's Mamba-2-style scalar decay. Query/key/value convolutions yield marginal gains on some benchmarks; we omit them for simplicity, retaining strong recall without the convolutions used by prior linear models (Jianlin, 2025; Yang et al., 2024b).

6.2 Router Design

We ablate the scoring function, stochastic exploration during training, and projection expressiveness on recall-heavy benchmarks (Table 6).

6.2.1 Softmax vs. Sigmoid Routing

The two standard choices in MoE architectures are softmax and sigmoid. **Softmax** routing normalizes scores globally across all M slots before selection: $\boldsymbol{m}_t = \text{softmax}(\boldsymbol{W}\boldsymbol{x}_t) \in \mathbb{R}^M$. Global normalization means that the scores of unselected slots influence the selected weights, coupling routing decisions across the full slot set.

Sigmoid routing (Equation (17)) scores each slot independently (Dai et al., 2024). Sigmoid routing outperforms softmax across most tasks (Table 6), and we adopt it as the default.

6.2.2 Gumbel Noise During Training

Routers can collapse onto a small subset of slots early in training (Shazeer et al., 2017), leaving unselected slots uninformative and reducing effective memory capacity. To prevent this, we inject Gumbel noise (Gumbel, 1954; Jang et al., 2016) into the router logits during training:

$$\boldsymbol{m}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \mathbf{n}), \qquad \mathbf{n} \in \mathbb{R}^M \sim \text{Gumbel}(\mathbf{0}, \mathbf{1}), \tag{19}$$

where $f(\cdot)$ is sigmoid or softmax. The noise encourages exploration of all M slots during training and is removed at inference for deterministic routing. Table 6 shows consistent gains across configurations.

6.2.3 Linear vs. MLP Projection

The standard routing projection is a single linear layer $\boldsymbol{W}\boldsymbol{x}_t$. Following Anthony et al. (2025), who showed that MLP projections yield richer router representations in MoEs, we ablate an MLP variant: $\boldsymbol{m}_t = f(\boldsymbol{W}_2\,\text{GELU}(\boldsymbol{W}_1\boldsymbol{x}_t + \boldsymbol{b}_1))$. The MLP router improves on some benchmarks (notably FDA)

Table 7: **Memory Shape and Top$_K$ Ablations.** Accuracy vs. #memory slots (fixed budget) and the Top$_K$ write budget on gray-highlighted SWDE/FDA/SQuAD and NIAH-1/2/3 (1K–32K). More slots trade per-slot capacity (`head_dim`) for token-wise storage. Larger Top$_K$ improves long-context retrieval; when Top$_K$ $= K$, all slots are written with router-scaled intensity.

Mem Slot # Num.	Top$_K$	SWDE acc↑	FDA acc↑	SQuAD acc↑	NIAH-1 1K	NIAH-1 2K	NIAH-1 4K	NIAH-1 8K	NIAH-1 16K	NIAH-1 32K	NIAH-2 1K	NIAH-2 2K	NIAH-2 4K	NIAH-2 8K	NIAH-2 16K	NIAH-2 32K	NIAH-3 1K	NIAH-3 2K	NIAH-3 4K	NIAH-3 8K	NIAH-3 16K	NIAH-3 32K
128	16	34.5	21.8	**38.7**	99.2	99.0	98.6	99.0	97.8	**98.8**	98.6	99.0	87.6	8.0	1.0	2.0	66.0	44.4	13.6	0.0	0.0	0.0
128	32	**35.6**	16.2	38.5	99.6	**100**	**100**	**100**	**99.4**	98.2	99.6	**99.8**	92.6	41.8	11.8	8.6	78.4	42.0	2.4	3.4	0.4	**2.8**
128	64	34.8	19.4	36.0	**99.8**	99.6	99.0	98.4	96.0	73.6	99.2	97.0	93.0	42.4	6.0	7.0	**80.8**	**54.4**	4.4	2.8	**0.8**	0.6
256	32	31.5	19.6	36.6	99.0	98.8	99.0	99.2	98.8	96.8	98.4	98.4	95.0	18.4	1.6	2.4	69.0	40.4	11.8	1.2	0.0	0.0
256	64	32.0	**22.9**	33.3	98.2	98.4	96.8	97.4	97.8	96.8	97.2	98.0	95.4	21.0	3.6	1.8	70.0	47.8	6.0	0.8	0.0	0.0
256	128	34.1	22.7	35.4	**99.8**	**100**	99.4	99.8	**99.4**	91.4	98.8	98.0	**98.8**	**81.6**	**23.0**	**8.8**	76.8	43.6	13.4	1.0	0.2	0.0
256	256	33.4	19.6	32.7	99.2	99.8	98.4	97.4	97.8	81.0	**99.8**	98.6	94.6	47.8	3.8	3.6	53.6	9.4	11.8	3.6	0.6	1.0
512	64	28.4	14.9	7.5	99.2	99.0	97.4	97.0	90.4	65.4	99.2	99.2	73.8	18.2	2.6	2.4	56.8	40.2	10.8	1.0	0.0	0.0
512	256	29.9	15.1	3.8	94.6	92.6	87.2	79.0	67.6	34.6	95.6	84.8	3.8	1.4	0.0	2.0	30.0	4.2	0.0	0.0	0.0	0.0
512	512	34.5	22.0	33.9	99.6	**100**	98.8	92.2	63.8	77.2	99.4	99.2	90.6	50.2	3.8	4.6	75.0	52.8	**20.6**	**6.8**	0.2	0.6

but underperforms the linear router on others, at the cost of additional parameters. Given the inconsistent gains, we select the best-performing configuration per Table 6 for the final Raven design.

6.3 Memory Shape: Head Dimension vs. Number of Slots

Under a fixed per-layer memory budget $M \times d \times H$ (H heads), more slots M require a smaller feature dimension d. Here, d controls per-slot representation capacity; M controls how many distinct items can be stored without interference.

Table 7 sweeps over budget-matched (M, d) configurations and consistently shows that allocating capacity toward more slots improves recall, even as d shrinks. This suggests that more slots reduce the probability that retrieval-critical tokens share slots with unrelated content and get overwritten. We fix $M = 256$ as the default, consistent with GLA, GDN, and GSA at both scales. Strong recall and length generalization hold across a wide range of memory shapes; most configurations outperform GDN and Mamba-2 on NIAH and other recall-intensive tasks.

6.4 Top-K Sparsity

The sparsity parameter K controls how many slots are updated at each step. Although sigmoid scores are naturally sparse, Top-K selection ensures that unselected slots remain exactly frozen rather than receiving negligible but nonzero updates.

Across memory shapes (Table 7), smaller K reduces interference and improves SWDE recall; larger K approaches dense SSM-like updates. We use $K = 32$ of $M = 256$ slots (12.5% occupancy), balancing long-lived storage in lightly used slots with flexibility to distribute content across multiple slots. The analysis of learned memory allocation is provided in Appendix C.

7 Limitations

Our experiments cover approximately 400M and 800M models trained on 15B and 32B tokens, respectively. Results on synthetic NIAH and selected extraction and question-answering benchmarks do not establish performance at larger scales or on arbitrary long-context tasks.

Fixed-size memory cannot preserve every token in an unbounded input. The memory-allocation visualizations show specialization on synthetic retrieval examples; whether it holds across tasks remains untested. Retrieval gains also do not imply uniform language-modeling improvements: at 800M, Raven trails GDN in average zero-shot accuracy (Table 3).

8 Conclusion

Routing Slot Memories (RSMs) formalize memory slot selection across linear sequence models. Raven combines sparse, input-dependent routing with explicit decay, decoupling *where* information is written from *how long* it persists.

Raven isolates retrieval-critical tokens in dedicated slots with variable effective sequence lengths and preserves unselected slots. It extrapolates beyond $16\times$ the training context without short-range

convolutions, outperforms dense SSMs on recall-intensive tasks, and remains competitive on general language modeling. These gains extend to full-attention and SWA hybrids, supporting routing and forgetting as distinct design axes.

Acknowledgments and Disclosure of Funding

No specific funding was received. The authors declare no competing interests.

References

Arshia Afzal. On the legacy of linear transformers in positional embedding. *preprint*, 2026. URL https://arshiaafzal.github.io/blog/2026/pe/.

Quentin Anthony, Yury Tokpanov, Skyler Szot, Srivatsan Rajagopal, Praneeth Medepalli, Rishi Iyer, Vasu Shyam, Anna Golubeva, Ansh Chaurasia, Xiao Yang, et al. Training foundation models on a full-stack amd platform: Compute, networking, and system design. *arXiv preprint arXiv:2511.17127*, 2025.

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, 2023. URL https://arxiv.org/abs/2312.04927.

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.

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

Aviv Bick, Tobias Katsch, Nimit Sohoni, Arjun Desai, and Albert Gu. Llamba: Scaling distilled recurrent models for efficient language processing. *arXiv preprint arXiv:2502.14458*, 2025a.

Aviv Bick, Kevin Y. Li, Eric P. Xing, J. Zico Kolter, and Albert Gu. Transformers to ssms: Distilling quadratic knowledge to subquadratic models, 2025b. URL https://arxiv.org/abs/2408.10189.

Aviv Bick, Eric Xing, and Albert Gu. Understanding the skill gap in recurrent language models: The role of the gather-and-aggregate mechanism, 2025c. URL https://arxiv.org/abs/2504.18574.

Aviv Bick, Eric P. Xing, and Albert Gu. Retrieval-aware distillation for transformer-ssm hybrids, 2026. URL https://arxiv.org/abs/2602.11374.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nvidia nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

Ricardo Buitrago Ruiz and Albert Gu. Understanding and improving length generalization in recurrent models. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=2OEb20dy7B.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2306.15595.

Yingfa Chen, Xinrong Zhang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. Stuffed mamba: Oversized states lead to the inability to forget, 2025. URL https://arxiv.org/abs/2410.07145.

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.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning*, 2024.

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.

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

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Learning and Modeling (COLM 2024)*, 2024.

Emil Julius Gumbel. Statistical theory of extreme values and some practical applications. *Applied Mathematics Series*, 33, 1954.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. *arXiv preprint arXiv:2010.04245*, 2020.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

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?, 2024. URL https://arxiv.org/abs/2404.06654.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying, 2024. URL https://arxiv.org/abs/2402.01032.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Su Jianlin. Why add short conv to linear attention?, 2025. URL https://kexue.fm/archives/11320.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moh: Multi-head attention as mixture-of-head attention. *arXiv preprint arXiv:2410.11842*, 2024.

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.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *The International Conference on Learning Representations (ICLR)*, 2015.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, volume 4, 1992.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Zhixuan Lin, Evgenii Nikishin, Xu Owen He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. *arXiv preprint arXiv:2503.02130*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. Pay attention to mlps. *Advances in neural information processing systems*, 34:9204–9215, 2021.

Peng Lu, Jerry Huang, Qiuhao Zeng, Xinyu Wang, Boxing Chen, Philippe Langlais, and Yufei Cui. Mamba modulation: On the length generalization of mamba. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2509.19633.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023.

Sajad Movahedi, Timur Carstensen, Arshia Afzal, Frank Hutter, Antonio Orvieto, and Volkan Cevher. Selective rotary position embedding. *arXiv preprint arXiv:2511.17388*, 2025.

Daniele Paliotta, Junxiong Wang, Matteo Pagliardini, Kevin Y. Li, Aviv Bick, J. Zico Kolter, Albert Gu, François Fleuret, and Tri Dao. Thinking slow, fast: Scaling inference compute with distilled reasoners, 2025. URL https://arxiv.org/abs/2502.20339.

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.

Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks, 2024. URL https://arxiv.org/abs/2402.04248.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.00071.

Hao Peng, Jungo Kasai, Nikolaos Pappas, Dani Yogatama, Zhaofeng Wu, Lingpeng Kong, Roy Schwartz, and Noah A Smith. Abc: Attention with bounded-memory control. *arXiv preprint arXiv:2110.02488*, 2021.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2108.12409.

Qwen. Qwen3-Next: Towards ultimate training & inference efficiency. https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd&from=research.latest-advancements-list, 2025. Accessed: 2025-11-03.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling, 2024. *URL https://arxiv.org/abs/2406.07522*, 2406:07522, 2024.

Ricardo Buitrago Ruiz and Albert Gu. Understanding and improving length generalization in recurrent models. *arXiv preprint arXiv:2507.02782*, 2025.

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.

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.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

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.

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.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

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

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. Mesanet: Sequence modeling by locally optimal test-time training. *arXiv preprint arXiv:2506.05233*, 2025.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, Garvit Kulshreshtha, Vartika Singh, Jared Casper, Jan Kautz, Mohammad Shoeybi, and Bryan Catanzaro. An empirical study of mamba-based language models, 2024. URL https://arxiv.org/abs/2406.07887.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models, 2025. URL https://arxiv.org/abs/2408.15237.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. Rnns are not transformers (yet): The key bottleneck on in-context retrieval, 2024. URL https://arxiv.org/abs/2402.18510.

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

Songlin Yang 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.

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

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

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024b.

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.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025a.

Yifan Zhang, Zhen Qin, and Quanquan Gu. Higher-order linear attention. *arXiv preprint arXiv:2510.27258*, 2025b. URL https://arxiv.org/abs/2510.27258.

Yu Zhang, Songlin Yang, Rui-Jie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *Advances in Neural Information Processing Systems*, 37:116870–116898, 2024.

Shu Zhong, Mingyu Xu, Tenglong Ao, and Guang Shi. Understanding transformer from the perspective of associative memory. *arXiv preprint arXiv:2505.19488*, 2025.

Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaiem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hakim Hacid. Falcon mamba: The first competitive attention-free 7b language model, 2024. URL https://arxiv.org/abs/2410.05355.

A Appendix

A.1 Notation

Table 8: **Notations.** Summary of notations used throughout the paper.

Definition	Notation
$\boldsymbol{S}_t$	Matrix-valued Hidden state
$\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$	Query, Key, Value vectors
$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	Query, Key, Value matrices
a_t	Scalar decay
$\boldsymbol{A}_t$	Matrix decay
$\boldsymbol{r}_t$	Router
$\sigma(.)$	Sigmoid function
$\phi(.)$	Non-linear function
$\odot$	Hadamard product

A.2 Slot Separability of Linear Updates

Proposition. Let $\boldsymbol{S}_{t-1} \in \mathbb{R}^{M\times d}$, $\boldsymbol{D}_t \in \mathbb{R}^{M\times M}$, $\boldsymbol{A}_t \in \mathbb{R}^{d\times d}$, and $\boldsymbol{U}_t \in \mathbb{R}^{M\times d}$, and consider

$$\boldsymbol{S}_t \;=\; \boldsymbol{D}_t \boldsymbol{S}_{t-1} \boldsymbol{A}_t \;+\; \boldsymbol{U}_t. \tag{20}$$

Denote by $\boldsymbol{S}_t[i] \in \mathbb{R}^{1\times d}$ the i-th row of $\boldsymbol{S}_t$. Then the map $\boldsymbol{S}_{t-1} \mapsto \boldsymbol{S}_t$ is *row-wise* (i.e., $\boldsymbol{S}_t[i]$ depends only on $\boldsymbol{S}_{t-1}[i]$) if and only if $\boldsymbol{D}_t$ is diagonal.

Proof. ($\Leftarrow$) If $\boldsymbol{D}_t = \mathrm{diag}(d_1, \ldots, d_M)$, then for each i,

$$\boldsymbol{S}_t[i] = d_i\, \boldsymbol{S}_{t-1}[i] \boldsymbol{A}_t + \boldsymbol{U}_t[i],$$

which depends only on $\boldsymbol{S}_{t-1}[i]$. Hence the update is row-wise and hence slot-separable.

($\Rightarrow$) Assume that the update is row-separable and $\boldsymbol{D}_t$ has a nonzero off-diagonal entry $\boldsymbol{D}_t[ik] \neq 0$ for some $k \neq i$. Choose $\Delta \in \mathbb{R}^{1\times d}$ such that $\Delta \boldsymbol{A}_t \neq \boldsymbol{0}$, which is possible because $\boldsymbol{A}_t \neq 0$. Now consider two states $\boldsymbol{S}_{t-1}$ and $\boldsymbol{S}'_{t-1}$ that differ only in row k, with

$$\boldsymbol{S}'_{t-1}[\ell] = \begin{cases} \boldsymbol{S}_{t-1}[k] + \Delta, & \ell = k, \\ \boldsymbol{S}_{t-1}[\ell], & \ell \neq k. \end{cases}$$

Then,

$$\boldsymbol{S}'_t[i] - \boldsymbol{S}_t[i] = \boldsymbol{D}_t[ik]\, \Delta \boldsymbol{A}_t \neq \boldsymbol{0}.$$

Thus, modifying only row $k \neq i$ changes $\boldsymbol{S}_t[i]$, contradicting row-separability. Therefore all off-diagonal entries of $\boldsymbol{D}_t$ must be zero, and hence for slot-separable update $\boldsymbol{D}_t$ must be diagonal. □

A.3 Experimental Details

A.3.1 Evaluation Benchmarks

Retrieval-Heavy Tasks. These benchmarks require locating and utilizing evidence present in the input context rather than relying on parametric knowledge. We further divide them into two subsets: (i) **Real-world retrieval** includes SQuAD (Rajpurkar et al., 2018), an extractive question-answering benchmark in which answers are spans within a supporting passage; SWDE (Arora et al., 2024), a structured web data extraction task over raw HTML pages with labeled attributes; and FDA (Arora et al., 2024), which requires extracting a fixed set of attributes from FDA 510(k) report PDFs (up to $\sim$20 pages). These tasks demand locating and, in some cases, aggregating evidence from long-form inputs. (ii) **Synthetic retrieval** uses RULER (Hsieh et al., 2024) with the S-NIAH-{1,2,3} tasks, which provide controlled settings in which a model must retrieve a key–value association embedded within a long "haystack." S-NIAH-1 uses a passkey-style setup with a repetitive background; S-NIAH-2 embeds the same key–value format within natural-text haystacks (e.g., essays); and S-NIAH-3 increases difficulty by requiring retrieval of longer values (UUIDs) from similar natural-text haystacks.

Language Modeling Tasks. These benchmarks place comparatively less weight on contextual retrieval, instead primarily testing factual knowledge and commonsense reasoning. We include PIQA (Bisk et al., 2019), Winogrande (Sakaguchi et al., 2019), HellaSwag (Zellers et al., 2019), and ARC (Challenge and Easy) (Clark et al., 2018).

A.3.2 Training Recipe

Our experiments follow the standard pipeline in the literature (Dao and Gu, 2024; Yang et al., 2023, 2024a,b), specifically that of Yang et al. (2024b). Models are trained at two scales: ∼400M and ∼800M parameters. All models are trained on SlimPajama-627B (Soboleva et al., 2023). The 400M models are trained for 15B tokens with a sequence length of 2048, while the 800M models are trained for 32B tokens with a sequence length of 4096. We use a batch size of 0.5M and the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 4×10^{-4} (Yang et al., 2024a). The learning rate follows a cosine scheduler with 1B tokens of warmup. We utilize the `flash-linear-attention`[3] repository for baselines as well as our codebase.

A.3.3 Models Configuration

Our experiments encompass several state-of-the-art linear models, including GLA, GSA, Mamba-2, GDN, and SWA. We follow their original configurations at both experimental scales. Importantly, to ensure a fair comparison, particularly on retrieval tasks, **we carefully match the overall memory elements** across all models. Since single-state SSMs such as GLA and GDN employ only a single recurrence in their original configurations (Yang et al., 2024a), we expand the value dimension of these models so that they share the same number of memory elements as other baselines.

All models use 24 layers, except for Mamba-2, which uses 48 layers as it does not rely on a channel mixer. For GLA, we include the short-range convolution, as it has been shown to significantly improve performance (Yang et al., 2024b); therefore, all our GLA baselines use short-range convolution. We also use the specialized Triton kernels provided by the `flash-linear-attention` repository for each model.

Transformer baselines all use 24 layers and 16 heads, following the LLaMA architecture (Dubey et al., 2024). We include three different positional encoding schemes, RoPE (Su et al., 2021), NoPE (Kazemnejad et al., 2023), and the Forgetting Transformer (FoX) (Lin et al., 2025), to ensure comprehensive evaluation. The number of memory slots for dual-state recurrences is set to 256.

A.3.4 NIAH Results

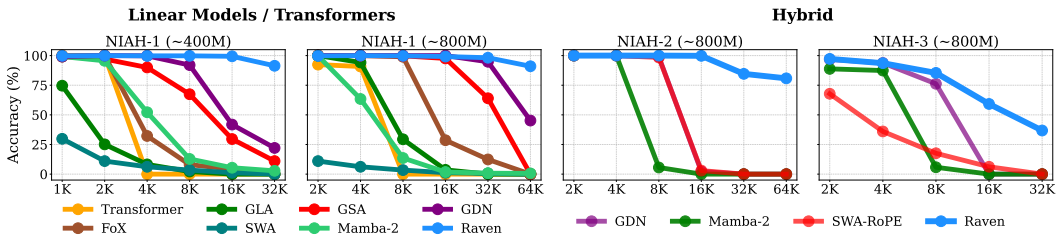


Figure 4: **NIAH Results.** ***(Left)*** NIAH-1 for 400M and 800M SSMs and transformers. ***(Right)*** NIAH-2 and NIAH-3 for 800M hybrids.

[3] https://github.com/fla-org/flash-linear-attention

A.4 Raven Recurrence

Raven Recurrence:

$$
\begin{aligned}
\boldsymbol{m}_t &= \sigma(\boldsymbol{W}\boldsymbol{x}_t), \quad \boldsymbol{g}_t = \begin{cases} \boldsymbol{m}_t[i] & i \in \mathrm{Top}_K(\boldsymbol{m}_t) \\ 0.0 & o.w \end{cases}, \\
\boldsymbol{r}_t &= \frac{\boldsymbol{g}_t}{\alpha \sum_{i=1}^{M} \boldsymbol{g}_t[i]}, \quad a_t = -\mathrm{SoftPlus}(\boldsymbol{w}^\top \boldsymbol{x}_t).\exp(\Delta), \\
\boldsymbol{S}_t^k &= \exp(a_t \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^k + (\mathbf{1} - \exp(a_t \boldsymbol{r}_t))\boldsymbol{k}_t^\top, \\
\boldsymbol{S}_t^v &= \exp(a_t \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^v + (\mathbf{1} - \exp(a_t \boldsymbol{r}_t))\boldsymbol{v}_t^\top. \\
\boldsymbol{o}_t &= (\boldsymbol{S}_t^v)^\top \mathrm{softmax}\big(\boldsymbol{S}_t^k \boldsymbol{q}_t\big).
\end{aligned}
$$

A.5 Efficient Training using Gated Linear Attention

As Raven recurrence for keys and values, both can be viewed as Gated Linear Attention (GLA) as presented in Yang et al. (2023). We first briefly describe the chunkwise parallel training strategy used in GLA models, and then present two efficient algorithms for training Raven based on GLA kernels.

GLA applies a linear recurrence of the form:

$$
\boldsymbol{o}_t = \mathrm{GLA}\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{G}_t\} : \boldsymbol{S}_t = \boldsymbol{S}_{t-1} \odot \boldsymbol{G}_t + \boldsymbol{v}_t \boldsymbol{k}_t^\top, \quad \boldsymbol{o}_t = \boldsymbol{S}_t \boldsymbol{q}_t,
$$

where $\boldsymbol{G}_t \in \mathbb{R}^{M \times d}$ is a two-dimensional forget gate. Assuming a diagonal forget gate $\boldsymbol{A}_t$, we can rewrite GLA as:

$$
\boldsymbol{o}_t = \mathrm{GLA}\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{G}_t\} : \boldsymbol{S}_t = \boldsymbol{S}_{t-1} \boldsymbol{A}_t + \boldsymbol{v}_t \boldsymbol{k}_t^\top, \quad \boldsymbol{o}_t = \boldsymbol{S}_t \boldsymbol{q}_t.
$$

By unrolling the above recurrence and only focusing in the output, we obtain:

$$
\boldsymbol{o}_t = \left(\sum_{\tau=0}^{t} \left(\prod_{i=\tau+1}^{t} \boldsymbol{A}_i \right) \boldsymbol{v}_\tau \boldsymbol{k}_\tau^\top \right) \boldsymbol{q}_t = \left(\sum_{\tau=0}^{t} \left(\prod_{i=0}^{t} \boldsymbol{A}_i \prod_{j=0}^{\tau} \boldsymbol{A}_j^{-1} \right) \boldsymbol{v}_\tau \boldsymbol{k}_\tau^\top \right) \boldsymbol{q}_t.
$$

We can incorporate the cumulative forget gates into the query/key pairs, leading to:

$$
\boldsymbol{o}_t = \sum_{\tau=0}^{t} \boldsymbol{v}_\tau \left(\prod_{j=0}^{\tau} \boldsymbol{A}_j^{-1} \boldsymbol{k}_\tau \right)^\top \left(\prod_{i=0}^{t} \boldsymbol{A}_i \boldsymbol{q}_t \right).
$$

As $\boldsymbol{A}_t = \mathrm{diag}(\boldsymbol{a}_t)$, we define the cumulative product of these decay vectors as

$$
\boldsymbol{b}_t = \prod_{i=\tau+1}^{t} \boldsymbol{a}_i,
$$

and stack them over a sequence of T tokens into a matrix $\boldsymbol{B} \in \mathbb{R}^{T \times d}$. Similarly by shaping the query, key, and value matrices, we can express the parallel form of the output $\boldsymbol{O} \in \mathbb{R}^{T \times d}$ as Yang et al. (2023):

$$
\boldsymbol{O} = \left(\left((\boldsymbol{Q} \odot \boldsymbol{B}) \left(\frac{\boldsymbol{K}}{\boldsymbol{B}} \right)^\top \right) \odot \boldsymbol{M} \right) \boldsymbol{V}, \tag{21}
$$

where $\boldsymbol{M} \in \mathbb{R}^{T \times T}$ is a binary causal mask. Since Equation (21) may suffer from numerical instability for large sequence lengths T, due to the cumulative product of small values in $\boldsymbol{B}$, parallel training applies Equation (21) in a chunkwise manner. Specifically, the computation is performed over n chunks of sequence length C, with $T = n \times C$. This algorithm results in efficient and fast training for GLA models using specialized GPU kernels.

Considering GLA's efficient training presented as $\mathrm{GLA}(\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{G}_t)$ we now present the efficient algorithms for training Raven.

Looking at Raven recurrence presented in Appendix A.4 we can re-write the output of Raven recurrence ($\boldsymbol{o}_t$) as two-pass GLA as used in GSA (Zhang et al., 2024), followed by:

RSM Parallel Form:

$$\boldsymbol{o}'_t = \text{GLA}\{\boldsymbol{q}_t, \boldsymbol{k}_t, 1 - \exp(a_t \boldsymbol{r}_t), \exp(a_t \boldsymbol{r}_t)\}$$
$$\boldsymbol{o}_t = \text{GLA}\{\text{softmax}(\boldsymbol{o}'_t), \boldsymbol{v}_t, 1 - \exp(a_t \boldsymbol{r}_t), \exp(a_t \boldsymbol{r}_t)\}$$

Therefore, we can directly adapt efficient implementation of GSA kernels using flash-linear attention (FLA) library (Yang and Zhang, 2024). We can use two separate passes of GLA for each hidden states and their parallel forms respectively and measure the output as shown above.

A.6 Raven Design Details

In this section, we describe in detail all our design choices for Raven and final block design of Raven sequence mixer and channel mixer in both SSM and hybrid architectures.

Raven Linear Transformer Raven follows the standard 400M-parameter model design used in the SSM literature (Movahedi et al., 2025; Yang et al., 2023, 2024a,b). We use four heads per block and separate per-head hidden states $\boldsymbol{S}_t^v$ and $\boldsymbol{S}_t^k$ for values and keys. We also use a channel mixer after each RSM block, with skip connections (He et al., 2016) as in Transformers and SSMs. We use QK normalization and apply output gating in the RSM block. We use the Llama tokenizer (Dubey et al., 2024) for Raven and all other models.

Raven Hybrid Model In the hybrid setup we interleaved Raven RSM block with SWA and softmax attention mixers. After each sequence mixer, gated MLP was used as the channel mixer with skip connections similar to hybrid models setup of Yang et al. (2024a). Softmax attention and SWA mixer of hybrid Raven use NoPE as position embedding. We have also evaluated RoPE as position embedding for SWA and attention mixers of hybrid Raven and observed their failure in length generalization (shown in Table 9).

In the SWA hybrid setting, greater pressure is placed on the linear component, since SWA layers can only retrieve information within a fixed local window and cannot directly attend to arbitrary positions. Importantly, SWA-RoPE does not harm length generalization in this setting, because every query–key pair is bounded by the window size; thus RoPE never encounters relative distances outside its training distribution regardless of the total sequence length. As a result, long-range retrieval falls entirely on the linear layer. Raven +SWA with RoPE achieves near-perfect NIAH-1 accuracy across the full 1K–32K range, while GDN+SWA degrades steadily and Mamba-2+SWA collapses to near zero beyond 4K. In this configuration Raven also leads on SWDE, FDA, and SQuAD, confirming that its advantages are architectural rather than dependent on a particular hybrid pairing.

Initialization. As Raven readout uses the softmax function, $\boldsymbol{o}_t = (\boldsymbol{S}_t^v)^\top \text{softmax}\big(\boldsymbol{S}_t^k \boldsymbol{q}_t\big)$, and the memory slots are initialized to zero, empty slots contribute a value of 1 in the readout (since $\exp(0) = 1$ in the softmax). In our initial experiments, we avoided this effect by dynamically masking empty slots with $-\infty$, as is done in softmax causal attention. However, we observed no difference when omitting this mask and using zero-initialized states directly in the readout. Therefore, we adopt this simpler setting for Raven.

A.7 Test-Time Training Point of View

Test-Time Training (TTT) (Sun et al., 2024) interprets the recurrence in linear transformers as an online learning update driven by a per-token objective. Consider the linear attention recurrence

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top. \tag{22}$$

For clarity, we transpose the content matrix $\boldsymbol{U}_t$ as $\boldsymbol{k}_t \boldsymbol{v}_t^\top$ so that it aligns with the TTT framework (Zhang et al., 2025a). This update can be viewed as a single step of gradient descent on the online loss

$$\mathcal{L}(\boldsymbol{S}) = -\langle \boldsymbol{S}\boldsymbol{k}_t,\, \boldsymbol{v}_t \rangle, \qquad \boldsymbol{S}_t = \boldsymbol{S}_{t-1} - \nabla_{\boldsymbol{S}} \mathcal{L}(\boldsymbol{S}), \tag{23}$$

Table 9: **Hybrid Models Retrieval Ability.** The performance of state-of-the-art linear models used as hybrid models with different attention types. Performance is reported on NIAH-1,2,3 and Gray-highlighted SWDE, FDA, and SQuAD.

Hybrid Component	Positional Embedding	Linear Model	No Conv.	SWDE acc↑	FDA acc↑	SQuAD acc↑	NIAH-1 1K	NIAH-1 2K	NIAH-1 4K	NIAH-1 8K	NIAH-1 16K	NIAH-1 32K	NIAH-2 1K	NIAH-2 2K	NIAH-2 4K	NIAH-2 8K	NIAH-2 16K	NIAH-2 32K	NIAH-3 1K	NIAH-3 2K	NIAH-3 4K	NIAH-3 8K	NIAH-3 16K	NIAH-3 32K
Attn	RoPE	GDN	✗	47.1	22.3	34.2	**100**	**100**	**53.4**	**23.4**	**11.6**	**6.0**	**100**	**100**	**53.0**	**20.2**	**10.4**	**5.3**	97.2	87.8	**27.0**	**9.4**	**2.4**	0.0
		Mamba-2	✗	**52.2**	**26.1**	**37.7**	**100**	96.8	16.8	0.0	0.0	0.0	**100**	**100**	1.6	0.0	0.0	0.0	**99.4**	**91.0**	0.0	0.0	0.0	0.0
		Raven	✔	34.5	9.7	33.3	**100**	77.8	0.0	0.0	0.0	0.0	**100**	99.8	0.0	0.0	0.0	0.0	71.8	62.0	0.0	0.0	0.0	0.0
Attn	NoPE	GDN	✗	54.6	67.2	34.5	**100**	**100**	**100**	**100**	93.2	70.5	**100**	**100**	**100**	8.0	0.0	0.0	**94.2**	70.2	50.0	0.0	0.0	0.0
		Mamba-2	✗	**56.3**	**68.8**	**36.0**	**100**	**100**	16.4	0.0	0.0	0.0	**100**	**100**	85.8	0.0	0.0	0.0	79.6	**80.6**	60.8	0.0	0.0	0.0
		Raven	✔	51.4	64.2	31.4	**100**	**100**	**100**	**100**	**98.4**	**78.6**	**100**	**100**	**100**	**100**	**95.4**	**65.4**	89.6	67.0	**73.8**	**60.0**	**10.2**	**14.4**
SWA	RoPE	GDN	✗	30.4	14.7	33.0	78.2	75.8	58.8	54.4	45.0	22.1	68.6	46.6	11.0	3.8	3.2	1.3	55.4	23.0	8.6	2.0	**2.0**	1.1
		Mamba-2	✗	12.0	9.3	36.1	29.8	11.0	6.2	3.4	1.2	0.6	33.4	16.8	5.8	5.2	3.2	**2.8**	4.2	2.8	6.6	**2.2**	0.0	1.4
		Raven	✔	**33.0**	**25.9**	**37.0**	**100**	**99.6**	**99.6**	**99.8**	**99.2**	**99.6**	**99.6**	**99.8**	**95.2**	**51.0**	**6.8**	2.4	**64.4**	**42.4**	**12.0**	0.8	1.8	**2.4**
SWA	NoPE	GDN	✗	**35.6**	**23.9**	32.4	**99.8**	**100**	**98.6**	**59.8**	**26.8**	10.2	**99.2**	**96.6**	**42.0**	6.6	**3.6**	1.3	**77.2**	**53.8**	**24.4**	**2.6**	1.2	0.0
		Mamba-2	✗	13.9	14.6	32.9	30.0	11.0	6.2	0.4	0.0	0.0	33.4	16.8	5.4	0.0	0.0	0.0	18.8	7.8	4.6	0.0	0.0	0.0
		Raven	✔	24.9	22.2	**33.8**	63.2	51.4	44.4	38.8	25.2	**17.8**	54.0	40.8	16.0	**7.8**	**3.6**	**2.8**	16.6	7.0	2.0	0.6	**1.6**	**1.2**

Table 10: An overview of different attention mechanisms through the lens of state updating rules and their learning objective under the TTT framework (Sun et al., 2024). Table style is inspired by Team et al. (2025).

Model	Objective $\mathcal{L}$	Update rule $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} - \nabla_{\boldsymbol{S}}\mathcal{L}(\boldsymbol{S})$
Linear Attention (Katharopoulos et al., 2020)	$-\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle$	$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{k}_t\boldsymbol{v}_t^\top$
RetNet (Sun et al., 2023)	$-\beta_t\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle \;+\; \frac{1}{2}\left\|\sqrt{1-a}\,\boldsymbol{S}_{t-1}\right\|_F^2$	$\boldsymbol{S}_t = a\boldsymbol{S}_{t-1} + \beta_t\boldsymbol{k}_t\boldsymbol{v}_t^\top$
Mamba-2 (Dao and Gu, 2024)	$-\beta_t\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle \;+\; \frac{1}{2}\left\|\sqrt{1-a_t}\,\boldsymbol{S}_{t-1}\right\|_F^2$	$\boldsymbol{S}_t = a_t\boldsymbol{S}_{t-1} + \beta_t\boldsymbol{k}_t\boldsymbol{v}_t^\top$
GLA (Yang et al., 2023)	$-\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle \;+\; \frac{1}{2}\left\|\sqrt{\mathrm{Diag}(1-a_t)}\,\boldsymbol{S}_{t-1}\right\|_F^2$	$\boldsymbol{S}_t = \mathrm{Diag}(a_t)\,\boldsymbol{S}_{t-1} + \boldsymbol{k}_t\boldsymbol{v}_t^\top$
Raven	$-(1 - \exp(a_t\boldsymbol{r}_t))\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle + \frac{1}{2}\left\|\sqrt{1-\exp(a_t\boldsymbol{r}_t)}\,\boldsymbol{S}_{t-1}\right\|_F^2$	$\boldsymbol{S}_t = \boldsymbol{S}_{t-1}\,\mathrm{Diag}(\exp(a_t\boldsymbol{r}_t)) + (1 - \exp(a_t\boldsymbol{r}_t))\,\boldsymbol{k}_t\boldsymbol{v}_t^\top$

where $\mathcal{L}$ serves as the online learning objective. By varying the choice of online objectives and the corresponding gradient-based update rules, a broad family of linear recurrent models can be recovered. An overview of existing diagonal linear recurrence models under this perspective is provided in Table 10.

As shown in Table 10, the forget gates used in different linear transformer architectures can be interpreted through a shared duality with weight decay (Krogh and Hertz, 1992) in the online learning objective $\mathcal{L}$ (corresponding to the $\|\boldsymbol{S}\|$ regularization term). The router in Raven extends this objective by introducing a form of ***selective Dropout*** (Srivastava et al., 2014) into the online learning loss, resulting in sparse state updates. This dropout effect primarily arises from the *sparsity of the router* $\boldsymbol{r}_t$ and the associated online regularizer $1 - \exp(a_t\boldsymbol{r}_t)$, shown in blue: when slot i is not selected ($\boldsymbol{r}_t[i] = 0$) the regularizer vanishes, so that slot incurs no weight decay and its content is preserved exactly. One can look at Raven update as applying ***sparse TTT*** and only updating selected slots.

Models using the Delta update rule, such as DeltaNet and KDA, apply a different online learning objective, which takes the form:

$$\mathcal{L}(\boldsymbol{S}) = \tfrac{1}{2}\|\boldsymbol{S}^\top\boldsymbol{k}_t - \boldsymbol{v}_t\|^2, \tag{24}$$

This online update rule has connections to *Associative Memory* introduced in Hopfield Networks (Hopfield, 1982). *We refer to Table 7 of Team et al. (2025) for more details on the TTT framework.*

A.8 Memory Dynamic Visualizations

We visualize the memory dynamics of Raven. As shown in Figure 6, different heads in Raven exhibit different Effective Sequence Lengths (*ESL*). Some heads are extremely sparse; for example, head 3 in layer 24 activates only a few memory slots to store tokens. In contrast, other heads, such as head 4 in layer 1, activate many more slots and store tokens in a more diverse manner.

Figure 5 further illustrates how different memory slots in Raven process the same simple text differently and how tokens are distributed across slots. Specifically, red regions indicate slots that store passkey tokens, while blue regions represent slots shared between passkey and non-passkey tokens. As observed, only a few heads are responsible for retrieving and storing passkey tokens without overwriting them (red regions). In contrast, some heads, such as head 2 in layer 24, do not allocate dedicated memory slots for retrieval tokens; instead, they automatically use shared memory for both retrieval and regular tokens.

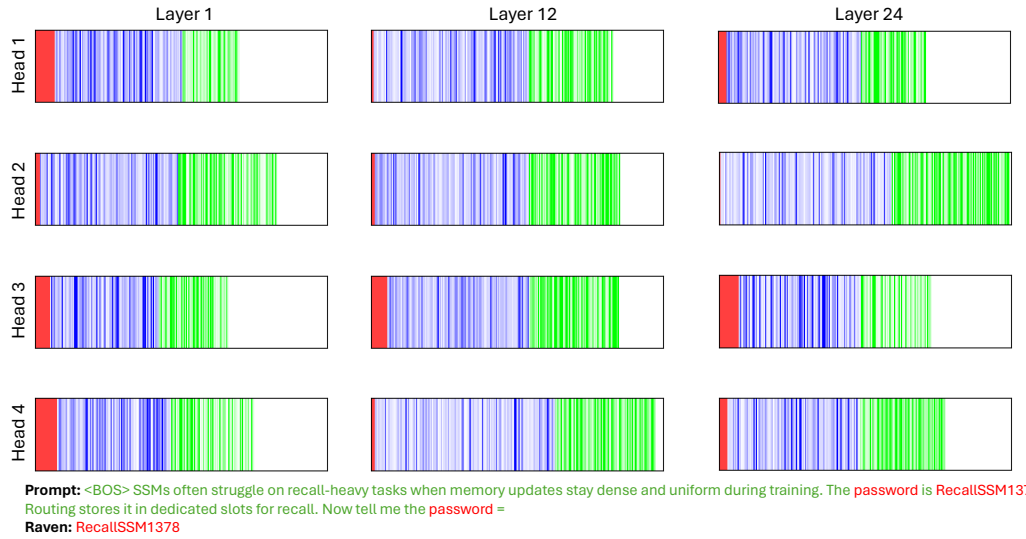


Figure 5: **Memory Visualization.** Memory of Raven while being asked the password within the prompt. The figures show the hidden state of Raven $\boldsymbol{S}_t$ at the end of processing the prompt and answering to the question (time point $t = L$) for **(Top)** Layer 24, **(Middle)** Layer 12, **(Bottom)** Layer 2 for the first head among 4 heads at 400M scale. Red slots are storing only retrieval (red) tokens, green slots are memory slots only storing the normal (green) tokens, and blue are shared memory slots between both token types.

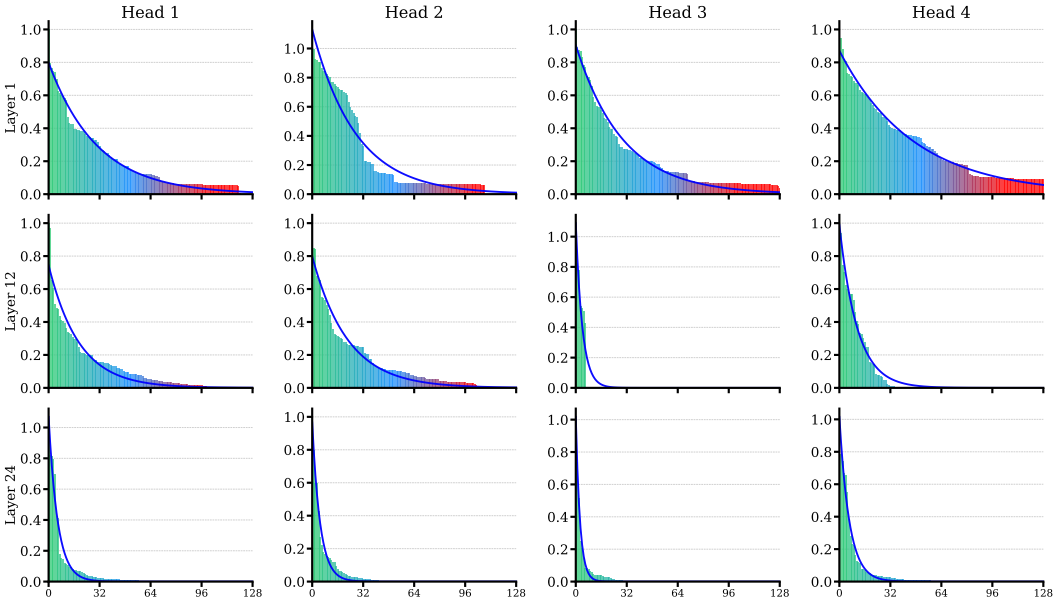


Figure 6: **Effective Sequence Length of Raven.** The normalized *effective sequence length* for a NIAH-1 sample with a sequence length of 16K. Results are from the first, 12th, and last (24th) layers, for all four heads of the *400M* model. Slots are reordered from highest to lowest.

Figure 8 also clearly shows that for a given sentence with a passkey Raven retrieval head clearly stores the password in unique sections (slots) of memory without over-writing it, while for other tokens it will store them in shared parts of memory. Moreover, a non-retrieval head stores all tokens equally in the memory.

B Related Work

B.1 Linear Sequence Mixers

Linear Transformers and SSMs. Softmax attention (Vaswani et al., 2017) scales quadratically with sequence length, making it costly for long contexts and motivating a growing body of sub-quadratic alternatives. Within this line, Mamba (Gu and Dao, 2024) matches transformer baselines at similar scale, but its sequential selective scan slows training. Mamba-2 (Dao and Gu, 2024) improves training throughput by simplifying the model and introducing State Space Duality (SSD), later scaled in follow-up work (Bick et al., 2025a; Waleffe et al., 2024; Zuo et al., 2024). On the linear-attention side (Katharopoulos et al., 2020), Gated Linear Attention (GLA) (Yang et al., 2023) focuses on faster implementations through optimized kernels and hardware-aware training.

A shared limitation is that both GLA and Mamba-2 rely on diagonal or scalar decay, which does not mix across dimensions of the hidden state and can restrict performance on tasks requiring richer state dynamics (Merrill et al., 2024). DeltaNet (Yang et al., 2024b) addresses this by adopting a recurrence based on the Delta rule (Schlag et al., 2021), where the forget gate depends on a row of the previous state, inducing non-diagonal decay and enabling mixing across state elements. This is made practical by an efficient parallel training algorithm (Yang et al., 2024a,b) that reformulates the recurrence into a chunkwise form, enabling hardware-efficient parallelization despite the non-diagonal structure of the forget gate.

Unlike prior approaches, Raven follows a different design point. It retains fast training by using diagonal decay at the level of individual memory elements, while coupling the elements within each row via a shared row-level update. As a result, Raven does not rewrite the full memory at every step, enabling persistence that is useful for language modeling.

Dual-State as Higher-Order Linear Attention. The recurrence in Equation (3) maintains two coupled memory states, the key and value caches. Its *state update* is linear (a rank-one write routed by a one-hot vector) while the *readout* applies standard softmax attention over the M cached tokens. Following Zhang et al. (2025b), this falls under *higher-order linear attention*, meaning that multiple coupled memories are updated by a linear recurrence (without requiring a softmax-free attention readout). Models such as GSA (Zhang et al., 2024), ABC (Peng et al., 2021), and SWA (Beltagy et al., 2020) are also second-order instances ($N = 2$), and the definition extends directly to N coupled memories.

Raven most closely resembles GSA: both maintain key/value slot memories with a softmax readout. GSA uses dense per-slot sigmoid gates without normalization across slots; Raven normalizes its selected router scores across slots and couples writing and decay so that unselected slots remain unchanged. Thus, the distinction includes normalized routing and selective decay, as well as Top-K sparsity.

Hybrid Architectures Following the success of SSMs in language modeling and the strong retrieval capabilities of Transformers, several studies have explored hybrid architectures that combine the two. This line of work divides broadly into two directions. The first is full pretraining: Samba (Ren et al., 2024) and Jamba (Lieber et al., 2024) interleave Mamba layers with full softmax attention blocks, Qwen3 (Yang et al., 2025) integrates GDN with softmax attention and MoE channel mixers, and GDN (Yang et al., 2024a) proposes variants that combine Mamba-2, GDN, and full attention within a unified architecture. The second direction distills pretrained Transformers into hybrid SSM variants (Bick et al., 2025b; Paliotta et al., 2025; Wang et al., 2025). Bick et al. (2026) take this further by showing that preserving only the Gather-and-Aggregate retrieval heads while converting the rest into SSM layers recovers most of the Transformer's recall capability at a fraction of the cost.

Retrieval in LLMs. A growing body of work has identified in-context retrieval as a primary differentiator between Transformer and SSM performance (Arora et al., 2023; Jelassi et al., 2024; Park et al., 2024), with theoretical and empirical evidence that SSMs struggle with associative recall and precise copying (Jelassi et al., 2024; Wen et al., 2024). Bick et al. (2025c) localize this limitation further, attributing the performance gap to a small subset of Gather-and-Aggregate (G&A) heads responsible for in-context retrieval — suggesting the gap reflects a specific functional deficit rather than a holistic architectural failure. Because linear models compress history into a fixed-size state,

they cannot fully reproduce G&A behavior. Our results suggest that Raven recovers a restricted but meaningful form of this capability within the constraints of fixed-size models, selectively retrieving information that is queried in advance.

Length Generalization in LLMs. Length generalization is the ability to preserve accuracy when inference sequences exceed training lengths. In Transformer LLMs, degradation is often tied to positional encodings and attention statistics shifting out of the training regime. ALiBi replaces explicit positional embeddings with distance-dependent attention biases, improving extrapolation from shorter training contexts (Press et al., 2022). For RoPE-based models (Su et al., 2021), Position Interpolation rescales position indices to keep relative rotations within the trained range, enabling large context extensions with limited fine-tuning (Chen et al., 2023); YaRN further improves stability and data efficiency for such adaptation (Peng et al., 2024). In recurrent and SSM-based models, length failures are attributed to "unexplored states": hidden states during long rollouts fall outside the training distribution, and broadening state coverage can substantially recover performance (Buitrago Ruiz and Gu, 2025). Complementary work analyzes out-of-distribution state-space dynamics to extend length generalization in Mamba (Lu et al., 2025).

Raven connects these perspectives via sparse memory routing: by updating only a subset of memory slots at each step, it reduces overwrite while implicitly exposing training to a distribution of effective write horizons, which improves long-range recall and length generalization.

B.2 Routing and Mixture of Experts (MoE)

Routing: Token-to-expert routing is a defining component of *Mixture-of-Experts* (*MoE*) models (Fedus et al., 2022; Jacobs et al., 1991), enabling efficient channel mixers at larger model scales. In MoEs, the router assigns each token to a small subset of *expert* feed-forward networks (FFNs). Therefore, only the selected experts are activated for each token during inference, improving per-token compute efficiency, especially at large model scales (e.g., Mixtral 8×7B (Jiang et al., 2024)). MoEs have also been used for channel mixing in hybrid architectures. For example, hybrid models such as Kimi Linear (Team et al., 2025) (based on GDN) and Nemotron (Blakeman et al., 2025) (based on Mamba-2) use MoE layers instead of gated MLPs as channel mixers. Unlike MoEs, which apply routing within their channel mixers, Raven applies routing in its sequence mixer and within its memory $\boldsymbol{S}_t$. Routing has also been used in prior SSMs such as Sparse State Expansion (SSE) (Pan et al., 2025), which routes over the key vector $\boldsymbol{k}_t$ and sparsely selects its components.

Shared Experts: In recent large language models, such as DeepSeek-V3 (Liu et al., 2024) and DeepSeekMoE (Dai et al., 2024), not all experts are selected dynamically by the router. Instead, a subset of feed-forward networks (FFNs) is always activated and processes *all* tokens; these are referred to as *shared experts*. This strategy helps avoid under-training experts in MoEs.

Mixture of Memories (MoM). Mixture of Memories (*MoM*) (Du et al., 2025) applies routing among heads of linear transformers, particularly GDN (Yang et al., 2024a). Therefore, not all tokens are processed by each head; instead, only selected heads store a token in their hidden states. In the spirit of our work, *MoM* can be considered a binary router r_t^h for each hidden state $\boldsymbol{S}_t^h$, which decides whether token t is stored in the hidden state of the h-th head $\boldsymbol{S}_t^h$. Moreover, MoM also includes a shared memory, which is updated for all tokens analogous to shared experts in MoEs.

While MoEs apply routing in the channel mixer and MoM applies routing across the heads of the sequence mixer, Raven applies RSM routing within each head's memory allocation, introducing a ***new dimension*** *for sparsity.*

Mixture of Heads (MoH). Mixture of Heads (MoH) applies routing to the attention heads of a softmax Transformer (Jin et al., 2024). Instead of having all heads process every token in multi-head attention (MHA), MoH routes each token to a subset of selected heads, while a set of shared heads always processes all tokens. This is conceptually similar to MoM, which applies routing across different linear Transformer heads: both route at the granularity of heads, whereas Raven applies routing at a higher resolution within its memory $\boldsymbol{S}_t$. Routing over heads could also be combined with the Raven router as a promising direction for future work, enabling extremely sparse writes in the sequence mixer.

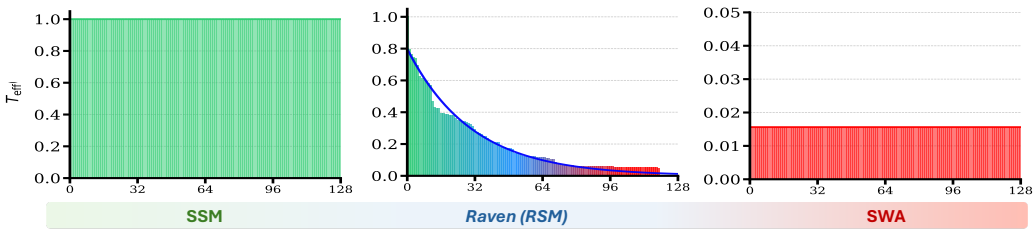


Figure 7: **Raven Effective Sequence Length.** Normalized effective sequence length for a NIAH-1 sample at sequence length 16K. SWA : each token stored in exactly one slot (FIFO). Raven : hidden state $\boldsymbol{S}_t$ for layer 1, head 1 (256 slots, Top_{32}). SSM : each token stored in all slots with decay. Slots are reordered by usage frequency; results are for 400M parameter models.

C Analysis: Learned Memory Allocation

A key question for any routed memory model is whether the routing is truly meaningful — does the model learn to allocate memory purposefully, or does it route arbitrarily? This section provides evidence that Raven develops structured, content-aware memory allocation. We study (i) the distribution of effective sequence lengths across slots, which reveals whether slots specialize in terms of how much they process, and (ii) routing patterns on a retrieval task, which reveals *what* is being stored and where.

C.1 Effective Sequence Length in Routed Memory

The routing mechanism in Raven has a direct consequence on how much each slot processes: a slot that is rarely selected accumulates fewer tokens than one that is frequently written to. To make this precise, we unroll the key recurrence:

$$\boldsymbol{S}_t^k \overset{\text{Unroll}}{=} \sum_{j=0}^{t} \prod_{i=j+1}^{t} \exp(a_i \boldsymbol{r}_i) \odot \big(\mathbf{1} - \exp(a_j \boldsymbol{r}_j)\big) \boldsymbol{k}_j^\top .$$

The same holds for $\boldsymbol{S}_t^v$. Writing $\boldsymbol{S}_t = [\boldsymbol{S}_t[1], \ldots, \boldsymbol{S}_t[M]]$ generically for either state, the update for slot i is:

$$\boldsymbol{S}_t[i] = \sum_{j=0}^{t} \boldsymbol{k}_j \left(\mathbf{1} - \exp(a_j r_j[i])\right) \odot \exp\left(\sum_{\ell=j+1}^{t} a_\ell \, r_\ell[i] \right),$$

where $r_t[i]$ is the i-th entry of $\boldsymbol{r}_t$. Since $r_t[i] \neq 0$ only when slot i is selected, $\boldsymbol{S}_t[i]$ depends only on tokens routed to it. This motivates a *per-slot notion of context length*.

Definition 4 (Effective Sequence Length)**.** Given routing weights for slot i across a sequence of length T, the *effective sequence length* is:

$$T_{\text{Eff}}(i) \;=\; \|\boldsymbol{r}_i\|_0 \;=\; \sum_{t=1}^{T} \mathbf{1}\big[r_{[i,t]} \neq 0\big] . \tag{25}$$

We compute the ESL per memory slot and compare Raven to SWA and SSMs in Figure 7. In SWA, ESL is uniform by construction, each slot receives exactly one token per cycle. In SSMs, ESL is also uniform but maximal, every slot sees every token. Raven breaks this symmetry, with slots falling into three regimes depending on how frequently the router selects them: short ESL (*red*), long ESL (*green*), and intermediate (*blue*). Figure 6 extends this analysis across all layers and heads, showing a consistent ESL spread throughout the model.

An important consequence of non-uniform ESL is **length generalization**. SSMs trained at a fixed context length can degrade on longer sequences (Chen et al., 2025). State passing addresses this by varying effective lengths during training (Ruiz and Gu, 2025). Raven achieves the same effect implicitly: within every training example, different slots see different subsequence lengths, naturally exposing the model to variable-length dynamics without any explicit intervention.

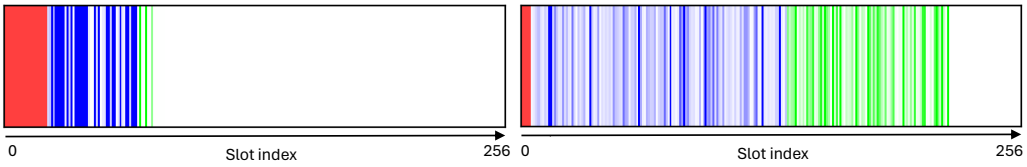


Figure 8: **Raven Memory Dynamics.** Memory allocation for two different heads of Raven on a synthetic NIAH-style task. Red slots store tokens that are important for retrieval (e.g., passwords), Green slots store non-retrieval tokens, and Blue slots are shared memory slots between the two type of tokens (red and green). Different heads allocate different amounts of slots to retrieval-important tokens, showing non-uniform memory allocation across heads.

C.2 Selective Allocation in a Retrieval Task

Non-uniform ESL alone does not tell us *what* gets routed where. To answer this, we visualize slot assignments on a synthetic retrieval task (Figure 8). The picture is clear: some slots specialize as *retrieval slots*, receiving almost exclusively passkey tokens (Mohtashami and Jaggi, 2023) (red), while others are reserved for non-retrieval content (green). Shared slots (blue) are rare in heads with strong retrieval behavior. These retrieval slots are analogous to retrieval heads (Bick et al., 2025c) in Transformers, but operate at finer resolution over the continuous state $\boldsymbol{S}_t \in \mathbb{R}^{M \times d}$. By routing retrieval-critical tokens to dedicated slots, Raven limits destructive overwrite and preserves them across long contexts. Section 5 shows this translates to high retrieval accuracy up to $\mathbf{16}\times$ the training length.

C.3 Intentional Memory Imbalance

The uneven allocation above is not a side effect; it is by design. MoE routers typically include a load-balancing loss (Shazeer et al., 2017) to encourage uniform token-to-expert assignment and prevent expert collapse. In our setting, uniform memory usage would be counterproductive: forcing equal ESL across slots would prevent the emergence of retrieval slots and expose stored tokens to unnecessary overwrite.

Raven therefore omits any *load-balancing* auxiliary loss, allowing the router to specialize freely. However, as discussed in Section 6, Raven applies Gumbel noise to encourage full exploration of its hidden-state memory and prevent collapse to only a few memory slots. The result, shown in Figure 5, is that Raven naturally routes retrieval-critical tokens to dedicated regions of $\boldsymbol{S}_t \in \mathbb{R}^{M \times d}$ without any explicit supervision to do so.

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: Yes they do.

   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: We discuss the limitations in the main text.

   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: [Yes]

Justification: We have proofs for all our theorems.

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 architecture and experimental setup are described in the paper and appendix. Code is available at Raven Code.

   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: Code is available at Raven Code; datasets and their sources are described in the experimental details.

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 a dedicated section on our experimental setup which follows the standard linear models.

   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: [Yes]

   Justification: We did.

   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 have mentioned details on our experiments and appendix section.

   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: Yes.

   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: [Yes]

    Justification: Our paper is mostly using small LMs and does not have specific social impact, however if they are going to be scaled up all ethical constraints for large language models should apply to our model as well.

    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: Our paper is methodological, theory, and uses small scale experimental setup.

    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: Yes.

    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: [N/A]

    Justification: Non new assets.

    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: No human subject.

    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: No human subject.

    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: [Yes]

Justification: Yes we used only to improve 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.