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.

Raven Code Raven Blog

1 Introduction

Fixed-size memory models often match Transformers in language modeling but still fall short in long-range recall (Bick et al., 2025c), despite being able to carry forward task-relevant features over arbitrarily long contexts. This gap reflects how linear models manage history. 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. 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 maintain a fixed set of memory slots and, at each step, (i) route newly written content into one or more of these slots and (ii) apply an explicit decay (forgetting) operator to the updated slot. Under this view, SSMs and SWA emerge as two extremes on the routing axis: SSMs write densely with gradual forgetting, while SWA writes sparsely and deterministically with hard eviction — making explicit a tradeoff that existing architectures have left implicit, 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 a selected subset of memory slots, and applies decay only to those updated slots*. This lets the model

[*]Equal contribution (alphabetical order)

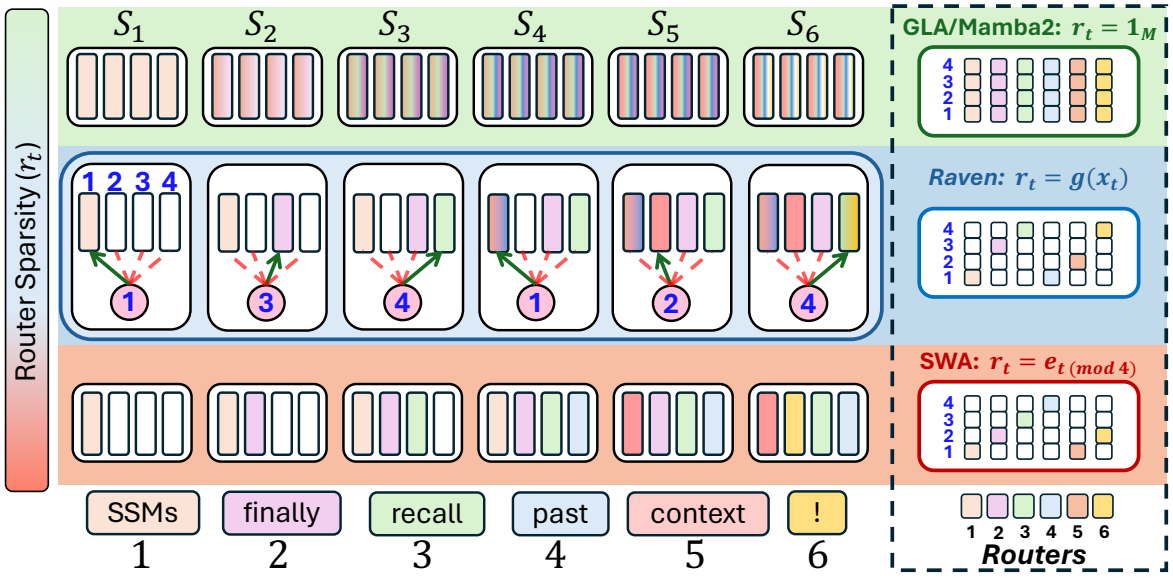


Figure 1: **Raven Overview.** Visualization of three different sequence mixers using ***Routing Slot Memories***, with different router choices. ***(a)*** SWA memory allocation as *first-in-first-out* strategy using a **one-hot vector** e_t as router. ***(b)*** SSM memory allocation, which projects each token in all memory slots using a **dense, all-ones router** 1_M ***(c)*** **Raven** memory allocation, which uses a **selective router** for writes. Visualization uses a sequence of $T = 6$ token and $M = 4$ memory slots for the hidden state $\boldsymbol{S}_t$ and $\text{Top}_K = 1$ for Raven router.

better organize the memory, keeping specific tokens recoverable even when the context grows far beyond the training length. Importantly, the Raven block simplifies prior linear models and *does not rely on convolutional or sliding-window attention layers* for effective retrieval (Gu and Dao, 2024; Yang et al., 2023).

As instantiations of RSMs, Raven can be contrasted with both SWA and SSMs:

Compared to SWA, Raven selectively routes writes to memory and gradually decays existing content in the chosen slots, generalizing SWA's fixed window approach. This enables Raven to achieve perfect recall on Needle-in-a-Haystack benchmarks where SWA performs near zero, with substantial improvements on recall-heavy tasks in both standalone and hybrid settings across different model scales.

Compared to SSMs, Raven avoids dense full-state updates by restricting both writing and forgetting to a selected subset of memory slots, providing long-term persistence that prior SSMs lack. Raven is competitive with or outperforms recent SSMs and linear models—including Mamba-2, Gated Delta-Net (GDN), and Gated Linear Attention (GLA)—on recall-intensive benchmarks, while maintaining strong accuracy even when extrapolating 16 $\times$ beyond those seen during training.

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 $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$, the model maintains a 2-dimensional memory matrix $\boldsymbol{S}_t \in \mathbb{R}^{M \times d}$ for each head. Here, M corresponds to the value head dimension, while d corresponds to the query/key head dimension (aka the state expansion). At step t, the token $\boldsymbol{x}_t \in \mathbb{R}^d$ is linearly projected to $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^d$ and $\boldsymbol{v}_t \in \mathbb{R}^M$ [1], and updates memory via

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

[1] 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).

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). Intuitively, $\boldsymbol{A}_t$ controls how past memory is retained, $\boldsymbol{v}_t\boldsymbol{k}_t^\top$ writes new content, and $\boldsymbol{o}_t = \boldsymbol{S}_t\boldsymbol{q}_t$ reads from memory. 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 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 d rows).

2.2 Sliding-Window Attention as a Dual-State SSM

Sliding-Window Attention (SWA) maintains a key/value cache for the most recent M tokens and restricts attention scores to this window. Unlike SSMs, which compress all history into a fixed-size state, SWA stores token-level memories for the last M steps. Nevertheless, SWA admits a state-space view: it maintains two coupled states (key and value caches) updated by a linear recurrence.

More precisely, let $\boldsymbol{S}_t^k, \boldsymbol{S}_t^v \in \mathbb{R}^{M\times d}$ denote the key and value caches at time t, where each row corresponds to a memory slot. 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:

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

Since $\boldsymbol{e}_t$ is one-hot, this simplifies to

$$\boldsymbol{S}_t^k = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^k + \boldsymbol{e}_t\boldsymbol{k}_t^\top, \qquad \boldsymbol{S}_t^v = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^v + \boldsymbol{e}_t\boldsymbol{v}_t^\top. \tag{4}$$

Thus, SWA computes attention over the window:

$$\boldsymbol{o}_t = (\boldsymbol{S}_t^v)^\top \mathrm{softmax}\big(\boldsymbol{S}_t^k\boldsymbol{q}_t\big), \tag{5}$$

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 5, or attend only over the filled prefix. From Equation (4), 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

$$\boldsymbol{S}_t = (\boldsymbol{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1} + \boldsymbol{e}_t\boldsymbol{u}_t^\top, \tag{6}$$

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

3 Routing Slot Memories

As shown in Section 2, linear sequence models such as SSMs and SWA both maintain a finite-dimensional state $\boldsymbol{S}_t \in \mathbb{R}^{M\times d}$ that compresses the entire history into M memory slots.

The central question is therefore not whether to forget—as all finite-state models inevitably must—but rather *where* information should be written and *which slots* should be preserved. We formalize this perspective through the lens of **Routing Slot Memories (RSMs)**, a unified class of slot-separable linear recurrences that makes the write location explicit via a *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:

$$\boldsymbol{S}_t = g_t(\boldsymbol{S}_{t-1}, \boldsymbol{x}_t), \qquad \boldsymbol{o}_t = f(\boldsymbol{S}_t, \boldsymbol{x}_t). \tag{7}$$

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:

$$\boldsymbol{S}_t[i] = g_{t,i}\big(\boldsymbol{S}_{t-1}[i],\, \boldsymbol{x}_t\big). \tag{8}$$

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

Corollary 3.1 (Slot Separability of a Linear Update)**.** Consider the linear update

$$\boldsymbol{S}_t \;=\; \boldsymbol{D}_t \boldsymbol{S}_{t-1} \boldsymbol{A}_t \;+\; \boldsymbol{U}_t, \tag{9}$$

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

$$\text{Update is slot-separable} \quad \Longleftrightarrow \quad \boldsymbol{D}_t \text{ is diagonal.} \tag{10}$$

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

Equation (9), which incorporates decays on both sides of the linear update, was also introduced by Zhong et al. (2025). Slot separability constrains how each slot is updated, but does not specify where new information is written. To make the write location explicit, we introduce a router.

Definition 2 (Router)**.** A *router* $\boldsymbol{r}_t = r(\boldsymbol{x}_t, t) \in \mathbb{R}^M$ is a vector that determines, at each time step, how new information is distributed across slots. The entry $\boldsymbol{r}_t[i]$ specifies the write intensity for slot i, gating how much of the incoming information is written to that slot. 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:

$$\boldsymbol{S}_t = \underbrace{(\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}}_{\text{preserved memory}} + \underbrace{\boldsymbol{r}_t \odot \big(\boldsymbol{D}_t\, \boldsymbol{S}_{t-1}\, \boldsymbol{A}_t + \boldsymbol{U}_t\big)}_{\text{updated memory}}, \tag{11}$$

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. Its KV-cache can be written as the recurrence:

$$\boldsymbol{S}_t^k = (\mathbf{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^k + \boldsymbol{e}_t \, \boldsymbol{k}_t^\top, \qquad \boldsymbol{S}_t^v = (\mathbf{1}_M - \boldsymbol{e}_t) \odot \boldsymbol{S}_{t-1}^v + \boldsymbol{e}_t \, \boldsymbol{v}_t^\top, \tag{12}$$

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 $[\boldsymbol{k}_t, \boldsymbol{v}_t] \in \mathbb{R}^{2d}$.

ABC. Attention with Bounded Memory Control (Peng et al., 2021) uses dense input-dependent routing without any decay:

$$\boldsymbol{S}_t^k = \boldsymbol{S}_{t-1}^k + \boldsymbol{r}_t \, \boldsymbol{k}_t^\top, \qquad \boldsymbol{S}_t^v = \boldsymbol{S}_{t-1}^v + \boldsymbol{r}_t \, \boldsymbol{v}_t^\top, \tag{13}$$

using a softmax router of $\boldsymbol{r}_t = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t)$. This way, the effect of each token across the M memory slots is bounded and sums to 1. However, ABC lacks decay of historical information, so it is equivalent to an RSM with the following parameters:

$$\boldsymbol{A}_t = \boldsymbol{I}, \quad \boldsymbol{D}_t = \boldsymbol{I}, \quad \boldsymbol{r}_t = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t), \quad \boldsymbol{U}_t = [\boldsymbol{k}_t, \boldsymbol{v}_t]$$

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

$$\boldsymbol{S}_t^k = (\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^k + \boldsymbol{r}_t \, \boldsymbol{k}_t^\top, \qquad \boldsymbol{S}_t^v = (\mathbf{1} - \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^v + \boldsymbol{r}_t \, \boldsymbol{v}_t^\top, \tag{14}$$

with $\boldsymbol{r}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/\tau}$. Each token now decides how strongly to update each slot, allowing the model to be more conservative about overwriting. However, the router is typically dense: every slot receives some write signal at every step, which limits the model's ability to isolate and protect specific memories. GSA in the RSM view has the parameters

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

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, as they all use a dense all-ones router:

$$\text{Mamba-2:} \quad \boldsymbol{S}_t[i] = a_t \, \boldsymbol{S}_{t-1}[i] + \boldsymbol{v}_t[i] \, \boldsymbol{k}_t^\top \qquad \boldsymbol{D}_t = \boldsymbol{I}, \quad \boldsymbol{A}_t = a_t \qquad (\textit{shared scalar} \text{ decay}), \tag{15}$$

$$\text{GLA/Mamba-1:} \quad \boldsymbol{S}_t[i] = \boldsymbol{S}_{t-1}[i] \odot \boldsymbol{a}_t + \boldsymbol{v}_t[i] \, \boldsymbol{k}_t^\top \qquad \boldsymbol{D}_t = \boldsymbol{I}, \quad \boldsymbol{A}_t = \mathrm{diag}(\boldsymbol{a}_t) \qquad (\textit{channel-wise} \text{ decay}). \tag{16}$$

One can also transpose the hidden state $\boldsymbol{S}_t^\top$ and apply the multiplications from the opposite side, resulting in an equivalent formulation that matches the notation 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:

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1}(\boldsymbol{I} - \beta_t \, \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \, \boldsymbol{v}_t \boldsymbol{k}_t^\top. \tag{17}$$

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.

Table 1: **Routing Slot Memories.** Design of the *Router* and *Decay* components across multiple architectures. The bar on the right along with row colors display each model's spectrum as a function of router sparsity, with SWA and SSM marking the two 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	$\boldsymbol{g}_t/\mathbf{1}^\top\boldsymbol{g}_t$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f\left(\boldsymbol{S}_t^k\boldsymbol{q}_t\right)$
GSA (Zhang et al., 2024)	$\mathbf{1}_M-\sigma(\boldsymbol{W}\boldsymbol{x}_t)^{1/\tau}$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f\left(\boldsymbol{S}_t^k\boldsymbol{q}_t\right)$
ABC (Peng et al., 2021)	$\mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t)$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f\left(\boldsymbol{S}_t^k\boldsymbol{q}_t\right)$
SWA (Beltagy et al., 2020)	$\boldsymbol{e}_t$	$[\boldsymbol{k}_t\ \boldsymbol{v}_t]^\top$	$\boldsymbol{I}$	$(\boldsymbol{S}_t^v)^\top f\left(\boldsymbol{S}_t^k\boldsymbol{q}_t\right)$

The routing/forgetting tradeoff. The RSM view exposes a fundamental tension across existing architectures. SSMs write densely to every slot and delegate all forgetting to decay, which is effective for smooth compression but unable to protect any slot from interference. SWA writes sparsely but is input-blind, discarding information by position rather than content. GSA and ABC (Peng et al., 2021) gain input-dependent routing, but their dense writes still expose every slot at every step. The missing combination—**sparse**, **input-dependent** routing paired with **explicit decay**—is precisely what Raven introduces in Section 4.

4 Raven: Persistent Memory with Sparse Routing

Section 3 exposes a gap in the design space: SSMs write densely and rely on decay to forget; SWA writes sparsely but is input-blind; GSA and ABC (Peng et al., 2021) add input-dependent routing, but the writes remain dense, since every slot receives a nonzero update at each step. Raven fills this gap. It maintains separate key/value states (as in SWA), uses a *sparse*, *input-dependent* router to control where writes land, and retains explicit decay to control how long they persist. Thus, relative to SWA, Raven improves memory allocation through content-dependent routing and position handling through decay rather than RoPE (Afzal, 2026); relative to dense SSMs, it addresses the persistence limitations of dense state updates by writing selectively to a subset of slots.

4.1 Raven Memory Update

Recurrence. The central design principle of Raven is to decouple *where* information is stored from *how long* it persists. We implement this with a routed slot update that lets different slots be updated at different times:

$$\begin{aligned}\boldsymbol{S}_t^k &= \exp(a_t\boldsymbol{r}_t)\odot\boldsymbol{S}_{t-1}^k + \left(\mathbf{1}-\exp(a_t\boldsymbol{r}_t)\right)\boldsymbol{k}_t^\top,\\ \boldsymbol{S}_t^v &= \exp(a_t\boldsymbol{r}_t)\odot\boldsymbol{S}_{t-1}^v + \left(\mathbf{1}-\exp(a_t\boldsymbol{r}_t)\right)\boldsymbol{v}_t^\top,\end{aligned} \tag{18}$$

with readout $\boldsymbol{o}_t = (\boldsymbol{S}_t^v)^\top\,\mathrm{softmax}\left(\boldsymbol{S}_t^k\boldsymbol{q}_t\right)$, where $\odot$ broadcasts over the feature dimension d. When $\boldsymbol{r}_t[i]$ is large, slot i is *decayed and overwritten*. When $\boldsymbol{r}_t[i]\approx 0$, $\exp(a_t\boldsymbol{r}_t[i])\approx 1$ and the slot is *left untouched*. Raven uses SWA-like separate states for keys and values memory writes.

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):

$$a_t = -\mathrm{SoftPlus}(\boldsymbol{w}^\top\boldsymbol{x}_t)\,\exp(\Delta). \tag{19}$$

with Δ being a learnable scalar as used 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.

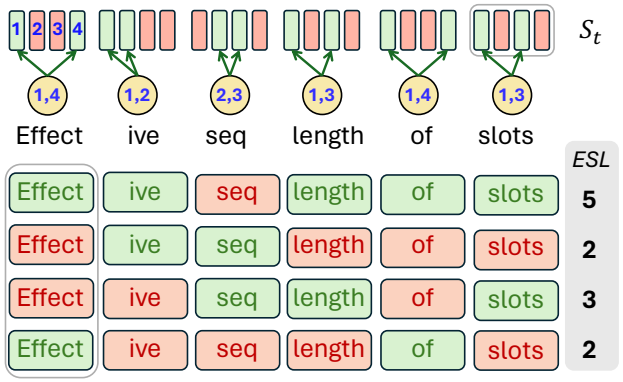


Figure 2: **Effective Sequence Length Visualization.** For a hidden state with $M = 4$ memory slots and a sequence of length $T = 6$, each memory slot processes a different *effective sequence length* ***(ESL)***, depending on the router.

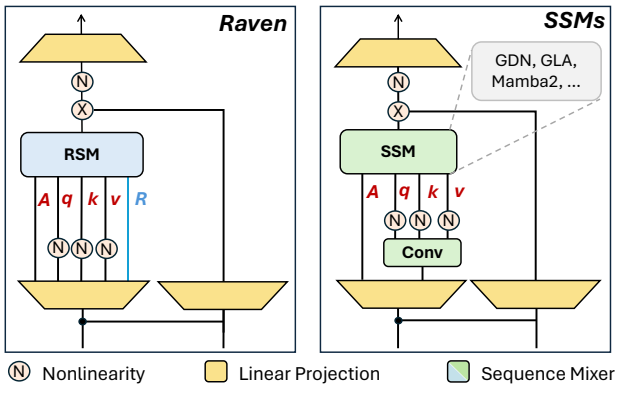


Figure 3: **Raven Neural Architecture.** ***(Left)*** Raven block vs. ***(Right)*** other linear models. Raven requires no short-range convolutions, yielding a simpler block than existing SSMs and linear transformers.

Router. The router determines which slots receive writes at each step. It needs to be (i) sparse, so that unselected slots are fully preserved, and (ii) input-dependent, so that the model can learn to route by content rather than by position. Raven achieves both 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:

$$\boldsymbol{g}_t = \mathrm{KeepTop}_K(\boldsymbol{m}_t) = \begin{cases} \boldsymbol{m}_t[i], & \text{if } i \in \mathrm{TopK}(\boldsymbol{m}_t), \\ 0, & \text{otherwise.} \end{cases}, \qquad \boldsymbol{r}_t = \frac{\boldsymbol{g}_t}{\alpha \sum_{i=1}^{M} \boldsymbol{g}_t[i]}. \tag{20}$$

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

Unlike MoE routers, Raven omits a load-balancing loss (Shazeer et al., 2017). While uniform expert usage is desirable in MoEs, uniform memory allocation is counterproductive here: Raven intentionally routes unevenly. Retrieval-critical tokens (e.g., passkeys) are routed to dedicated slots and protected from overwrite, while general content is distributed across shared slots. The non-uniform distribution of routing results in different slots observing different sequence lengths as shown in Figure 2 (detailed at Section 7). This mirrors the finding that recall in Transformers concentrates in a small subset of heads (Bick et al., 2025c); Figure 7 shows Raven exhibiting the same specialization at the slot level.

4.2 Raven Block Design

Raven uses a simple block consisting of a channel mixer, standard nonlinearities, and normalization. Notably, it omits short-range convolutions entirely.

- **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 apply short-range convolutions to 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. A comparison between the Raven block and other linear models is shown in Figure 3.

Table 2: **In-context recall benchmarks and NIAH accuracy vs. context length and cache size.** We report accuracy (%) on SWDE/FDA/SQuAD and on single NIAH-1/2/3 across context lengths. Rec. mem. and Conv. mem. denote the millions of cached state elements used during decoding. The common baseline of 12.5M recurrent elements corresponds to a 24-layer model with a total state size of 0.57M per layer (derived from 2× factors in architectures like GSA/SWA or larger single tensors in Mamba-2/GDN).

Model	**# Params**	**(Rec / Conv)**	**Recall tasks**			**NIAH-1**						**NIAH-2**						**NIAH-3**					
	(M)	Mem. (M elts)	SWDE	FDA	SQuAD	1K	2K	4K	8K	16K	32K	1K	2K	4K	8K	16K	32K	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

4.3 Unifying Sparse and Selective Writes

The recurrence in Equation (18) is specific to Raven, but the underlying write-and-forget structure it embodies is shared across a broader family of models. 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 occupies the middle ground between both, using a diagonal overwrite matrix $\boldsymbol{P}_t$ rather than a rank-one projector $\boldsymbol{r}_t\boldsymbol{r}_t^\top$:

$$\boldsymbol{S}_t \;=\; \boldsymbol{S}_{t-1}\,(\boldsymbol{I} - \boldsymbol{P}_t) + \boldsymbol{v}_t\,\boldsymbol{p}_t^\top, \tag{21}$$

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.

5 Empirical Validation

We evaluate Raven across three axes: retrieval ability, general language modeling, and hybrid performance. Together, these experiments validate that sparse, input-dependent routing improves memory utilization in ways that translate across tasks, context lengths, and architectural configurations.

Experimental Setup. We follow the standard training recipe used across linear and quadratic sequence models, carefully matching memory size and parameter counts across baselines to ensure fair comparison. Full training details and model configurations are provided in Appendix A.3.

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	400	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	<u>40.5</u>
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	<u>44.5</u>
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	<u>44.5</u>
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

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 models trained at a context length of 2048 tokens and 800M models trained at the context length of 4096. The results in Table 2 are stark. Strong baselines like Mamba-2 and GDN degrade significantly beyond 8K tokens in 400M range — already $4\times$ their training length, as their dense state update forces every slot to compress the full token history, rapidly diluting stored information. Raven maintains near-perfect accuracy ($\geq \mathbf{99}\%$) up to 16K tokens and is the *only model* (in 400M scale) to retain strong performance ($> \mathbf{91}\%$) at 32K, $\mathbf{16}\times$ its training length. This is a direct consequence of sparse routing: by writing selectively, Raven prevents memory slots from being overloaded and naturally extrapolates to longer contexts without any explicit length curriculum. As shown in Figure 4, Raven achieves the strongest performance on the NIAH-1 task, attaining perfect recall compared to both linear models and softmax transformers at both parameter scales. Raven outperforms all other baselines on the NIAH (1, 2, 3) tasks at the 800M scale and demonstrates the strongest length generalization beyond its training sequence length. Furthermore, Raven surpasses even strong Transformer models such as FoX (Lin et al., 2025) on NIAH-1, owing to its exceptional length generalization ability.

Recall-Intensive Benchmarks. Beyond synthetic retrieval, we evaluate on real-world recall-heavy tasks requiring multi-hop reasoning: single-document QA (SQuAD), web data extraction (SWDE), and document-level information extraction (FDA). As shown in Table 2, Raven consistently outperforms linear-time baselines across all three. On SWDE, Raven reaches **34.1%** accuracy, surpassing Mamba-2 (25.7%) and GLA (29.0%) and narrowing the gap to the Transformer upper bound among 400M models. On FDA, Raven is the only model to exceed 22%, improving by $\sim$8 points over the best SSM baseline. These gains go beyond verbatim copying: selective forgetting preserves the semantic structure needed for multi-hop reasoning, not just token-level recall. At the 800M scale, Raven remains clearly competitive with the best linear Transformers, such as GDN. Among softmax-based Transformers, particularly FoX, these models store all tokens in memory and, unlike SSMs, do not rely on fixed-size memory. In Section 5.3, we show that hybrid models match or outperform these strong Transformers, even when replacing half of the softmax layers with linear layers.

Retrieval Without Convolutions. Many recent linear models — including GLA, Mamba-2, and GDN — rely on 1D convolutions over token features (often applied to projected $\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$ streams) for local context integration and training stability. Raven attains state-of-the-art retrieval *without them*, suggesting that input-dependent routing and per-slot decay are sufficient to address the recall bottleneck: the first controls *where* information is written, and the second controls *how long* it persists.

5.2 Language Modeling

The retrieval gains of Raven are only meaningful if they do not come at the cost of general language modeling quality. A model that achieves long-context retrieval by sacrificing compression ability would be of limited practical value. Table 3

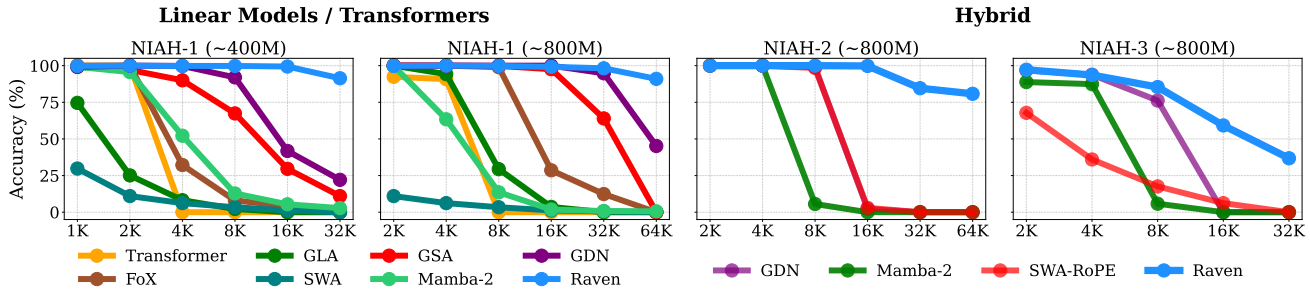


Figure 4: **NIAH Results.** ***(Left)*** NIAH-1 for 400M and 800M SSMs and transformers. ***(Right)*** NIAH-2 and NIAH-3 for 800M hybrids.

compares Raven against Mamba-2, GLA, GDN, and strong Transformer baselines including FoX (Lin et al., 2025) at both 400M and 800M parameter scales across standard zero-shot benchmarks.

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 slightly on average accuracy, though GDN uses significantly more parameters (892M vs. 792M). Across both scales, the introduction of sparse routing and input-dependent forgetting does not measurably degrade general language modeling, the two objectives are compatible, not in tension.

Raven provides a substantial improvement over SWA and related variants such as GSA, as shown in Table 2. It is the only model that both selectively forgets historical information—thereby encoding the relative positions of tokens—and selectively routes each token to specific parts of its memory through input-dependent routing.

5.3 Hybrid Raven

Hybrid architectures that interleave linear layers with attention have become a standard design pattern (Blakeman et al., 2025a), combining the inference efficiency of linear recurrences with the expressiveness of exact attention at critical layers. We evaluate Raven as a drop-in linear component paired with two full NoPE attention. We further ablate alternative configurations using RoPE as positional embeddings and SWA instead of full attention, resulting in four alternatives, which are evaluated in Table 9. In this section, we primarily focus on hybrid models using full NoPE attention, as this is the most informative configuration.

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. As shown in Table 9, hybrid models match or outperform strong Transformer baselines such as FoX (presented in Table 2) on recall-heavy tasks, particularly when evaluating beyond their training context length. Among all models, the Raven hybrid with NoPE attention performs best in this regime. Figure 4 clearly demonstrates the strength of Raven. 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.

Hybrid Raven vs. SWA-RoPE Another popular hybrid configuration combines SWA with RoPE positional embeddings with full softmax NoPE attention. In this setting, SWA-RoPE is used as the linear component for efficiency instead of SSMs. As discussed in Section 4, Raven improves on SWA in two ways: it replaces the fixed ring buffer with selective, input-dependent routing, and it replaces RoPE-based position handling with decay that implicitly encodes relative position.

These changes lead to large gains when Raven is used as the linear component in hybrid models. As shown in Table 4, the Raven hybrid consistently outperforms SWA-RoPE on both recall-intensive benchmarks and NIAH. In particular, it

Table 4: **Hybrid Models Retrieval Ability.** We evaluate GDN, Mamba-2, SWA-RoPE, and Raven as the linear component in a hybrid setting with full attention. Attention blocks use NoPE positional embeddings. We report performance on NIAH-1, NIAH-2, NIAH-3 and on highlighted SWDE, FDA, and SQuAD. The 400M models are trained on 2K sequences, and the 800M models are trained on 4K sequences. 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

substantially improves length generalization, alleviating the main limitation of SWA-based hybrids.

6 Ablations

We ablate the key design choices in Raven across four axes: architectural components, router design, memory shape, and routing sparsity. Together, these experiments validate the core design decisions and provide guidance on how to configure Raven for different settings.

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 consistently improves retrieval performance, consistent with its role in prior models (Dao and Gu, 2024; Yang et al., 2024b) (see Figure 3). For decay, the temperature-scaled sigmoid parametrization used in GLA/KDA (Team et al., 2025) underperforms Raven's decay, suggesting that Mamba-2-style input-dependent scalar decay is better suited to the routed setting. Finally, adding short-range convolutions over queries, keys, and values yields marginal gains on some benchmarks but does not change the overall picture: Raven maintains strong recall without them, unlike prior linear models (Jianlin, 2025; Yang et al., 2024b). We omit convolutions in the final design for simplicity.

6.2 Router Design

The router is the component of Raven with the least precedent in prior sequence modeling work, standard SSMs and linear transformers have no notion of input-dependent slot selection. The closest analog in the literature is the MoE router, which similarly maps a token to a sparse subset of modules. We therefore draw inspiration from MoE design choices and adapt them to the memory routing setting. Below we ablate the three main degrees of freedom: the scoring function, the use of stochastic exploration during training, and the expressiveness of the projection. Results are in Table 6, evaluated on recall-heavy benchmarks where routing quality matters most.

6.2.1 Softmax vs. Sigmoid Routing

The scoring function determines how slot scores are computed before Top-K selection. The two standard choices in MoE architectures are softmax and sigmoid.

Softmax routing normalizes scores globally across all M slots before selection:

$$\boldsymbol{m}_t = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{x}_t) \in \mathbb{R}^M. \tag{22}$$

Table 5: **Architectural Components Effect.** Ablation results of different architectural choices of Raven. Results are for 256 memory slots with Top_{32} selected by sigmoid router. Models are trained with 400M parameters on 15B tokens. **Lmb.** is lambada perplexity on evaluation set and **Avg.** shows the average performance on language modeling tasks also shown 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.** Router design effect on recall-heavy tasks. All models have 256 slots with Top_{32}. Blue highlights the default setup used for Raven. Results for 400M parameter models.

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

Global normalization means that the scores of unselected slots influence the selected weights, coupling routing decisions across the full slot set. This competitive structure is natural in MoEs — where experts should not all fire at once — but is less appropriate for memory, where independent slot assessment is preferable.

Sigmoid routing (Equation (20)) scores each slot independently (Dai et al., 2024). Slots do not compete with each other, so the router can assign high confidence to multiple slots simultaneously without suppressing the rest. Importantly, sigmoid scores are naturally sharp: many slots receive values close to zero without any explicit sparsity constraint, meaning Top-K selection mostly confirms what the router has already decided rather than imposing an artificial cutoff. Sigmoid routing outperforms softmax across most tasks (Table 6), and we adopt it as the default.

6.2.2 Gumbel Noise During Training

Without intervention, routers can collapse onto a small subset of slots early in training and never recover — a well-known failure mode in MoEs (Shazeer et al., 2017). In our setting, router collapse is particularly harmful: slots that are never selected remain uninformative, permanently reducing the effective memory capacity of the model. To prevent this, we inject Gumbel noise (Jang et al., 2016) into the router logits during training:

$$\boldsymbol{m}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \mathbf{n}), \qquad \mathbf{n} \in \mathbb{R}^M \sim \text{Gumbel}(\mathbf{0}, \mathbf{1}), \tag{23}$$

where $f(\cdot)$ is sigmoid or softmax and noise is sampled from Gumbel distribution (Gumbel, 1954). The noise encourages the router to explore all M slots during training, building informative representations across the full memory. It is removed at inference for deterministic routing. Table 6 shows consistent gains from Gumbel noise across configurations, confirming that slot exploration during training translates to better-utilized memory at test time.

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) 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

For a fixed memory budget, increasing the number of slots M requires reducing the per-slot feature dimension d, and vice versa. These two quantities play fundamentally different roles: d controls how richly each slot can represent a token, while M

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	2K	4K	8K	16K	32K	**NIAH-2** 1K	2K	4K	8K	16K	32K	**NIAH-3** 1K	2K	4K	8K	16K	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

controls how many distinct items the model can store without interference. The total memory per layer scales as $M \times d \times H$, where H is the number of heads.

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 slot granularity is the primary driver of retrieval quality: a finer-grained router has more targets to route to, which reduces the probability that a retrieval-critical token shares a slot with unrelated content and gets overwritten. Per-slot capacity, by contrast, appears to matter less, a smaller d is a mild cost compared to the benefit of increased slot count. We fix $M = 256$ as the default, as it is also consistent with previous linear models in 400M and 800M scales such as GLA,GDN, and GSA. Table 7 further shows that the strong recall performance and length generalization of Raven *hold consistently across a **wide range of memory shapes***. In particular, most memory configurations outperform linear baselines such as 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. Even without hard Top-K selection, Raven's sigmoid router already produces naturally sparse routing weights: most slot scores fall close to zero, so the effective number of meaningfully written slots is small. Top-K selection takes this further by completely zeroing out all but the K highest-scoring slots, ensuring that unselected slots are exactly frozen rather than receiving negligible but nonzero updates.

We vary K across all memory shape configurations in Table 7. Smaller K enforces stricter selectivity, reducing cross-token interference and improving recall on SWDE. Larger K transitions Raven toward dense updates, progressively recovering SSM-like behavior where every slot is touched at every step. We find that $K = 32$ (out of $M = 256$ slots, i.e., 12.5% occupancy) strikes the best balance: it preserves long-lived traces in lightly-used slots while giving the router enough flexibility to distribute general content across multiple slots when appropriate.

7 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.

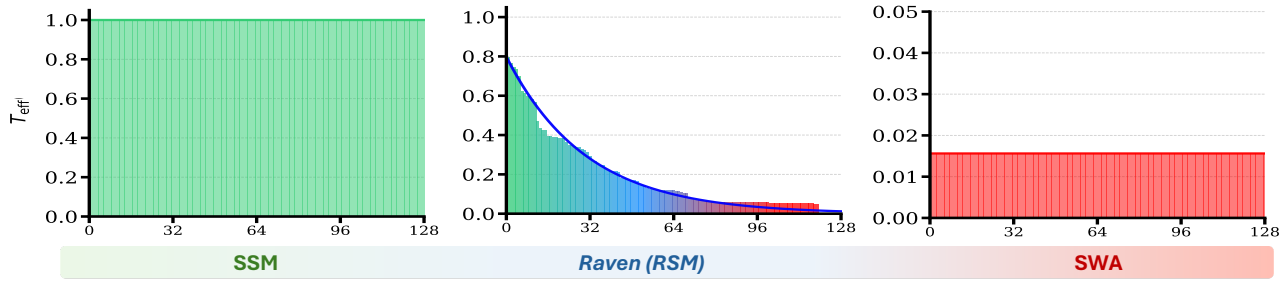


Figure 5: **Raven Effective Sequence Length.** Normalized effective sequence length for a NIAH-1 sample at sequence length 16K. SWA : each token stored in exactly one slot (FIFO). Raven : hidden state $\boldsymbol{S}_t$ for layer 1, head 1 (256 slots, Top_{32}). SSM : each token stored in all slots with decay. Slots are reordered by usage frequency; results are for 400M parameter models.

7.1 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:

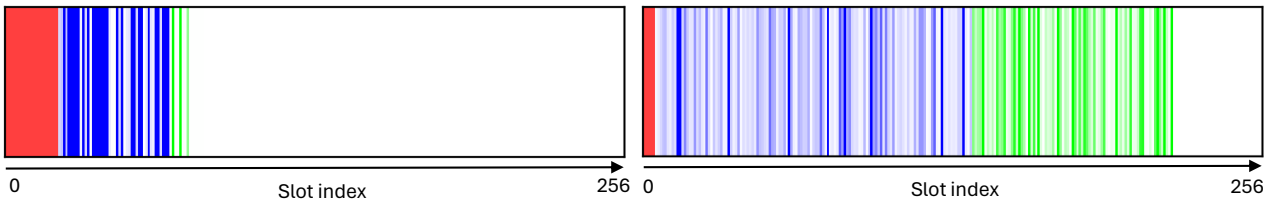
$$T_{\text{Eff}}(i) \;=\; \|\boldsymbol{r}_i\|_0 \;=\; \sum_{t=1}^{T} \mathbf{1}\big[r_{[i,t]} \neq 0\big] . \tag{24}$$

We compute the ESL per memory slot and compare Raven to SWA and SSMs in Figure 5. 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 8 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.

7.2 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 6). 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



Prompt: <BOS> SSMs often struggle on recall-heavy tasks when memory updates stay dense and uniform during training. The password is RecallSSM1378. Routing stores it in dedicated slots for recall. Now tell me the password =

Raven: RecallSSM1378

Figure 6: **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.

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.

7.3 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 7, 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.

8 Related Work

8.1 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 (4) 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.

Among these, Raven most closely resembles GSA: both are dual-state linear attention models over keys and values, with a softmax readout and normalized writes that treat memory as a stack of slots. The key distinction is that Raven *selectively writes to a subset of slots*, leaving the rest unchanged, whereas GSA updates all slots at every timestep. This selective updating is precisely what enables persistent memory across timesteps.

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.

8.2 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).

8.3 Routing and Mixture of Experts (MoE)

Routing: Routing mechanism was first introduced in *Mixture-of-Experts* (*MoE*) models (Fedus et al., 2022; Jacobs et al., 1991) to build more efficient channel mixers in larger model scales. In MoEs, the router assigns each token to a small subset of *expert* feed-forward networks (FFNs), therefore during inference only the (*selected*) experts are activated for each token which results in per-token compute efficiency especially in large model scales (i.e. Mixtral 7B (Jiang et al., 2024)). MoEs have also been used in hybrid architectures as well for the channel mixing parts as in hybrid models like Kimi-Linear (Team et al., 2025) (based on GDN) and Nemotron (Blakeman et al., 2025b) (based on Mamba-2), MoE is used instead of Gated MLP for the channel mixer. Unlike MoEs, which apply routing within their channel mixer, Raven applies routing in its sequence mixer and within its memory $\boldsymbol{S}_t$. Routing was also used in prior SSMs such as Sparse State Expansion (SSE) (Pan, 2001) which the model applies the routing into the keys vector $\boldsymbol{k}_t$ and sparsely selects parts of the key vector.

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 for avoiding 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 apply routing in the heads of the sequence mixer Raven using RSM applies routing within the memory allocation of each head of the model, 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 idea is conceptually similar to MoM, which applies routing across different linear Transformer heads. Similar to MoM, MoH also applies routing over attention 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.

9 Conclusion

We introduced Routing Matrix Mixers (RSMs), a unified framework that formalizes memory slot selection across linear sequence models, and used it to expose a fundamental gap in existing architectures: no prior model combines sparse, input-dependent routing with explicit decay. Raven fills this gap by decoupling *where* information is written from *how long* it persists, giving the model fine-grained control over memory allocation.

Raven learns to isolate retrieval-critical tokens in dedicated slots and naturally induces variable effective sequence lengths across memory. Relative to SWA, it improves two key aspects: content-dependent memory allocation through routing, and selective decay instead of position-based eviction. Relative to dense SSMs, it writes sparsely and selectively, leaving unselected slots unchanged rather than updating the entire state at every step. This allows dedicated retrieval slots to persist with less interference.

These properties let Raven extrapolate to contexts more than $16\times$ longer than those seen during training, without relying on additional components such as short-range convolutions. On recall-intensive benchmarks, Raven consistently outperforms dense SSMs while remaining competitive on general language modeling, and these gains carry over to hybrid architectures paired with both full attention and SWA.

More broadly, the RSM framework suggests that routing and forgetting are orthogonal design axes that existing models have largely conflated. We hope this perspective opens new directions for sequence models that treat memory allocation as a first-class design choice rather than a byproduct of decay.

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. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025a.

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*, 2025b.

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.

Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

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. Latest advancements. 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
$\boldsymbol{A}_t$	Vector decay
$\boldsymbol{r}_t$	Router
$\sigma(.)$	Sigmoid function
$\phi(.)$	Non-linear function
$\odot$	Hadamard product

A.2 Slot Separability of Linear Updates

Claim. 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

$$\boldsymbol{S}_t \;=\; \boldsymbol{D}_t \boldsymbol{S}_{t-1} \boldsymbol{A}_t \;+\; \boldsymbol{U}_t. \tag{25}$$

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 $k \neq i$. Considering $\Delta \in \mathbb{R}^{1\times d}$ satisfy $\Delta \boldsymbol{A}_t \neq \boldsymbol{0}$ as $\boldsymbol{A}_t \neq 0$. Considering 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 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 ∼1.3B parameters. All models are trained on SlimPajama-627B (Soboleva et al., 2023). The 400M models are trained for 15B tokens. 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 training sequence length for the 400M models is set to 2048. The learning rate follows a cosine scheduler with 1B tokens of warmup. We utilize the `flash-linear-attention`[2] repository for baselines as well as our codebase.

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

A.3.2 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.4 Raven Recurrence

Raven Recurrence:

$$\boldsymbol{m}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t), \quad \boldsymbol{g}_t = \begin{cases} \boldsymbol{m}_t[i] & i \in \mathrm{Top}_K(\boldsymbol{m}_t) \\ 0.0 & o.w \end{cases}, \boldsymbol{r}_t = \frac{\boldsymbol{g}_t}{\alpha \sum_{i=1}^{M} \boldsymbol{g}_t[i]}, \quad a_t = -\mathrm{SoftPlus}(\boldsymbol{w}^\top \boldsymbol{x}_t).\exp(\Delta),$$

$$\boldsymbol{S}_t^k = \exp(a_t \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^k + (\boldsymbol{1} - \exp(a_t \boldsymbol{r}_t))\boldsymbol{k}_t^\top, \quad \boldsymbol{S}_t^v = \exp(a_t \boldsymbol{r}_t) \odot \boldsymbol{S}_{t-1}^v + (\boldsymbol{1} - \exp(a_t \boldsymbol{r}_t))\boldsymbol{v}_t^\top.$$

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

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 $\boldsymbol{G}_t \in \mathbb{R}^{d \times M}$ 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 $\boldsymbol{A}_t = \text{diag}(\boldsymbol{a}_t)$, 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):

$$\boldsymbol{O} = \left(\left((\boldsymbol{Q} \odot \boldsymbol{B}) \left(\frac{\boldsymbol{K}}{\boldsymbol{B}} \right)^{\top} \right) \odot \boldsymbol{M} \right) \boldsymbol{V}, \tag{26}$$

where $\boldsymbol{M} \in \mathbb{R}^{T \times T}$ is a binary causal mask. Since Equation (26) 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 (26) 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 $\text{GLA}(\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{G}_t)$ 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:

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

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 design the standard 400M parameter model in the literature of SSMs (Movahedi et al., 2025; Yang et al., 2023, 2024a,b). We use 4 heads per each block and two hidden states for $\boldsymbol{S}_t^v, \boldsymbol{S}_t^k$ for keys and values per-head. Also, we used a channel mixer after each RSM block with skip connections (He et al., 2016) similar to transformers and SSMs. We used QK-Normalization and applied output gating for the RSM block. We used Llama (Dubey et al., 2024) tokenizer 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, 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 achieves near-perfect NIAH-1 accuracy across the full 1K–32K range, while

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**

GDN+SWA degrades steadily and Mamba-2+SWA collapses to near zero beyond 4K. On recall-intensive benchmarks, Raven leads on SWDE, FDA, and SQuAD in both hybrid configurations, confirming that its advantages are architectural rather than dependent on a particular hybrid pairing.

Initialization. As Raven readout uses the softmax function, $\boldsymbol{o}_t = (\boldsymbol{S}_t^v)^\top \mathrm{softmax}\big(\boldsymbol{S}_t^k \boldsymbol{q}_t\big)$, 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

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top. \tag{27}$$

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

$$\mathcal{L}(\boldsymbol{S}) = -\langle \boldsymbol{S}\boldsymbol{k}_t,\, \boldsymbol{v}_t \rangle, \qquad \boldsymbol{S}_t = \boldsymbol{S}_{t-1} - \nabla_{\boldsymbol{S}} \mathcal{L}(\boldsymbol{S}), \tag{28}$$

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 $\exp(a_t \boldsymbol{r}_t)$, shown in blue. 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:

$$\mathcal{L}(\boldsymbol{S}) = -||\boldsymbol{S}\boldsymbol{k}_t - \boldsymbol{v}_t||_F^2, \tag{29}$$

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.*

Table 10: **Table 7:** 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 from 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	$-(1-\exp(a_t\boldsymbol{r}_t))\langle \boldsymbol{S}_{t-1}^\top \boldsymbol{k}_t,\, \boldsymbol{v}_t\rangle \;+\; \frac{1}{2}\left\|\sqrt{\exp(a_t\boldsymbol{r}_t)}\,\boldsymbol{S}_{t-1}\right\|_F^2$	$\boldsymbol{S}_t = \mathrm{Diag}(\exp(a_t\boldsymbol{r}_t))\,\boldsymbol{S}_{t-1} + \boldsymbol{k}_t\boldsymbol{v}_t^\top$

A.8 Memory Dynamic Visualizations

We visualize the memory dynamics of Raven. As shown in Figure 8, 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 7 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 6 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.

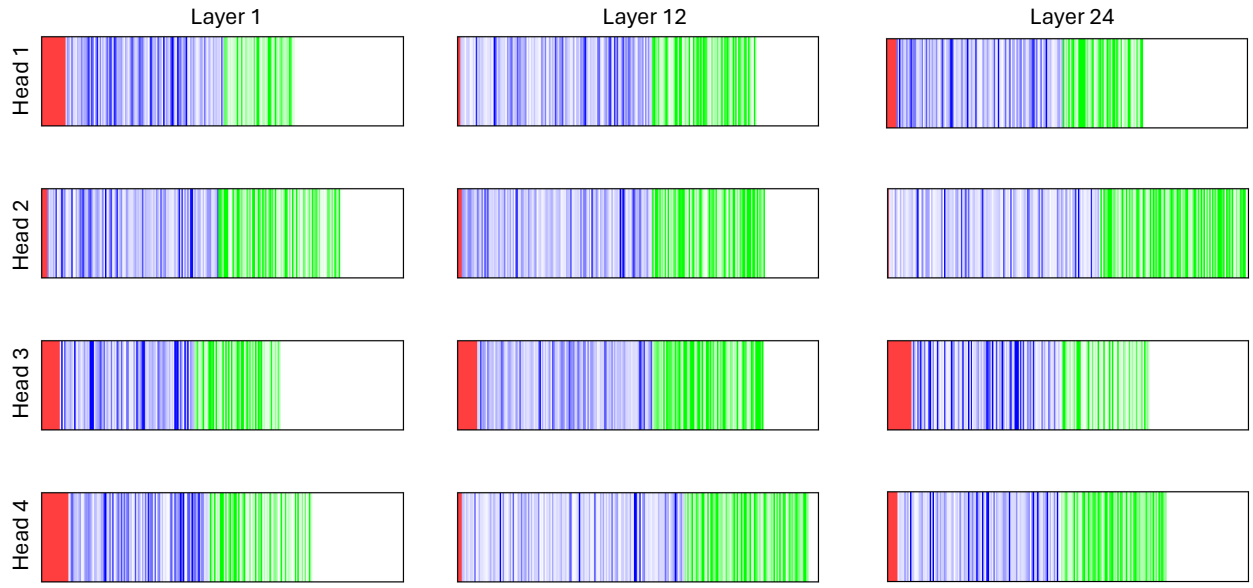


Figure 7: **Memory Visualization.** Memory of Raven while being asked the password within the prompt. The figures show the hidden state of Raven $\boldsymbol{S}_t$ at the end of processing the prompt and answering to the question (time point $t = L$) for **(Top)** Layer 24, **(Middle)** Layer 12, **(Bottom)** 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.

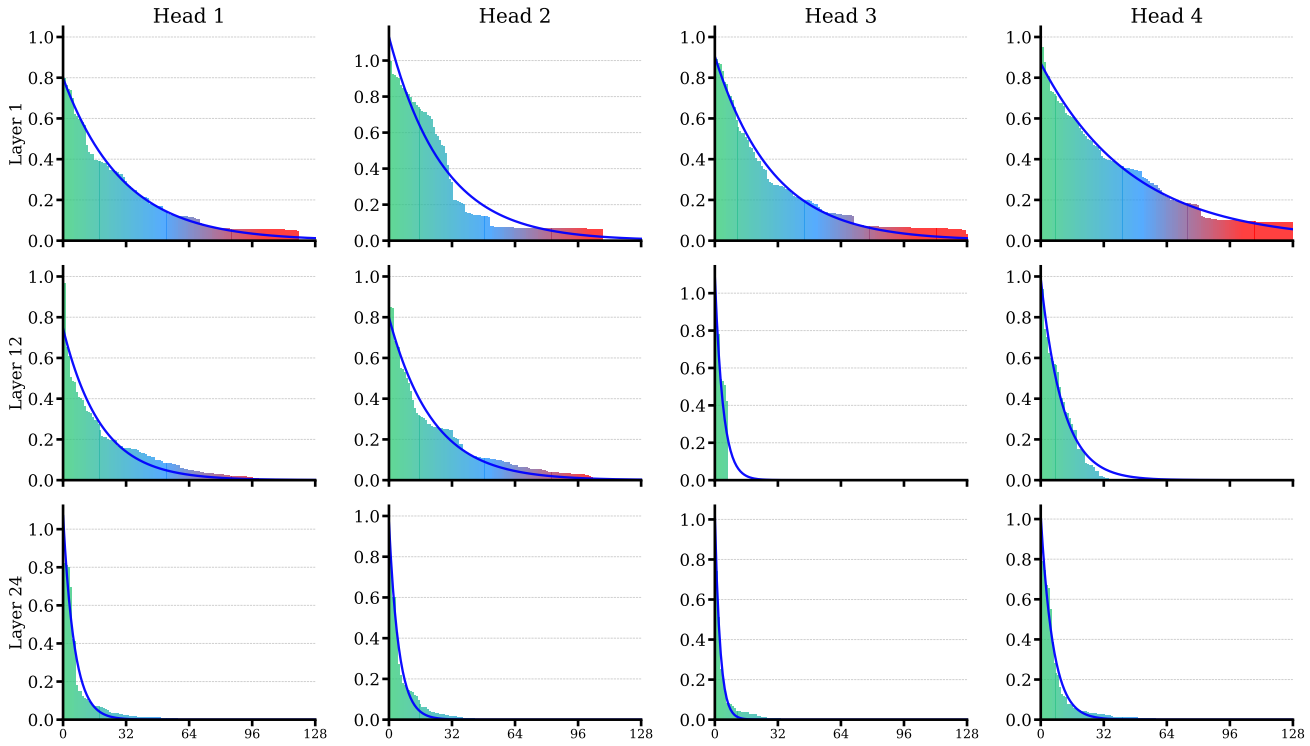


Figure 8: **Effective Sequence Length of Raven.** The normalized *effective sequence length* for a NIAH-1 sample with a sequence length of 16K. Results are from first, 12th and last (24th) layer for first, second and forth head of *400M* model. Slots are reordered from highest to lowest.